

Module 8 - Échanges de données

Première partie : GET et POST

Pour tester rapidement du code PHP : <http://sandbox.onlinephpfunctions.com/>

Exercice 1 : GET et révision du Javascript orienté objet

Certains sites éducatifs proposent à leurs visiteurs de tester leurs connaissances en répondant à des quizz comportant plusieurs questions à choix multiples. Pour rendre ces quizz plus attractifs, certains sites ont eu l'idée de présenter les réponses sous la forme de boutons qui se déplacent sur l'écran, mêlant test de connaissances et agilité à la souris !

Cet exercice vous propose de programmer ces « boutons volants » du côté Javascript, d'envoyer au serveur la réponse choisie par l'utilisateur via la méthode GET et de laisser le serveur vérifier s'il s'agit d'une réponse correcte ou pas. Dans le cadre de cet exercice, on se contentera de tester les tables de multiplication des nombres entre 2 et 10 en posant une question générée aléatoirement par Javascript.

Étape 1

Du côté HTML d'abord, créez une page comportant un paragraphe suivi d'un div vide de 600 pixels de large sur 400 pixels de haut puis d'un second paragraphe indiquant « Cliquez sur la réponse correcte ! »

Prévoyez un identificateur pour le premier paragraphe et pour le div afin de pouvoir y faire référence dans le code Javascript. Utilisez des règles CSS pour donner une bordure au div, préciser ses dimensions et le mettre en positionnement relatif (ce qui est nécessaire pour pouvoir déplacer des boutons à l'intérieur et qui correspond à la propriété CSS « position: relative »).

Étape 2

Ajoutez un script Javascript qui se déclenchera une fois la page chargée.

Ce script devra :

- déterminer aléatoirement deux entiers entre 2 et 10 ;
- écrire la question « Que vaut [nombre1] x [nombre2] ? » dans le premier paragraphe ;
- ajouter des boutons contenant les réponses proposées dans le div (à ajouter plus tard).

Étape 3

Pour la gestion des boutons volants, créez un fichier de code Javascript séparé.

Ce fichier comportera une fonction constructrice appelée FlyingButton. Pour rappel, tout objet Javascript créé via un « new FlyingButton(...) » aura automatiquement comme prototype l'objet FlyingButton.prototype ; c'est donc dans cet objet que vous devrez placer les méthodes partagées par tous les boutons volants (ainsi que les attributs et méthodes « de classe »).

Pour vous aider à vous souvenir du fonctionnement de l'orienté objet en Javascript, la construction de ce fichier sera décomposé en plusieurs étapes.

Définissez une fonction constructrice `FlyingButton`. Lors de la création d'un bouton volant, on précisera deux arguments : le texte du bouton et la fonction à déclencher en cas de clic.

En plus de stocker les arguments reçus dans des attributs sur l'objet créé, la fonction constructrice devra également fabriquer un nouvel élément DOM correspondant au bouton, et conserver une référence vers cet élément DOM sous la forme d'un attribut (par exemple `this.elem`). En détails :

- créez l'élément via `document.createElement` ;
- ajoutez-lui la classe « `flyingButton` » (à définir dans le document HTML pour pouvoir modifier l'apparence des boutons) ;
- placez le texte à afficher à l'intérieur du bouton (via `innerHTML`) ;
- liez à l'événement « `onclick` » de l'élément la fonction à déclencher en cas de clic ;
- ajoutez l'élément dans le div (via `appendChild`).

Testez votre code en ajoutant manuellement quelques boutons dans le script Javascript du fichier principal (en utilisant la fonction constructrice).

Modifiez la définition de classe CSS `flyingButton` pour qu'elle impose une largeur minimale de 60 pixels et une hauteur minimale de 30 pixels (ajoutez d'autres propriétés selon vos désirs).

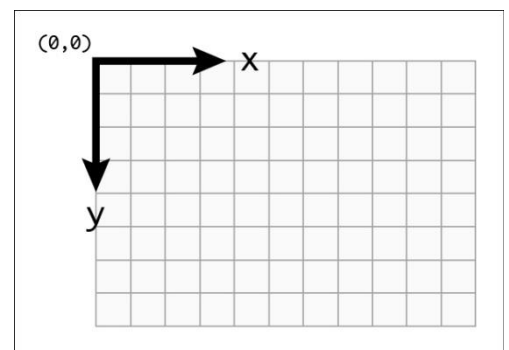
Étape 4

Il est temps de s'occuper du positionnement et du déplacement des boutons à l'intérieur du div.

La position d'un bouton volant sera repérée par deux coordonnées (en X et en Y) et son déplacement sera déterminé par deux vitesses (vitesse en X et vitesse en Y). Le mouvement sera simulé ainsi : si, à un moment, le bouton est positionné en (30,50) et a une vitesse de (2,-1), à chaque itération d'une boucle, le bouton se déplacera de 2 pixels vers la droite et de 1 pixel vers le haut, passant successivement aux positions (32,49), (34,48), (36,47), etc.

Les quatre attributs à ajouter aux boutons volants seront déterminés aléatoirement (voir `Math.random`) lors de la création de ceux-ci :

- la position en x du coin supérieur gauche du bouton sera un entier déterminé aléatoirement et compris entre 0 et `600 - this.elem.clientWidth` (en supposant que `this` se réfère au bouton, `this.elem` cible l'élément DOM représentant le bouton et `this.elem.clientWidth` indique sa largeur en pixels) ;
- la position en y du coin supérieur gauche du bouton sera un entier déterminé aléatoirement et compris entre 0 et `400 - this.elem.clientHeight` ;
- les vitesses en x et en y seront un entier déterminé aléatoirement et compris entre -3 et 3.



Simplement modifier les attributs « posX » et « posY » (ou quels que soient les noms que vous leur avez donnés) d'un bouton volant ne suffit pas à positionner le bouton dans le div : il faut également communiquer la position attendue à l'élément DOM qui représente le bouton en question !

Concrètement, pour positionner précisément un bouton à l'intérieur d'un div, il faut :

- s'assurer que le div a bien la propriété CSS « position : relative » ;
- donner au bouton la propriété CSS « position : absolute » (utilisez le style CSS créé pour les boutons volants) ;
- indiquer la position via les propriétés CSS left (décalage par rapport au bord gauche) et top (décalage par rapport au bord supérieur).

Pour modifier les propriétés left et top d'un élément DOM appelé elemDOM via Javascript, on utilise des instructions telles que :

```
elemDOM.style.top = valeur + "px";  
elemDOM.style.left = valeur + "px";
```

Conseil . Pour vous faciliter la tâche, définissez une méthode setPosition(posX,posY) disponible sur tous les boutons volants et s'occupant de mettre à jour non seulement les attributs indiquant la position mais également la position de l'élément DOM associé au bouton volant. Elle vous sera également utile pour coder le déplacement du bouton !

Étape 5

Ajoutez à chacun des boutons créés par la fonction constructrice une méthode move() qui s'occupera d'effectuer un déplacement.

Dans les cas simples, il suffira d'ajouter la vitesse à la position pour obtenir la nouvelle position (à la fois en x et en y), puis d'appeler la méthode setPosition pour conclure.

Cependant, il faut également tenir compte des cas où le bouton sortirait du cadre du div. Cela peut se faire assez facilement en implémentant un mécanisme de « rebond » (le bouton semble rebondir contre le bord du div et repartir en sens inverse) grâce à l'algorithme suivant.

Soit posX' la nouvelle position en X.

Si $\text{posX}' < 0$ (sortie à gauche) ou $\text{posX}' + \text{largeur du bouton} > 600$ (sortie à droite)

changer le signe de la vitesse en X (pour repartir dans l'autre sens) et recalculer

Soit posY' la nouvelle position en Y.

Si $\text{posY}' < 0$ (sortie en haut) ou $\text{posY}' + \text{hauteur du bouton} > 400$ (sortie en bas)

changer le signe de la vitesse en Y (pour repartir dans l'autre sens) et recalculer

Pour faire bouger les boutons volants, il suffit donc d'appeler la méthode move() sur chacun des boutons créés. Pour que cela soit possible, il faut conserver quelque part la

liste des divers boutons créés. Pourquoi ne pas conserver cette liste sous la forme d'un tableau qui serait un attribut de l'objet « FlyingButton » (qui est également la fonction constructrice) ?

Juste après la définition de la fonction constructrice, ajoutez

```
|| FlyingButton.buttons = [];
```

et, dans la définition de la cette même fonction constructrice, ajoutez l'instruction suivante, qui a pour effet d'ajouter « this » (donc, le nouvel objet créé) à la fin du tableau.

```
|| FlyingButton.buttons.push(this);
```

Définissez une fonction qui lancera la méthode `move()` sur chacun des boutons de ce tableau (une bonne occasion d'utiliser `forEach`). Tant qu'à faire, définissez cette fonction comme un attribut de l'objet `FlyingButton`, de sorte que celle-ci puisse être déclenchée par l'appel suivant.

```
|| FlyingButton.moveAll();
```

Puis, finalement, définissez une fonction qui se chargera d'exécuter `moveAll` toutes les 30 millisecondes. Faites en sorte qu'on puisse l'appeler simplement via la commande suivante.

```
|| FlyingButton.startMove();
```

Étape 6

Revenez au fichier principal et à son script Javascript. Modifiez celui-ci pour qu'il accomplisse les actions suivantes.

1. Choisir aléatoirement deux entiers entre 2 et 10 et mettre à jour le premier paragraphe (déjà codé normalement).
2. Ajouter quatre boutons dans le div, chacun portant comme texte « [nombre1] x [nombre2] = [valeur] »
 - a. avec [valeur] = le produit des deux nombres pour le 1^{er} bouton ;
 - b. avec [valeur] = la somme des deux nombres pour le 2^e bouton ;
 - c. avec [valeur] = le produit des deux nombres + 2 pour le 3^e bouton ;
 - d. avec [valeur] = le produit des deux nombres + 6 pour le 4^e bouton ;
3. S'arranger pour que, en cas de clic sur un des boutons, on se redirige vers la page « `verification.php` » en passant, selon la méthode GET, les arguments suivants :
 - a. nombre1 : le premier nombre ;
 - b. nombre2 : le second nombre ;
 - c. reponse : la réponse choisie.

Étape 7

Créez un fichier « `verification.php` » qui

- affichera « Vous ne pouvez pas accéder directement à cette page. » si on ne lui transmet pas en méthode GET des valeurs pour nombre1, nombre2 et reponse ;
- affichera « Réponse correcte ! » si `reponse = nombre1 * nombre2` ;
- affichera « Réponse incorrecte ! » sinon.

Exercice 2 : Formulaires et validation

Dans le cadre de cet exercice, vous réaliserez un formulaire permettant à un utilisateur d'entrer son nom et sa date de naissance. Une fois les données envoyées au serveur, celui-ci produira une page reprenant les futurs anniversaires de la personne en question et affichant un certain nombre d'informations choisies par l'utilisateur.

Étape 1

Tout d'abord, créez un formulaire HTML dont la structure ressemble à celui qui est présenté ici (choisissez vos propres styles).

Le diagramme illustre la structure d'un formulaire HTML, divisé en trois sections distinctes par des barres de titre orange. La première section, intitulée "Identification", contient un champ de saisie pour le "Prénom". La deuxième section, intitulée "Date de naissance", comprend un menu déroulant pour le "Jour de naissance" (pré-rempli avec "1"), une liste à puces pour le "Mois de naissance" (avec "janvier" sélectionné par défaut), et un champ de saisie pour l'"Année de naissance" (pré-rempli avec "2000"). La troisième section, intitulée "Colonnes à afficher", propose trois options à cocher : "dates d'anniversaire", "jours d'anniversaire" et "âge". À la base du formulaire se trouve un bouton "Afficher les anniversaires !".

Quelques consignes :

- Le formulaire devra être découpé en trois sections « fieldset ».
- Les étiquettes telles que « Prénom », « janvier », « dates d'anniversaire »... seront produites via la balise `<label>` et liée à l'élément associé via un attribut « for » (voir slides du cours de HTML/CSS).
- Pour le jour de naissance, on utilisera une liste déroulante (balise `<select>`).

- Pour les mois et les colonnes à afficher, on présentera les options dans une liste .
- L'année de naissance sera introduite dans un champ <input> de type numérique.
- N'oubliez pas d'associer à tous les éléments du formulaire un champ « name » adéquat.
- La date choisie par défaut est le 1^{er} janvier 2000.
- Pour les affichages répétitifs, vous pouvez utiliser du code Javascript (par exemple placer les noms des mois dans un tableau et afficher la liste des options via une boucle).
- Choisissez la méthode d'envoi POST pour le formulaire. Dans un premier temps, vous pouvez utiliser comme cible l'adresse « vm-debian.iesn.be/ig40/showGetPost.php » qui se contente d'afficher les données GET et POST reçues pour vérifier que vous n'avez pas oublié de champ.

Étape 2

Pour l'instant, il n'y a aucune validation des informations rentrées par l'utilisateur. HTML5 peut se charger de certaines vérifications de manière automatique.

Modifiez le code HTML afin que

- le prénom soit un champ obligatoire (attribut required) et
- l'année de naissance doive être comprise entre 1900 et 2050 (attributs min et max).

Vérifiez ces validations.

Étape 3

Pour des validations plus fines, il faut utiliser Javascript. Créez une fonction dateValide() qui renvoie vrai si la date choisie est valide et faux sinon.

Pour ce faire, vous pouvez commencer par travailler sur des fonctions annexes (comme une fonction indiquant si une année est bissextile ou non, une fonction renvoyant le numéro du dernier jour pour un mois et une année donnée et une troisième fonction vérifiant si un jour, un mois et une année donnée forment une date valide).

Dans le code HTML, à droite de la liste déroulante permettant de choisir le jour, ajoutez une balise span vide telle que

```
<span id="commentaire"></span>
```

Modifiez le code de la fonction dateValide() pour qu'en plus de renvoyer un booléen, elle affiche un commentaire sur la date actuellement entrée dans ce span. Par exemple, en cas d'erreur, elle pourrait afficher « Cette date n'est pas valide ! » (et ne rien afficher / vider le span si la date est valide).

Manuellement ou à l'aide d'un script Javascript, faites en sorte que la fonction dateValide soit exécutée à chaque modification d'un des champs du fieldset « Date de naissance » (c'est-à-dire en cas de modification du jour, du mois ou de l'année). Pour cela, utilisez l'événement « onchange ».

Notez que, pour le moment, la date est vérifiée à chaque modification mais que rien n'empêche l'utilisateur de cliquer sur « Afficher les anniversaires ! » en cas de date incorrecte.

Via Javascript, dans une fonction qui sera exécutée lorsque la page sera entièrement chargée, associez la fonction `dateValide` à l'événement `onclick` du bouton d'envoi. Observez l'effet.

Étape 4

Rédigez maintenant le script PHP censé recevoir les informations envoyées par cette page HTML. Dans un premier temps, vous pouvez vous contenter de vérifier les informations reçues via la méthode POST et de les afficher.

Étape 5

Peaufinez l'affichage produit par le script PHP de sorte que, si les données envoyées sont valides, il affiche « Cher <prénom>, voici vos futurs anniversaires. » suivi d'un tableau contenant, pour chacune des années à venir, les colonnes suivantes :

1. une colonne reprenant l'année (de 2015 à 2025) ;
2. si l'utilisateur a coché l'option « dates d'anniversaire », une colonne reprenant la date d'anniversaire au format JJ/MM/AAAA ;
3. si l'utilisateur a coché l'option « jours d'anniversaire », une colonne reprenant les jours correspondant à ses futurs anniversaires (c'est-à-dire lundi, mardi, mercredi, jeudi, vendredi, samedi ou dimanche) ;
4. si l'utilisateur a coché l'option « âge », une colonne indiquant l'âge qu'il atteindra lors de chacun de ses futurs anniversaires.

Pour le calcul du jour (colonne 3), référez-vous au site php.net (et plus particulièrement à la page <http://php.net/manual/en/function.date.php>) pour voir comment l'obtenir facilement.

Deuxième partie : Cookies & sessions

Pour tester rapidement du code PHP : <http://sandbox.onlinephpfunctions.com/>

Exercice 1 : Cookies colorés

Dans le cadre de cet exercice, vous allez utiliser des cookies pour gérer les préférences d'un utilisateur en matière de couleurs.

Étape 1

Dans un premier temps, créez une page HTML avec le code suivant.

```
<!doctype HTML>
<html>
  <head>
    <meta charset="utf-8" />
    <style>
      body {
        background-color: black;
      }
      #wrapper {
        width: 800px;
        padding: 8px;
        margin-left: auto;
        margin-right: auto;
        background-color: lightgrey;
      }
      #info {
        display: flex;
        flex-direction: row;
        align-items: stretch;
        border: 1px solid black;
        padding: 4px;
      }
      #info div {
        border: 1px solid blue;
        background-color: lightblue;
        padding: 6px;
        margin: 6px;
        flex-grow: 1;
      }
      #pCookies {
        height: 100px;
        padding: 4px;
        border: 1px solid grey;
        background-color: white;
      }
      #main {
        padding: 6px;
        color: black;
      }
```



```

        background-color: white;
    }
</style>
</head>
<body>
    <div id="wrapper">
        <p id="main">
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
            Phasellus pellentesque lorem augue, in hendrerit tortor fermentum at.
            In hac habitasse platea dictumst. Praesent nec iaculis magna, in
            lacinia libero. Nulla vel tincidunt tellus, in varius nibh. Donec ut
            odio ac odio malesuada auctor nec at nisi. Sed lobortis nibh quis erat
            laoreet luctus. Interdum et malesuada fames ac ante ipsum primis in
            faucibus. Ut auctor dignissim urna, nec aliquet leo aliquam at.
            Integer eu lectus posuere, accumsan enim sit amet, placerat enim. Duis
            in metus nisl. Maecenas consequat velit id turpis sagittis viverra.
            Proin et ligula ut eros fermentum gravida nec eu nulla.
        </p>
        <div id="info">
            <div>
                <form action="" method="POST">
                    <label for="cTexte">Couleur du texte :</label><br/>
                    <input type="text" id="cTexte" name="cTexte" /><br/>
                    <label for="cFond">Couleur du fond :</label><br/>
                    <input type="text" id="cFond" name="cFond" />
                    <p><button id="bMAJ">Mise à jour</button></p>
                </form>
            </div>
            <div>
                Vos cookies :
                <p id="pCookies"></p>
                <button id="bRafraichir">Rafraîchir</button>
            </div>
            <div>
                <button id="bSupprimer">Supprimer les cookies</button><br/>
                <button id="bRecharger">Recharger la page</button>
            </div>
        </div>
    </div>
</body>
</html>

```

Observez le contenu de la page, tant au niveau du code HTML qu'au niveau de son apparence.

Étape 2

Ajoutez des scripts Javascript pour associer des actions aux boutons.

Commencez par le bouton « bMAJ » qui est censé ajouter deux cookies appelés « cTexte » et « cFond » et contenant les couleurs de texte et de fond entrés par l'utilisateur. Pour rappel, vous pouvez obtenir la valeur d'un champ `<input>` via une expression similaire à la suivante.

```
|| document.getElementById("idInput").value
```

Continuez avec le bouton « bRafrachir » qui est censé afficher dans le paragraphe « pCookies » le contenu des cookies.

Finalement, occupez-vous du bouton « bSupprimer » qui devrait vider tous les cookies actuels. Pour ce faire, en Javascript, vous devez redéfinir les cookies un par un en leur associant une date d'expiration antérieure au moment présent. Cela peut se faire comme suit.

Pour obtenir un moment correspondant à « avant maintenant » :

```
|| var moment = new Date ();           // correspond à maintenant
|| moment.setTime(moment.getTime() - 1000);
||                                     // il y a 1000 millisecondes.
```

Pour préciser la date d'expiration d'un cookie en Javascript :

```
|| document.cookie = "nom=valeur;expires=" + moment.toUTCString();
```

Testez les fonctionnalités que vous venez de coder.

Étape 3

Pour l'instant, vous avez effectué vos tests localement. Téléchargez le document HTML sur le serveur et ouvrez-le à nouveau.

Notez que, la première fois que vous ouvrez le document, il n'y a aucun cookie (c'est normal : les précédents cookies étaient associés au site « local »).

Enregistrez des couleurs pour le texte et pour le fond et observez que les cookies s'enregistrent bien cette fois-ci. Rechargez la page et cliquez sur « Rafrâchir » pour observer que les cookies persistent d'une session à l'autre (à moins que vous ne leur ayez donné des dates d'expiration très proches).

Sous Firefox, cliquez sur le bouton du menu principal (les trois barres tout à droite) puis choisissez l'option « Options ». Cliquez sur l'onglet « Privacy » et, dans la section « History », choisissez l'option « Use custom settings for history ».

Un peu plus bas, cliquez sur « Show cookies » puis recherchez le site « vm-debian.iesn.be » et observez que les deux cookies sont bien sauvegardés à cet endroit.

Étape 4

Transformez le fichier .html en un fichier .php et ajoutez-y quelques scripts PHP.

Le premier script se trouvera au début du document et initialisera les variables \$cTexte et \$cFond. Les valeurs par défaut seront respectivement « black » et « white » mais, si le site reçoit un cookie définissant cTexte (ou cFond), la valeur du cookie devra être utilisée à la place de la valeur par défaut.

Les deux autres petits scripts écriront simplement les valeurs de \$cTexte et de \$cFond au bon endroit dans la définition du style associé à l'identificateur #main.

Mettez le document à jour sur le serveur et testez ses fonctionnalités. Observez qu'en cas de mise à jour des couleurs, quand la page est rechargée, les nouvelles couleurs sont immédiatement utilisées. **Assurez-vous de bien comprendre la manière dont les données des nouvelles couleurs sont transmises.**

Codez également une fonction Javascript associée au bouton bRecharger, qui se contentera de recharger la page, ce qui peut se faire simplement via la méthode

```
|| window.location.reload();
```

Étape 5

Jusqu'ici, toutes les modifications apportées aux cookies étaient réalisées localement via Javascript. **Dans cette étape, vous allez utiliser PHP pour donner l'ordre de modification des couleurs.**

Modifiez le premier script PHP pour qu'il teste tout d'abord s'il a reçu une valeur pour cTexte et pour cFond via la méthode POST et, si c'est le cas, pour qu'il donne l'ordre de conserver ces valeurs dans le cookie. Concrètement, cela revient à ajouter les lignes suivantes.

```
|| if (isset($_POST["cTexte"]) && isset($_POST["cFond"]))  
|| {  
||     setcookie('cTexte', $_POST['cTexte'], time() + 60*60*24*7);  
||     setcookie('cFond', $_POST['cFond'], time() + 60*60*24*7);  
|| }
```

Supprimez également l'action Javascript associée au bouton « bMAJ ». Au lieu d'exécuter un script, en cas de clic sur le bouton, il faudra envoyer les données du formulaire via POST à la page PHP. Pour ce faire, assurez-vous de

- a) placer l'attribut method="POST" dans la balise <form> ;
- b) placer un attribut action="..." ciblant la page PHP dans la balise <form> ;
- c) supprimer l'attribut onclick du bouton « bMAJ » ;
- d) **ajouter l'attribut type="submit" au bouton « bMAJ ».**

Rechargez la page. Puis modifiez les couleurs. Observez que l'effet n'est plus instantané mais qu'il faut charger la page une seconde fois pour qu'il prenne effet. Pourquoi ?

Modifiez le code PHP pour que l'effet d'un changement soit instantané.

Exercice 2 : dans mon panier j'ai emmené...

Le but de cet exercice est de réaliser une page web dynamique permettant à un client de choisir divers articles à acheter. Le contenu de son panier sera conservé sur le serveur, en utilisant des données de session.

Dans le cadre de cet exercice, tout le code sera concentré sur une page unique et les actions choisies par l'utilisateur seront transmises via la méthode GET, c'est-à-dire dans l'URL de connexion.

Étape 1

Créez tout d'abord le canevas de votre page au format HTML. Celle-ci sera composée de trois parties : (1) une en-tête contenant les boutons d'actions principaux, (2) une partie principale où on affichera les messages ainsi que la liste des articles disponibles et (3) une section où le contenu actuel du panier de l'utilisateur sera présenté.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Achetez nos jeux !</title>
  </head>
  <body>
    <div id="wrapper">

      <!-- EN-TETE -->
      <header>
        <button id="bRPG">Jeux de rôle</button>
        <button id="bNonRPG">Autres jeux</button>
        <button id="bCommande">Passer commande</button>
        <button id="bEmpty">Vider votre panier</button>
      </header>

      <!-- MESSAGE ET LISTE DES ARTICLES -->
      <div id="main">
        <div id="items">
        </div>
      </div>

      <!-- PANIER -->
      <div id="basket">
        <h2>Votre panier :</h2>
        <div id="basketContents">
        </div>
      </div>

    </div>
  </body>
</html>
```

Pour améliorer l'esthétique, vous pouvez utiliser les styles suivants (ou créez les vôtres).

```
body {
  background-color: black;
  font-family: "comic sans MS";
  font-size: 80%;
}
#wrapper {
  width: 800px;
  background-color: beige;
  padding: 10px;
}
header {
```

```

background-color: orange;
padding: 10px;
margin-bottom: 10px;
display: flex;
justify-content: space-around;
}
header button {
width: 150px;
}
#main {
text-align: center;
}
#main p {
font-weight: bold;
}
#main table {
margin-left: auto;
margin-right: auto;
}
#basket {
margin-top: 6px;
padding: 6px;
border: 2px solid orange;
}
#basket h2 {
margin-top: 0;
border-bottom: 2px solid darkorange;
}
#items tr:hover {
background-color: orange;
}

```

Voici un exemple de résultat final attendu :

Jeux de rôle
Autres jeux
Passer commande
Vider votre panier

Jeu	Prix	Genre	Editeur	
Planescape: Torment	9.09 €	RPG	Black Isle Studios	<button>Commander</button>
Neverwinter Nights 2 Complete	22.78 €	RPG	Obsidian Entertainment	<button>Commander</button>
Legend of Grimrock II	21.79 €	RPG	Almost Human	<button>Commander</button>
Shadowrun returns	13.69 €	RPG	Harebrained Schemes	<button>Commander</button>
Pillars of Eternity: Hero Edition	41.99 €	RPG	Obsidian Entertainment	<button>Commander</button>
Arx Fatalis	5.49 €	RPG	Arkane Studios	<button>Commander</button>
Wasteland 2: Director's Cut	39.99 €	RPG	inXile Entertainment	<button>Commander</button>
Gothic 3: Enhanced Edition	9.09 €	RPG	Piranha Bytes	<button>Commander</button>
Baldur's Gate II: Enhanced Edition	18.19 €	RPG	Beamdog	<button>Commander</button>

Votre panier :

- Shadowrun returns (13.69€)
- Legend of Grimrock II (21.79€)
- Neverwinter Nights 2 Complete (22.78€)

Total : 58.26€

Étape 2

Après avoir observé le contenu et la structure de la page HTML, sauvez-la avec l'extension PHP sur le serveur. Dans la suite de cet énoncé, cette page sera appelée `achats.php`.

Créez également deux autres fichiers textes appelés `rpg.json` et `nonrpg.json` contenant le code suivant. Il s'agit de fichiers de données au format JSON reprenant les articles disponibles.

Pour `rpg.json` :

```
{
  "Planescape: Torment" : {
    "price" : 9.09,
    "genre" : "RPG",
    "editor" : "Black Isle Studios"
  },
  "Neverwinter Nights 2 Complete" : {
    "price" : 22.78,
    "genre" : "RPG",
    "editor" : "Obsidian Entertainment"
  },
  "Legend of Grimrock II" : {
    "price" : 21.79,
    "genre" : "RPG",
    "editor" : "Almost Human"
  },
  "Shadowrun returns" : {
    "price" : 13.69,
    "genre" : "RPG",
    "editor" : "Harebrained Schemes"
  },
  "Pillars of Eternity: Hero Edition" : {
    "price" : 41.99,
    "genre" : "RPG",
    "editor" : "Obsidian Entertainment"
  },
  "Arx Fatalis" : {
    "price" : 5.49,
    "genre" : "RPG",
    "editor" : "Arkane Studios"
  },
  "Wasteland 2: Director's Cut" : {
    "price" : 39.99,
    "genre" : "RPG",
    "editor" : "inXile Entertainment"
  },
  "Gothic 3: Enhanced Edition" : {
    "price" : 9.09,
    "genre" : "RPG",
    "editor" : "Piranha Bytes"
  },
  "Baldur's Gate II: Enhanced Edition" : {
    "price" : 18.19,
```

```

    "genre" : "RPG",
    "editor" : "Beamdog"
  }
}

```

Pour nonrpg.json :

```

{
  "Rocket League" : {
    "price" : 14.84,
    "genre" : "Action",
    "editor" : "Psyonix"
  },
  "Call of Duty: Black Ops III" : {
    "price" : 34.99,
    "genre" : "FPS",
    "editor" : "Activision"
  },
  "Anno 2205" : {
    "price" : 28.78,
    "genre" : "Strategy",
    "editor" : "Ubisoft"
  },
  "Counter Strike: Global Offensive" : {
    "price" : 9.26,
    "genre" : "FPS",
    "editor" : "Valve"
  },
  "Grand Theft Auto V" : {
    "price" : 28.89,
    "genre" : "Action",
    "editor" : "Rockstar North"
  },
  "FIFA 16" : {
    "price" : 15.18,
    "genre" : "Sport",
    "editor" : "Electronic Arts"
  },
  "Star Wars: Battlefront" : {
    "price" : 39.99,
    "genre" : "Action",
    "editor" : "Origin"
  }
}

```

Pour chacun des jeux vendus, on indique son prix (en Euros), son genre et son éditeur.

Note. Pour éviter les soucis de compatibilité, enregistrez les fichiers .json au format ANSI, pas UTF-8 !

Dans un premier temps, ajoutez du code PHP pour ajouter dans le div « items » un tableau affichant tous les jeux cités dans le fichier rpg.json (suivez le format de la capture d'écran ci-dessus et, entre autres, n'oubliez pas d'afficher le symbole €), sans toutefois vous préoccuper du code derrière les boutons « Commander ».

Pour récupérer la liste des jeux du fichier rpg.json, utilisez les instructions PHP suivantes.

```
$fichier = fopen("rpg.json", "r"); // ouverture en lecture
$contenu = fread($fichier, filesize("rpg.json"));
$items = json_decode($contenu,true);
```

La dernière instruction permet de décoder le contenu du fichier en l'interprétant comme une description au format JSON et en le traduisant en un tableau associatif.

Étape 3

Modifiez votre code PHP pour qu'on puisse choisir la liste des jeux affichée en entrant les URL suivantes.

```
...achats.php?games=rpg
...achats.php?games=nonrpg
```

Faites aussi en sorte que, par défaut, on utilise le fichier rpg.json.

Associez aussi une action aux boutons « Jeux de rôle » et « Autres jeux ».

Étape 4

Occupez-vous désormais des boutons « Commander » et faites en sorte qu'un clic sur l'un d'eux occasionne une redirection (via document.location en Javascript) vers la page

```
...achats.php?games=XXX&order=YYY
```

où XXX est le nom du fichier qui est actuellement ouvert (soit rpg, soit nonrpg) et YYY est le nom du jeu commandé.

Comme le nom du jeu pourrait contenir des espaces ou d'autres caractères problématiques, il faudra tout d'abord l'encoder via la fonction adéquate (en Javascript ou en PHP).


Pour l'instant, on ne mémorisera pas les commandes, mais on affichera un message indiquant que la commande a été effectuée. Ainsi, si la page achats.php est appelée avec une donnée pour la clef « order », il faudra afficher

```
Commande : <nomDuJeu>
```

dans un paragraphe au début du div « main ».

Testez votre code.

Étape 5

Les commandes effectuées seront stockées sous la forme d'un tableau dans les informations de session. Ce tableau, associé à la clef « order », contiendra le nom et le prix de chacun des jeux commandés. 

Plus précisément, le tableau en question sera 

```
$_SESSION['order']
```


et dans chacune des cases garnies de ce tableau, on trouvera un tableau associatif présentant le nom (clef « name ») et le prix (clef « price ») du jeu commandé. Par exemple, le 3^e jeu commandé aura

```
|| $_SESSION['order'][2]['name']
```

comme nom et

```
|| $_SESSION['order'][2]['price']
```

comme prix.

Pour pouvoir manipuler les informations de session, il faut impérativement activer la commande `session_start()`; avant le début du contenu HTML. Concrètement, cela signifie qu'il vous faudra ajouter un script PHP avant toute balise HTML. En fait, ce script PHP doit commencer dès la première ligne de votre fichier (une simple ligne vide en trop pourrait causer des problèmes).

Le tout début de votre fichier PHP devra donc être

```
|| <?php  
|| ... script avec session_start();
```

Effectuez les deux modifications suivantes.

1. Si la page `achats.php` est appelée avec une valeur pour la clef « order », chargez tout d'abord le contenu du fichier json et vérifiez qu'il y a bien un jeu de ce nom (pourquoi une telle vérification est-elle nécessaire ?).

Si ce n'est pas le cas, complétez le message « Commande : <nomJeu> » en « Commande : <nomJeu> incorrect ! » et ne faites rien de plus.

Si c'est le cas, complétez le message en « Commande : <nomJeu> ajouté ! » et ajoutez au tableau `$_SESSION['order']` une nouvelle « case » contenant le nom et le prix du jeu.

2. Garnissez le div « basketContents » avec la liste des jeux commandés. Si celle-ci est vide, on affichera « Votre panier est vide. » Dans le cas contraire, on affichera les noms et prix des jeux ainsi que le total de la commande (voir capture d'écran).

Étape 6

Testez votre code en commandant quelques jeux.

Sous Firefox, cliquez sur le bouton principal (symbolisé par trois barres horizontales) puis choisissez « Options » et ensuite l'onglet « Privacy ». Dans « History », choisissez l'option « Use custom settings » puis accédez à vos cookies.

Dans ceux associés au site « vm-debian.iesn.be », notez que vous avez désormais une valeur pour la clef `PHPSESSID`. Il s'agit de votre identifiant de session PHP. C'est cette valeur, qui est transmise au serveur à chaque contact, qui lui permet de retrouver vos informations de session.

Étape 7

Supprimez vos cookies pour le site vm-debian.iesn.be puis rechargez la page achats.php. Notez que votre panier est vide... c'est normal : comme vous n'avez plus votre ancien identifiant de session, PHP vous en a attribué un nouveau.

Revenez dans les options de Firefox, supprimez les valeurs des cookies pour vm-debian.iesn.be et assurez-vous que l'option « Accept cookies from sites » est décochée. Testez à nouveau votre code.

Le résultat ici dépend des paramètres choisis dans php.ini. Si on l'y autorise, PHP peut détecter que les cookies sont désactivées et passer l'identifiant de session en utilisant la méthode GET.

Étape 8

Pour terminer, faites en sorte que si la page achats.php est appelée avec une valeur pour la clef « clear », le panier soit vidé. Associez l'action adéquate au bouton « Vider votre panier ».

Quant au bouton « Passer commande »... ce sera pour une autre histoire...

Exercice 3 : gestion des membres

Dans cet exercice, vous aurez à gérer une liste de membres qui peuvent se connecter à un site. Les membres enregistrés (ainsi que leur mot de passe et leur statut - administrateur ou pas) seront conservés dans un fichier texte sur le serveur, au format JSON.

Pour tout ce qui concerne la gestion des fichiers par PHP, référez-vous au site php.net, et plus particulièrement à la documentation des fonctions fopen, fread, fwrite et fclose. Pour ce qui concerne l'utilisation du format JSON, consultez la documentation des fonctions json_decode et json_encode.

Étape 1

Dans un premier temps, créez une page HTML appelé ajout.html et comportant le code suivant.

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Ajout de membre</title>
    <style>
      #commentaire {
        color: red;
        font-weight: bold;
      }
      .message {
        color: blue;
        font-weight: bold;
      }
    </style>
```

```

</head>
<body>
  <form action="ajout.php" method="POST">
    <p>
      <label for="pseudo">Pseudo : </label>
      <input id="pseudo" name="pseudo" type="text" />
    </p>
    <p>
      <label for="mdp">Mot de passe : </label>
      <input id="mdp" name="mdp" type="password" />
      <span id="commentaire"></span>
    </p>
    <p>
      <input id="admin" name="admin" type="checkbox" />
      <label for="admin">Admin</label>
    </p>
    <button id="bSubmit" type="submit">Ajouter</button>
  </form>
</body>
</html>

```

Le but final de cette page est de permettre à un administrateur d'encoder de nouveaux membres.

Dans un premier temps, ajoutez des validations HTML et Javascript au formulaire en question.

- Les champs « pseudo » et « mot de passe » ne peuvent pas être vides (validation HTML).
- Le champ « mot de passe » devra contenir au moins 6 caractères et être composé d'au moins un chiffre (validation Javascript).
- Arrangez-vous pour qu'en cas de modification du champ « mot de passe », si le nouveau mot de passe est invalide, on affiche un message d'erreur explicite dans le span identifié par « commentaire » (par exemple : « Minimum 6 caractères ! » ou « Au moins 1 chiffre ! »).
- Assurez-vous que le span « commentaire » soit bien vidé lorsque le mot de passe entré est valide. Organisez votre code intelligemment : créez une fonction qui s'occupe de placer un commentaire à l'écran... et arrangez-vous pour ne pas faire une recherche via `document.getElementById` à chaque fois ?
- Faites également en sorte qu'il soit impossible d'envoyer le formulaire tant que le mot de passe n'est pas valide.

Étape 2

Renommez votre fichier en `ajout.php` et ajoutez-y du code PHP.

Le code en question vérifiera si le serveur a reçu des valeurs pour « pseudo » et pour « mdp » via la méthode POST. Si c'est le cas, on ajoutera au début de la page (juste avant le formulaire) un message indiquant

```

|| Utilisateur <pseudo> (admin) ajouté !

```

s'il s'agit d'un administrateur, et

```
|| Utilisateur <pseudo> ajouté !
```

dans le cas contraire.

Pour le moment, ne vous préoccupez pas de savoir si l'utilisateur a été bien ajouté ou pas... affichez tout simplement le message. Utilisez un paragraphe de classe « message ».

Étape 3

Créez un second fichier, appelé par exemple membres.php, dans lequel vous allez placer tout le code relatif aux opérations de lecture/écriture du fichier où sera conservée la liste des membres. Ajoutez au premier fichier une instruction permettant de s'assurer que PHP ira bien lire le contenu du nouveau fichier.

Dans ce nouveau fichier, commencez par la ligne

```
|| $nomFichier = "membres.txt";
```

qui place dans la variable globale le nom de fichier qui sera utilisé.

Rédigez une fonction ajouteMembre(\$pseudo, \$mdp, \$admin) qui (a) vérifie si le membre décrit par ces arguments est déjà présent dans le fichier et (b) renvoie false sans rien faire d'autre s'il est déjà présent ou (c) renvoie true après avoir ajouté le membre s'il n'était pas présent.

Le fichier contiendra la description JSON d'un tableau associatif associant à tout pseudo de membre un tableau associatif reprenant son mot de passe (associé à la clef « mdp ») et le fait qu'il soit admin ou pas (associé à la clef « admin »). Ainsi, si on appelle \$membres le tableau associatif décrit dans le fichier et que « Toto » est un membre administrateur dont le mot de passe est « 123456 », on aura :

```
|| $membres["Toto"]["mdp"] = "123456" et  
|| $membres["Toto"]["admin"] = true
```

Si la liste comporte deux membres (le Toto ci-dessus et un autre membre, Tata), voici ce que pourrait contenir le fichier membres.txt.

```
|| {"Toto":{"mdp":"123456","admin":true},"Tata":{"mdp":"654321aaa","admin":false}}
```

Pour vous aider, voici une découpe possible pour le code de la fonction ajouteMembre :

1. Ouvrir le fichier en lecture.
2. Si le fichier existe...
 - a. Lire le contenu du fichier dans une variable.
 - b. Transformer le contenu en un tableau associatif \$membres.
 - c. Fermer le fichier.
 - d. Le membre a été trouvé si \$membres[\$pseudo] existe.
3. Sinon...
 - a. \$membres = tableau vide
 - b. Le membre n'a pas été trouvé.

4. Fin Si
5. Si le membre a été trouvé...
 - a. Renvoyer false.
 - b. Fin de la fonction.
6. Sinon...
 - a. Ajouter le nouveau membre dans le tableau \$membres.
 - b. Ouvrir le fichier en ré-écriture (écrasement).
 - c. Écrire dans le fichier la version JSON de \$membres.
 - d. Fermer le fichier.
 - e. Renvoyer true.

Modifiez le fichier ajout.php pour que, si le serveur reçoit des informations en POST, il fasse appel à la fonction ajouteMembres et que, en fonction de la réponse de cette fonction, il affiche soit le message précédent (utilisateur ajouté), soit

```
|| Utilisateur <nom> déjà présent !
```

Testez votre code complètement (éventuellement, passez-le tout d'abord sur le site Sandbox afin d'éliminer les erreurs de syntaxe).

Étape 4

Créez un nouveau fichier appelé liste.php.

Celui-ci devra produire une page HTML affichant la liste des membres actuellement enregistrés dans un format similaire à celui de la capture d'écran ci-contre.

Les lignes bleutées correspondent aux administrateurs.

Quelques consignes :

- Dans le fichier membres.php, créez une nouvelle fonction appelée listeMembres() et renvoyant la liste des membres actuels sous la forme d'un tableau associatif.
- Dans le fichier liste.php, faites appel à cette fonction.
- Utilisez une règle CSS pour les lignes correspondant aux administrateurs.
- C'est une bonne occasion d'utiliser un « foreach ».

Étape 5

Dans liste.php et ajout.php, au début du <body> HTML, ajoutez les lignes suivantes.

```
<header>
  <button id="bAjout" onclick="location =
  'ajout.php';">Ajouter/retirer un membre</button>
  <button id="bListe" onclick="location =
  'liste.php';">Liste des membres</button>
  <button id="bDeconnexion">Déconnexion</button>
</header>
```

Si vous le désirez, ajoutez également les styles suivants.

```
|| header {
```

Liste des membres	
Pseudo	Mot de passe
Toto	123456
Tata	654321aaa
Rututu	999999
Pointu	64646464
Versatile	111222333
Ludo	alalalal

```

background-color: beige;
padding: 12px 0;
width: 600px;
}
header button {
margin: 0 6px;
width: 184px;
}

```

L'effet des deux premiers boutons a déjà été encodé. Testez le tout.

Notez que, dans un site plus large, on aurait pu placer la description du header dans un fichier séparé et l'inclure grâce aux commandes PHP.

Étape 6

Vous avez sans doute remarqué que, dans le header ajouté, le premier bouton s'intitule « Ajouter/retirer un membre ». Hors, pour le moment, la page ajout.php ne permet que la première action...

Éditez le fichier ajout.php pour ajouter un second bouton de soumission pour la suppression. Ajoutez également un attribut « name » et un attribut « value » aux deux boutons afin d'obtenir les lignes suivantes.

```

<button id="bAjout" type="submit" name="action"
value="ajouter">Ajouter</button>

<button id="bSupp" type="submit" name="action"
value="supprimer">Supprimer</button>

```

Ces attributs indiquent au navigateur qu'il doit transmettre une valeur supplémentaire via la méthode POST. Cette valeur sera associée à la clef « action » et vaudra soit « ajouter », soit « supprimer » (en fonction du bouton sur lequel on aura appuyé).

Dans le cas d'une suppression, il suffira d'entrer le pseudo. Il faut donc enlever la validation HTML qui requerrait que le champ « mot de passe » soit complété.

Pour que tout fonctionne, voici les actions qu'il vous reste à accomplir...

1. Ajouter dans membres.php une fonction supprimeMembre(\$pseudo) qui va lire la liste des membres et supprimer le membre dont le pseudo est donné ; la fonction renverra un booléen indiquant si une suppression a eu effectivement lieu (c'est-à-dire si le membre en question a bien été trouvé).
2. Modifier le script PHP du fichier ajout.php pour qu'il vérifie qu'en plus d'une valeur pour « pseudo » et pour « mdp », on lui a bien transmis une valeur pour « action » via POST. Si cette valeur est « ajouter », le code précédent s'appliquera. Si la valeur est « supprimer », il faudra faire appel à la nouvelle fonction supprimeMembre(\$pseudo) et, en fonction de son résultat, afficher « Utilisateur <pseudo> supprimé ! » ou « Utilisateur <pseudo> pas trouvé ! ».
3. Finalement, vu qu'il s'agit ici d'effectuer une suppression définitive, il est préférable de demander une confirmation à l'utilisateur. Cela peut se faire en ajoutant à

l'événement « onclick » du bouton « Supprimer un membre » une fonction qui renverra true en cas de confirmation et false sinon. Le code de cette fonction peut être relativement simple (pensez à la fonction « confirm » de Javascript).

Étape 7

Soyons réaliste... C'est une très mauvaise idée de stocker les mots de passe « en clair » dans un fichier, parce que l'obtention du fichier donnerait accès non seulement à tous les comptes utilisateurs du site en question mais sans doute sur d'autres sites également (la plupart des personnes utilisant des mots de passe identiques ou au moins similaires sur divers sites).

Utilisation d'une fonction de hashage. Plutôt que de stocker un mot de passe « P4SSWORD », on garde en mémoire le résultat donné par une fonction de hashage pour ce mot de passe (comme par exemple les fonctions SHA1 ou MD5). Ici, il pourrait par exemple s'agir de la chaîne « 752c14ea195c460bac3c3b7896975ee9fd15eeb7 » qu'on stockera dans un fichier (ou une base de données).

Lors d'une tentative de connexion, pour vérifier qu'un utilisateur entre « le bon mot de passe », on transforme le mot de passe proposé en utilisant la même fonction et on compare les résultats.

Un peu de sel... Si la plupart des bases de données utilisaient le même algorithme de hashage, obtenir des informations sur les membres d'un site permettrait de se connecter quasiment partout ailleurs. En effet, si on sait que le résultat du hashage est la chaîne « 752c14ea195c460bac3c3b7896975ee9fd15eeb7 », il suffit de trouver un mot de passe qui, hashé, donne ce même résultat... et on peut se connecter partout.

Pour accroître la sécurité, on « sale » (du verbe saler, ajouter quelques grains de sel) les mots de passe avant de les soumettre à la fonction de hashage. Saler un mot de passe revient généralement à lui ajouter un préfixe et un suffixe, chaque site utilisant un préfixe et un suffixe propre.

Ainsi, si l'utilisateur entre le mot de passe « P4SSWORD », pour trouver l'information à stocker dans la base de données, on va tout d'abord saler le mot passe. Le résultat pourrait être quelque chose comme « 4@\$P4SSWORD!Z# » si le site choisit de saler via le préfixe « 4@\$ » et le suffixe « !Z# ». C'est ce mot de passe salé qui sera passé à la fonction de hashage et le résultat de cette opération qui sera stocké.

Naturellement, quand l'utilisateur tentera de se connecter, il faudra saler puis hasher sa proposition de mot de passe avant de comparer le résultat avec l'information stockée.

En PHP 5.5 et plus, il existe une bibliothèque utilisant des fonctions telles que password_hash et password_verify qui s'occupent du salage, de l'encodage et de la vérification des mots de passe. Dans le cadre de ce laboratoire, on va plutôt utiliser la méthode « à l'ancienne » en accomplissant les étapes une par une.

Avant de continuer, choisissez un préfixe et un suffixe pour le salage. Vous pouvez les ajouter comme variables globales dans le fichier membres.php.

Dans la fonction ajouteMembre, assurez-vous de saler (avec le préfixe et le suffixe choisis) puis de hasher (via la fonction md5) le mot de passe de l'utilisateur avant de l'encoder dans le fichier.

Une fois le tout réalisé, créez un nouveau membre administrateur (vous en aurez besoin plus tard). Observez la valeur codée dans la liste des membres.

Étape 8

Créez un nouveau fichier login.html. Dans celui-ci, placez un formulaire demandant un pseudo et un mot de passe (les deux étant requis avant de pouvoir soumettre le formulaire) ainsi qu'un bouton « Connexion ».

```
<form action="login.php" method="POST">
  <p>
    <label for="pseudo">Pseudo : </label>
    <input id="pseudo" name="pseudo" type="text" required="required" />
  </p>
  <p>
    <label for="mdp">Mot de passe : </label>
    <input id="mdp" name="mdp" type="password" required="required" />
  </p>
  <button id="bLogin" type="submit">Connexion</button>
</form>
```

Il s'agira de la page d'entrée du site, permettant aux membres de se connecter. Voici les restrictions qu'on désire implémenter : seuls les membres ont accès à la page « Liste des membres », seuls les membres administrateurs ont accès à la page « Ajouter/retirer un membre ».

Dans un premier temps, revenez au fichier membres.php pour y ajouter une fonction verifieMembre(\$pseudo,\$mdp) testant les informations de connexion rentrées par un utilisateur. Cette fonction doit pouvoir retourner plusieurs réponses... pour rendre le code plus clair, utilisez les quatre constantes définies ci-dessous (n'oubliez pas de recopier ces 4 lignes au début de membres.php). Pour rappel : les constantes sont automatiquement globales, inutile de les déclarer en global dans les fonctions où vous désirez les utiliser !

```
define("LOGIN_OK", 1);
define("LOGIN_ADMIN", 2);
define("USER_NOT_FOUND", 3);
define("INCORRECT_PASSWORD", 4);
```

De retour à login.php, ajoutez un script en début de fichier vérifiant si le serveur reçoit des valeurs pour « pseudo » et pour « mdp » en méthode POST. Si c'est le cas, le script vérifiera que ces informations sont correctes en utilisant la fonction verifieMembre.

En cas de problème, on affichera un message adéquat (par exemple « Utilisateur inconnu ! » ou « Mot de passe incorrect ! ») avant le formulaire.

En cas de réussite, il faudra stocker dans les informations de session (a) le login de l'utilisateur et (b) un booléen indiquant s'il s'agit d'un administrateur ou pas. Dans tous les cas, cela nécessite de créer une session. Voici quelques instructions qui vous seront nécessaires.


```
// Pour créer une session ou ouvrir la session en cours
session_start();

// Pour stocker des informations dans la session
$_SESSION["pseudo"] = $pseudo;

// Pour rediriger vers la page liste.php en cas de connexion réussie
header('location: liste.php');
```

Note importante. La première et la troisième des instructions présentées ci-dessus se traduisent par des informations qui doivent être placées dans l'en-tête du message http envoyé par le serveur. Elles doivent donc se trouver dans un script PHP positionné avant toute balise HTML, en tout début de fichier. (Même une ligne blanche avant la balise ouvrante `<?php` pourrait causer des problèmes !)

Il reste à sécuriser les pages `liste.php` et `ajout.php`.

Pour la première, voici le code que vous pourriez placer en tout début de script (ici encore, il doit s'agir d'un script PHP situé avant toute balise HTML).

```
session_start();
if (!isset($_SESSION['pseudo'])) {
    header('location: login.php');
}
```

En clair : si aucune information concernant un « pseudo » ne se trouve dans les données de la session, alors l'utilisateur est redirigé vers la page `login.php`.

Quelques suggestions pour terminer cet exercice :

- Ajouter « Bienvenue, <pseudo> ! » dans le header des pages réservées aux membres.
- Dans la page `login.php`, faire en sorte que, si l'utilisateur entre un pseudo qui existe avec un mot de passe incorrect, lorsque la page est affichée à nouveau, le pseudo entré est conservé (il ne doit pas le retaper).
- Protéger la page `ajout.php` de sorte que seuls les administrateurs y aient accès (les personnes non connectées sont redirigées vers `login.php` ; les membres non administrateurs sont redirigés vers `liste.php`).
- Faire en sorte que le bouton « Ajouter/retirer un membre » ne soit affiché que pour les administrateurs.
- Faire en sorte qu'en cas de clic sur le bouton « déconnexion », on soit envoyé vers la page `login.php` en transmettant l'information « `clear=yes` » au format GET ; et que, dans ce cas-là, le script de la page `login.php` supprime la session en cours.

Troisième partie : Ajax et XHR

Pour tester rapidement du code PHP : <http://sandbox.onlinephpfunctions.com/>

Exercice 1 : Définitions

Dans le cadre de cet exercice, vous allez construire une page HTML contenant un court texte relatif décrivant les principes d'Ajax. Parmi ce texte se trouveront un certain nombre de mots dont l'utilisateur pourra obtenir la définition par un simple clic. En cas de clic, le navigateur demandera la définition correspondante au serveur et l'affichera sur la page actuelle (sans charger une nouvelle page).

Étape 1

Dans un premier temps, créez une page HTML avec le code suivant.

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8" />
    <style>
      mark {
        border-bottom: 1px dashed blue;
        background-color: transparent;
      }
      footer {
        min-height: 20px;
        background-color: beige;
      }
    </style>
    <script>
      var pDef;
      window.onload = init;
      function init() {
        pDef = document.getElementById("pDefinition");
      }
    </script>
  </head>
  <body>
    <h1>Cliquez sur les mots soulignés en pointillés pour obtenir une
    définition.</h1>
    <p>Dans une application Web, la méthode <mark>classique</mark> de
    dialogue entre un <mark>navigateur</mark> et un <mark>serveur</mark>
    est la suivante : lors de chaque manipulation faite par l'utilisateur,
    le <mark>navigateur</mark> envoie une <mark>requête</mark> contenant
    une référence à une page Web, puis le <mark>serveur</mark> Web
    effectue des calculs, et envoie le résultat sous forme d'une page Web
    à destination du <mark>navigateur</mark>. Celui-ci affichera alors la
    page qu'il vient de recevoir. Chaque manipulation entraîne la
    <mark>transmission</mark> et l'affichage d'une nouvelle page.
```

```

L'utilisateur doit attendre l'arrivée de la réponse pour effectuer
d'autres manipulations.</p>
    <p>En utilisant <mark>Ajax</mark>, le dialogue entre le
<mark>navigateur</mark> et le <mark>serveur</mark> se déroule la
plupart du temps de la manière suivante : un programme écrit en
langage de programmation <mark>JavaScript</mark>, incorporé dans une
page web, est exécuté par le <mark>navigateur</mark>. Celui-ci envoie
en arrière-plan des <mark>demandes</mark> au serveur Web, puis modifie
le contenu de la page actuellement affichée par le
<mark>navigateur</mark> Web en fonction du résultat reçu du
<mark>serveur</mark>, évitant ainsi la transmission et l'affichage
d'une nouvelle page complète.</p>
    <footer>
        <p id="pDefinition"></p>
    </footer>
</body>
</html>

```

Observez le contenu de la page, tant au niveau du code HTML qu'au niveau du texte. Notez la présence d'un paragraphe identifié par « pDefinition » et l'utilisation de l'événement window.onload pour définir un objet global Javascript (pDef) permettant de cibler ce paragraphe. Il s'agit de l'endroit où on écrira la définition.

Étape 2

Du côté PHP, sur le serveur, construisez un fichier appelé definitions.php et contenant les quelques définitions suivantes sous la forme d'un tableau associatif \$defs.

```

<?php

$defs = array(
    "classique" => "habituelle",
    "serveur" => "machine où le site web est hébergé",
    "navigateur" => "programme qui permet de naviguer sur le web",
    "Ajax" => "Asynchronous Javascript and XML",
);

```

Complétez ce fichier pour qu'en cas d'appel tel quel

```

...(adresse serveur).../definitions.php?mot=navigateur

```

Il « renvoie » la définition correspondant au mot « navigateur ».

« Renvoyer » signifie ici simplement écrire la définition (via echo par exemple). Le document que le code PHP doit produire n'est pas une page HTML mais un simple fichier texte (contenant, dans ce cas-ci, la définition demandée).

Si le mot demandé n'est pas trouvé dans le dictionnaire \$defs, on envoie le message « Aucune définition trouvée ».

Si aucun mot n'est donné, on envoie le message « Appel sans mot à définir. »

Étape 3

Vérifiez votre code en entrant directement dans votre navigateur les adresses suivantes.

```
...(adresse serveur).../definitions.php?mot=navigateur  
...(adresse serveur).../definitions.php?mot=motpasdefini  
...(adresse serveur).../definitions.php
```

Le navigateur devrait afficher « en brut » la réponse textuelle créée par le script PHP.

Étape 4

Revenez à votre document HTML.

Histoire de mettre en évidence les mots qui peuvent être cliqués, arrangez-vous pour que, lors du survol d'un mot marqué par la souris, celui-ci soit écrit sur fond gris.

Ajoutez ensuite du code Javascript ciblant chacun des mots marqués et leur associant un événement en cas de clic. Dans un premier temps, et pour tester votre code, faites en sorte que l'événement en question affiche simplement « Ok » dans le paragraphe pDef.

Dans un second temps, pour revoir la programmation Javascript, faites en sorte que l'événement en question affiche le message « Vous demandez la définition du mot XXX. » dans le paragraphe pDef.

Étape 5

Consultez maintenant les slides correspondant à Ajax et faites en sorte qu'en cas de clic, le navigateur envoie une demande de définition au script PHP puis, qu'au moment où la réponse est reçue, celle-ci soit affichée dans le paragraphe pDef.

Le fichier definitions.php doit absolument se trouver sur le serveur (vu qu'il faut un interpréteur PHP pour l'exécuter). Par contre, la page HTML pourrait se trouver a priori soit en local (sur votre disque dur / clef USB) soit sur le serveur. Réalisez des tests dans les deux cas.

Étape 6

Au lieu de renvoyer un simple texte, on peut envoyer des informations structurées au format JSON.

Du côté PHP tout d'abord, modifiez la définition du tableau associatif \$defs pour que, dorénavant, il associe à chaque mot défini un tableau associatif contenant une valeur pour la clef « def » (qui sera la définition), une valeur pour la clef « auteur » (qui contiendra l'auteur de la définition – choisissez des noms au hasard) et une valeur pour la clef « source » (qui indique d'où vient la définition : dictionnaire, Wikipedia, ...).

Si le mot demandé est trouvé, renvoyez l'écriture en JSON (via la fonction json_encode) du tableau associatif qui lui correspond. Testez l'effet via la page HTML (en vous restreignant dans un premier temps aux mots qui possèdent bien une définition).

Il faut également tenir compte du cas où le mot demandé n'existe pas... une solution « propre » serait d'ajouter aux informations envoyées un champ (une clef dans le tableau

associatif) indiquant si le mot a été trouvé ou pas. Ainsi, si ce champ s'appelle « trouve », du côté javascript, il suffira de tester si celui-ci est vrai ou faux pour savoir s'il faut afficher la définition ou « Aucune définition trouvée ». Concrètement, cela signifie

- ajouter l'association trouve/vrai au tableau associatif extrait de \$defs dans le cas où le mot demandé a été trouvé ;
- renvoyer la version JSON du tableau associatif contenant uniquement trouve/faux dans le cas contraire.

Une fois encore, testez le message renvoyé par le script PHP en utilisant votre page HTML (qui, pour le moment, l'affiche « en brut », au format JSON).

Étape 7

Revenez du côté Javascript et modifiez l'affichage du message renvoyé par le script PHP.

En cas de mot non défini, il faudrait afficher simplement « Aucune définition trouvée ».

En cas de mot défini, il faudrait afficher la définition au format suivant :

```
MOTDÉFINI
... définition ... (par auteur)
Source : source
```

Arrangez-vous pour construire éventuellement des styles adéquats et pour modifier la structure HTML du footer.

Exercice 2 : Les soldes

Contrairement aux exercices précédents qui étaient détaillés étape par étape, pour ce dernier énoncé récapitulatif, on vous présente uniquement l'objectif final. À vous de déterminer comment procéder et quels éléments mettre en place pour y arriver.

Dans le cadre de cet exercice, on suppose qu'un site de vente en ligne dispose d'un certain stock à écouler en solde. L'état actuel du stock est conservé sur le serveur, au format JSON. Le fichier contient la description d'un tableau dont chaque cellule contient un tableau associatif relatif à un article et indiquant (a) le nom de cet article et (b) le nombre d'articles restant en stock.

(Pour simplifier le problème, on ne considère pas dans cet énoncé les prix des articles, ni l'établissement des factures.)

Un utilisateur devra pouvoir charger une page « soldes.php » qui lui indiquera la liste des articles en stock. Pour chaque article en stock, on affichera son nom, le nombre d'articles restant en stock et un bouton permettant d'acheter un de ces articles, avec les particularités suivantes.

- Le nombre d'articles encore disponible sera mis à jour toutes les secondes (parce que, entre temps, d'autres clients auraient pu en acheter un).
- En cas de clic sur le bouton d'achat, il faudra vérifier (sur le serveur) qu'il reste bien un article disponible et afficher sur la page web « Achat réussi » ou « Stock épuisé ».

- Le client ne chargera aucune autre page que « soldes.php ». L'évolution des stocks et les affichages liés aux commandes seront effectuées dynamiquement en Javascript sans charger une nouvelle page ni recharger la page.

Pour tester votre code, n'hésitez pas à vous mettre à plusieurs sur le même « site de vente », afin de vérifier que les commandes d'une personne fait bien baisser le stock restant affiché sur les pages des autres personnes.

Aussi, attention aux modifications concurrentes du fichier contenant l'état du stock actuel : pour bien faire, il faut éviter que deux processus différents (par exemple deux achats) aient accès au fichier en même temps car cela pourrait fausser les données. Par exemple, s'il reste 5 savons et que deux acheteurs en commandent un en même temps, un problème pourrait survenir si les actions se produisaient dans l'ordre suivant :

1. Processus d'achat 1 lit le fichier : stock = 5
2. Processus d'achat 2 lit le fichier : stock = 5
3. Processus d'achat 1 décrémente sa variable stock
4. Processus d'achat 2 décrémente sa variable stock
5. Processus d'achat 1 écrit le nouveau stock : 4
6. Processus d'achat 2 écrit le nouveau stock : 4
7. (au final, le stock est de 4 alors qu'il aurait dû être de 3).

Pour éviter ces problèmes, jetez un coup d'œil du côté de la fonction php « flock » (pour f-lock, file lock).