Module 307

Réaliser des pages Web interactives

Rapport personnel

Malori Burgy

Module du 15.05.2023 au 13.06.2023

Version 1

Du 15.05.2023

Table des matières

[1 Introduction 4](#_Toc136865159)

[2 Exercices 4](#_Toc136865160)

[2.1 Exercice 1 4](#_Toc136865161)

[2.2 Exercice 2 5](#_Toc136865162)

[2.3 Exercice 3 7](#_Toc136865163)

[2.4 Exercice 4 9](#_Toc136865164)

[2.5 Exercice 5 10](#_Toc136865165)

[2.6 Exercice 6 13](#_Toc136865166)

[2.7 Exercice 7 15](#_Toc136865167)

[2.8 Exercice 8 18](#_Toc136865168)

[2.9 Exercice 9 20](#_Toc136865169)

[2.10 Exercice 10 22](#_Toc136865170)

[2.11 Exercice 11 27](#_Toc136865171)

[2.12 Exercice 12 30](#_Toc136865172)

[2.13 Exercice 13 31](#_Toc136865173)

[2.14 Exercice 14 33](#_Toc136865174)

[2.15 Exercice 15 35](#_Toc136865175)

[2.16 Exercice 16 37](#_Toc136865176)

[2.17 Exercice 17 38](#_Toc136865177)

[2.18 Exercice 18 39](#_Toc136865178)

[2.19 Exercice 20 42](#_Toc136865179)

[3 Projet 44](#_Toc136865180)

[3.1 Introduction du projet 44](#_Toc136865181)

[3.1.1 Définition: but, objectifs et fonctionnement désiré de votre projet​ 44](#_Toc136865182)

[3.1.2 Explications API 44](#_Toc136865183)

[3.1.3 Site de référence 44](#_Toc136865184)

[3.1.4 Fonctionnement: requête, réponse, format​ 44](#_Toc136865185)

[3.1.5 Exemple d’utilisation 44](#_Toc136865186)

[3.2 Analyse 44](#_Toc136865187)

[3.2.1 Diagramme de use cases / Explications des cas 44](#_Toc136865188)

[3.2.2 Maquette 44](#_Toc136865189)

[3.3 Conception 44](#_Toc136865190)

[3.3.1 Diagramme de navigation 44](#_Toc136865191)

[3.4 Implémentation 44](#_Toc136865192)

[3.4.1 HTML / CSS (buts des fichiers, éventuellement des extraits de code avec explications) 44](#_Toc136865193)

[3.4.2 Javascript général (buts des fichiers, extraits de code avec explications) 44](#_Toc136865194)

[3.4.3 Javascript spécifiques (buts des fichiers, extraits de code avec explications) 44](#_Toc136865195)

[3.4.4 Descente de code complète pour une action qui fait appel à un service Web 44](#_Toc136865196)

[4 Conclusion 44](#_Toc136865197)

# Introduction

# Exercices

## Exercice 1

Dans cet exercice, nous avons un affichage qui contient du texte et un bouton avec notre prénom. Quand on clique sur ce bouton, il va dire que c’est nous qu’il l’avons pressé.

Voici un aperçu de la maquette avant que je change les informations avec mon nom et prénom :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

* Comment ajouter un écouteur "click" via Javascript.

Dans le modèle, l’écouteur du bouton est ajouté depuis le fichier javascript. Pour rajouter un événement, on utilise simplement la fonction « addEventListener » avec l’ID qui fait référence à notre bouton, ici il s’appelle « testez ». Voici à quoi ressemble le code :

// Ecouteur du bouton "Testez-moi..."

document.getElementById("testez").addEventListener("click", testez);

On peut voir aussi dans l’html que l’id de mon bouton est bien « testez ».

<body onload="initCtrl()">

    <div id="container">

      <p>Cliquez sur le bouton qui contient votre prénom (js) !</p>

      <button id="testez">Teste-moi, James</button>

      <p id="info">&nbsp;</p>

    </div>

  </body>

Dans l’exercice, il est demandé de changer cette manière de faire et de directement ajouter l’écouteur sur le bouton dans le fichier html. Il est préférable d’utiliser l’autre méthode à l’avenir mais pour l’exercice, c’est bien de pouvoir expérimenter les deux.

J’ai donc directement rajouté l’attribut « onclick » à mon bouton et je lui ai donner ma fonction javascript « testez () ».

<body onload="initCtrl()">

  <div id="container">

    <p>Cliquez sur le bouton qui contient votre prénom (html) !</p>

    <button onclick="testez()">Teste-moi, Malori</button>

    <p id="info">&nbsp;</p>

  </div>

</body >

Voici ce que cela donne une fois que j’ai tout changé :

Une image contenant texte, capture d’écran, Police, affichage

Description générée automatiquement

Il est aussi possible d’exécuter le code JS une fois que tout a été chargé en changeant simplement e place le « onload » on peu le mettre sur la balise de fermeture du Body ce n’est pas un problème.

En web, on va parler de « DOM » Document Object model, c’est un peut l’arborescence de la page et de ce qui la compose.



Le script javascript, ne peut être lancé uniquement une fois que toute cette arborescence a été créé. On peut la lancer en début ou en fin de chargement comme indiqué au-dessus et on peut aussi déplacer el lancement du script au besoin.

## Exercice 2

Dans cet exercice, on nous demande de simuler une page de login qui va afficher un formulaire de connexion. Pour le moment le test de login va se faire localement avec javascript. Les informations de connexions étant les suivantes :« admin » quelle que soit la casse pour l’utilisateur et « emf123 » pour le mot de passe).

Voici la maquette :

Une image contenant texte, logiciel, capture d’écran

Description générée automatiquement

Si on introduit les bonnes informations de connexion :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Et si on n’introduit pas les bonnes informations :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

On va commencer par analyser le code HTML de la page :

<body onload="initCtrl()">

   <form class="user-form">

      <fieldset>

         <legend>Identification:</legend>

         <div class="field">

            <label for="username">Nom d'utilisateur:</label>

            <input type="text" size="30" id="username" placeholder="un nom svp" autofocus />

         </div>

         <div class="field">

            <label for="username">Mot de passe:</label>

            <input type="password" size="30" id="password" placeholder="un mot de passe svp" />

         </div>

         <input type="button" value="Valider" id="valider" onclick="validerUtilisateur();">

      </fieldset>

   </form>

</body>

L’attribut **« placeHolder »** va nous permettre de mettre du texte dans le champ que l’utilisateur devra remplir, cela lui permet aussi de savoir ce qu’il faudra qu’il y rentre.

L’attribut **« autofocus**» est Boolean qui indique que le « focus » de la page doit se faire sur cet élément.

On peut voir qu’ici on utilise un **« input »** et pas un **« button »,** la seule différence qu’il y a entre les deux, c’est que le « button » est un peu plus personnalisable et polyvalent que « input » On peut lui spécifier une valeur qui sera son texte mais c’est tout.

On peut aussi voir déjà la fonction js « validerUtilisateur » qui est appelé lorsque l’on clique sur le bouton valider.

On peut après se pencher sur le fichier javascript. J’ai réalisé une fonction « validerUtilisateur » qu’on définit avec le préfixe **« function »** qui récupère les informations de connexion entrées et teste si elles sont valides. Elle va tester si le nom d’utilisateur en minuscule est valable. Pour cela, on utilise la méthoed **« toLowerCase »** comme en java.

function validerUtilisateur() {

 // récupère les objets InputText du formulaire HTML et leur valeur

   let username = document.getElementById("username").value;

   let password = document.getElementById("password").value;

    // validation du user et password

  if (username.toLowerCase() === "admin" && password === "emf123") {

    window.alert("Validation OK.");

  } else {

    window.alert("Utilisateur ou mot de passe incorrect !!!");

  }

}

J’ai fait deux variables grâce au mot clé **« let »** suivit d’un nom qui vont rechercher grâce à leur id les éléments « username » et « password ». Ils sont ensuite testés dans une condition en dessous.

En javascript, il n’est pas nécessaire de déclarer le type des variables.

**La fonction « alert() »** sert à afficher un pop-up pour dire si la validation est ok ou non.

Si on doit débugger et afficher par exemple les valeurs des deux champs, on peut utiliser la ligne suivante dans notre fonction JavaScript :

console.log("utilisateur: " + username + " mot de passe: " + password);

On peut afficher cette console en appuyant sur **F12**.

## Exercice 3

Pour cet exercice, on va partir sur la base de ce que l’on a fait à l’exercice précédent. Il va falloir changer un peu l’affichage de noter page pour qu’elle ressemble à ça :

Une image contenant texte, logiciel, capture d’écran

Description générée automatiquement

La première question à répondre est Comment allons-nous centrer et styliser ce bouton de validation ?

On va commencer par encapsuler le bouton dans une balise **« div »** à qui on va rajouter une classe que je nomme **« button ».**

 </div>

         <div class="button">

         <input type="button" value="Valider" id="valider" onclick="validerUtilisateur();">

      </div>

Il va suffire ensuite d’ajouter la classe **« button »** dans mon fichier css et dans le bouton lui-même dans le fichier html changer son type en **« submit »** Nous n’aurons donc plus besoin du « onclick » présent.

Le nouveau code est donc :

<div class="button">

         <input type="submit" value="Valider" id="valider" >

      </div>

Voici à quoi ressemble mon code css :

.user-form input[type=submit] {

  width: auto;

  background-color: grey;

  border: none;

  padding: 5px 20px;

  font-size: 16px;

  color: white;

  margin-top: 0.5cm;

  font-weight: 400;

}

.user-form .button {

  display: flex;

  justify-content: center;

  align-items: center;

Pour la deuxième partie, la question est la suivante : Comment envoyer el formulaire à un script PHP qui fera la validation ?

Pour rappel, il existe une multitude de types de boutons en html. Voici une image récapitulative de ces différents types :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Dans notre cas, on a changé notre bouton en type **« submit »** parce qu’il envoie un formulaire vers un serveur quand on le clique et redirigera l’utilisateur vers une page qui a été défini dans l’attribut **« action »,** le **« button »** est simplement un bouton cliquable sans grandes fonctionnalités particulières.

On a donc effectivement besoin d’un **« submit »** et comme indiqué dans l’explication ci-dessus, on va devoir spécifier la page dans l’attribut **« action »** de la balise **« form »**

Ma balise **« Form »** ressemble donc à ça :

<form class="user-form" action="http://burgym.emf-informatique.ch/307/Exercices/exercice\_3/login.php" method="POST">

L’attribut **“action”** envoie le contenu du formulaire vers une page PHP.

Et l’attribut **« method »** qui est sois **GET** sois **POST**, ici **POST** à l’occurrence indique que l’on fait une modification /un ajout. Un **GET** aurait simplement permit e ressortir une information et non de la modifier.

J’ai ensuite rajouté le script PHP fournit dans l’exercice, maintenant le fichier javascript ne va plus nous servir la validation étant faite avec le script PHP. Pour PHP, il faut des attributs **« name »** qui fonctionnent exactement de la même manière que les id que nous avons déjà.

Voici le code PHP qui était fourni, on peut voir qu’il commence et se termine par une balise « **< ?PHP>** ». Pour convertir mon nom d’utilisateur en minuscule, ici la fonction utilisée est **« strtolower »,** j’affiche les informations grâce à un **« echo »**

<?PHP

//Burgy Malori

// le 15.05.2023

//script PHP de l'exercice 3

// test si on a reçu une donnée de formulaire nommée "username"

if (isset($\_POST['username'])) {

  // récupération des données transmises dans des variables locales

    $username = strtolower($\_POST['username']);

    $password = $\_POST['password'];

  // affichage des infos reçues

  echo "username: ".$username."</br>";

  echo "password: ".$password."</br>";

  // test username et mot de passe

  if (($username == "admin") && ($password == "emf123")) {

    echo "<script>alert('Validation OK');</script>";

  } else {

    echo "<script>alert('Utilisateur ou mot de passe incorrect !!!');</script>";

}

}

?>

## Exercice 4

Cet exercice nous sert d’introduction à un outil qui sert à tester du JavaScript à partir de code HTML et CSS. On appelle cela un **« fiddle »** qui se traduirait comme une astuce.

J’ai donc créé un compte sur le site : <http://jsfiddle.net/>

Pour pouvoir tester ces **« fiddle ».** Grâce à ce site, il est possible de tester du code sans écrire toutes les références aux fichiers, il nous suffit juste de taper le code dont nous avons besoin.

Voici un exemple de petite application, j’ai un carré bleu sur une page et quand je clique dessus, un pop-up m’indique que ce carré a été cliqué.

Voici à quoi ressemble la page de l’outil, pour montrer que je n’ai vraiment pas écrit grand-chose et que tout fonctionne :

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Description générée automatiquement

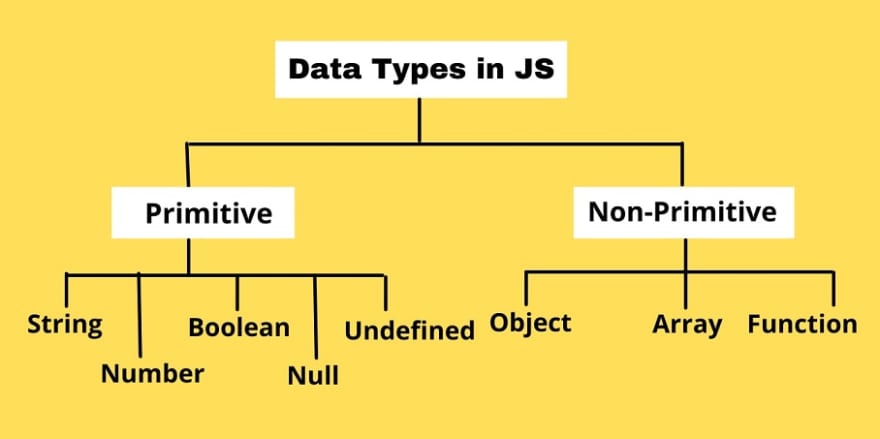
Cela illustre bien l’utilité d’un tel outil. On peut facilement et rapidement tester des bouts de code selon nos besoins.

## Exercice 5

Dans cet exercice, nous allons nous pencher sur les variables en JavaScript. A noter qu’il y a différentes manières de déclarer une variable :

1. myVar;
2. **va**r myVar;
3. **let** myVar;
4. **const** myVar=1;

Une variable peut aussi avoir un type même s’il n’est pas obligatoire de le préciser, voici une image qui récapitules les types que l’on peut rencontrer en javaScript :

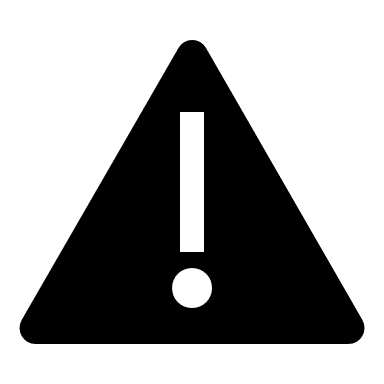


Les deux premières solutions sont équivalentes mais on ne les utilise pas vraiment car elles laissent beaucoup trop de liberté, on peut par exemple déclarer deux fois la même variable avec le même nom ce qui est plutôt problématique…

On devrait plutôt prioriser le mot-clé **« let ».**

Le mot-clé **« const »** est surtout utiliser pour déclarer des constantes et les objets car on peut toujours changer ses propriétés**.**

La grande différence qu’il y a entre les deux mots est qu’une variable déclaré avec un **« let »** peut être réaffecté, on peut changer sa valeur, cela n’est pas possible avec une variable déclaré avec **« const ».**

***Comme en Java, la convention de mettre les constantes en majuscules est utilisée.***

Pour tester tout ça, l’exercice proposait une série d’opérations à effectuer dans une console du navigateur ou dans un jsfiddle, j’ai choisi la deuxième option.

* Effacer le contenu de la console avec « console.clear() »

Console.clear

* Créer une variable nommée « a » ;

let a;

* Afficher le contenu de « a » avec console.log;

console.log(a);

* Stocker la valeur 15 dans cette variable ;

a = 15;

* Afficher le contenu de cette variable dans la console sous la forme « Ma variable a = ? »

console.log("Ma variable a = " + a);

* Créer une variable nommée « b » et lui assigner directement la valeur 9 ;

let b = 9;

* Afficher le contenu de cette variable dans la console sous la forme « Ma variable b = ? »

console.log("Ma variable b = " + b);

* Faire l’addition de ces 2 variables en affichant directement le résultat dans la console sous cette forme : « 15 + 9 = ? » ; (essayer d’utiliser un littéral avec `…${…}…`)

console.log(" 15+ 9 = "+ (a+b));

$c = a+b;

console.log("Avec un litteral: 15+ 9 = "+ $c);

* Compléter en faisant de même pour une soustraction, une multiplication et une division des deux variables ;

console.log(" 15/9 = "+ (a/b));

console.log(" 15\*9 = "+ (a\*b));

console.log(" 15- 9 = "+ (a-b));

* Stocker « Bonjour » dans la variable a ;

a="bonjour";

* Stocker « les amis » dans la variable b ;

b= "les amis"

* Afficher « bonjour les amis » dans la console en concaténant les variables ;

console.log(a+" "+ b)

* Faites la même chose en utilisant un littéral avec `…${…}…`;

$c=a+b;

console.log($c);

* Stocker « true » dans la variable a ;

a=true ;

* Stocker « false » dans la variable b ;

b=false ;

* Effectuer une opération AND entre les 2 variables et afficher le résultat sous cette forme « true AND false = ? » ;

console.log("true AND False = " + a&b);

* Effectuer une opération OR entre les 2 variables et afficher le résultat sous cette forme « true OR false = ? » ;

console.log("true OR False = " + a|b);

* Stocker la date du jour dans la variable a avec new Date();

a = new Date(Date.now());

* Calculer une nouvelle date dans la variable b qui est 61 jours avant la date courante (utilisation getDate, setDate)

let e = (24\*60\*60\*1000) \* 61;

b= new Date(a.setTime(a.getTime()-e));

* Afficher les dates contenues dans les variables a et b en vous aidant de « toLocaleString(), toLocaleDateString() et toLocaleTimeString()» . Afficher la date et l'heure, la date uniquement et l'heure uniquement.

console.log(b.toLocaleString());

console.log(new Intl.DateTimeFormat('FR-CH').format(b));

console.log(b.toLocaleTimeString());

* Le mot réservé « typeof » permet de connaitre le type utilisé momentanément pour une variable. Stocker la valeur de Math.PI dans a, « bonjour » dans b, créer et assigner true dans c, créer, assigner la date courante dans d et déclarez la variable e sans rien lui affectez, puis afficher le type pour les 5 variables :

a= Math.PI;

b= "Bonjour";

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquementlet c = true;

let d = new Date(Date.now());

let e;

console.log(typeof(a));

console.log(typeof(b));

console.log(typeof(c));

console.log(typeof(d));

console.log(typeof(e));

Voici le résultat que ces dernières lignes m’ont donné :

## Exercice 6

Ici, nous allons reprendre l’exercice un et cette fois allons le réaliser avec un switch. On va changer quelques données pour que l’application affiche le jour de la semaine directement en remplacement du contenu qu’une balise html.

Voici la maquette au démarrage :

Une image contenant texte, logiciel, capture d’écran

Description générée automatiquement

Après le clic, on affiche le jour de la semaine :

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

On reprend le code de l’exercice 1, on change l’HTML selon la consigne, le texte change et la fonction appelée va changer. Voici a quoi ressemble le nouveau **« body »** de l’exercice basé sur ce qui est dit dans la consigne et le code existant :

<body>

<div id="container">

<p>

Cliquez sur le bouton ci-dessous pour découvrir quel jour nous sommes.

</p>

<button onclick="afficherJourSemaine();">Testez-moi</button>

<p id="info">&nbsp;</p>

</div>

</body>

Dans le fichier javaScript, on va donc créer notre fonction **« afficherJourSemaine() ».** Pour ce faire, il va falloir utiliser un switch avec le numéro du jour et on va utiliser une méthode de la classe **« Date »** pour retourner le numéro du jour de la semaine.

Voici à quoi ressemble ma fonction :

function afficherJourSemaine() {

  let info = document.getElementById("info");

  //ATTENTION avec la méthode getDay(), le 0 corresponds à DIMANCHE et non lundi

  let jSem = new Date().getDay();

  // je fait une variable avec le début de ma phrase

  //et je lui rajouterai le jour correspondant dans mon switch selon la valeur de getDay()

  let onEst = "On est ";

  switch (jSem) {

    case 0:

      onEst += "dimanche";

      break;

    case 1:

      onEst += "lundi";

      break;

    case 2:

      onEst += "mardi";

      break;

    case 3:

      onEst += "mercredi";

      break;

    case 4:

      onEst += "jeudi";

      break;

    case 5:

      onEst += "vendredi";

      break;

    case 6:

      onEst += "samedi";

      break;

  }

  // l'instruction inner html nous permet de modifier le contenu d'uen balise ou dintégrer un ojet dans une page

  //ici on rajoute donc la phrase "on est[..] suivit du jour correspondant"

  info.innerHTML = onEst;

}

L’autre possibilité, est d’utiliser un tableau. On aura moins de code avec un tableau et il fonctionne de la même manière qu’un tableau en java. La différence est qu’on peut agrandir et rétrécir un tableau javascript.

Pour faire un tableau avec mes jours de la semaine par exemple, je vais déclarer une constante, on va l’appeler « tab » et je vais lui donner entre croches tous les jours de la semaine.

Voici un exemple de déclaration de tableau avec tous les jours de la semaine :

const jourSemaine = ["dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi"];

Pour faire une fonction javascript, on va regarder si un élément « getDay » corresponds à un des index de notre tableau, l’affichage est le même que pour la fonction d’avant. Il faut faire attention ici, car le nombre 1 corresponds en fait au dimanche. Il faudra juste commencer notre tableau par dimanche du coup.

Voici à quoi ressemble la fonction JavaScript, elle utilise le tableau que j’ai déclaré en dessus :

const jourSemaine = [

  "dimanche",

  "lundi",

  "mardi",

  "mercredi",

  "jeudi",

  "vendredi",

  "samedi",

];

function afficherJourSemaineTab() {

  //on va aller rechercher l'élément avec l'ID "info"

  //Ensuite on crée une nouvelle date qui nous donne le numéro du jour de la semaine

  //Puis on crée un élément div pour que l'on puisse mettre en ligne nos deux résultat

  //on passe le jour de la semaien a notre tableau qui aura le jour correspondant à l'index qu'on lui donne vu que tout est dans l'ordre

  //on rajoute au paragraphe avec l'id "info" la nouvelle ligne qui nosu dit tq uel jour on est avec un tableau

  let info = document.getElementById("info");

  let jSem = new Date().getDay();

  let div = document.createElement("div");

  div.innerHTML = "On est " + jourSemaine[jSem] + " via [tableau]";

  info.appendChild(div);

}

A noter que ce n’est pas tout ce qu’on peut faire avec un tableau en javaScript évidemment.

Pour accéder à un élément dans un tableau c’est comme en java, on indique le numéro de l’index de la cellule dont on veut la valeur :

jourSemaine[0] ;

Me redonnerait par exemple **« dimanche ».**

Pour modifier un élément on part de la ligne d’au-dessus et on ajoute la nouvelle valeur :

jourSemaine[0] = « Premier jour de la semaine »;

Change donc Dimanche en **« premier jour de la semaine ».**

Pour ajouter et supprimer des éléments dans un tableau on utilise les méthodes **« push »** et **« pop** ».

**« Push »** av nous permettre de rajouter quelque chose dans notre tableau par exemple :

jourSemaine.push(« je suis un nouveau jour ») ; ;

**« Pop »** va supprimer le dernier élément de la liste, si on reprend la ligne du dessus il va supprimer **« je suis un nouveau jour » :**

jourSemaine.pop();

Pour afficher le contenu d’un tableau, on fait comme en java et on peut utiliser la méthode **« toString() » :**

jourSemaine.toString() ;

Me redonnerais donc : « dimanche, lundi, mardi, mercredi, jeudi, vendredi, samedi ».

## Exercice 7

En se basant sur l’exercice précédent, nous avons dû ici tester plusieurs sortes de boucle en javascript.

La maquette ressemble à ça :

Une image contenant texte, capture d’écran, logiciel, affichage

Description générée automatiquement

On peut voir que chaque bouton correspond à une boucle et qu’en cliquant dessus, on va évidemment les utiliser dans notre code.

Voici à quoi cela doit ressembler si on clique sur le bouton **« for »**

Une image contenant texte, logiciel, affichage, Icône d’ordinateur

Description générée automatiquement

Le bouton **« while »**

Une image contenant texte, logiciel, affichage, Icône d’ordinateur

Description générée automatiquement

Et enfin le bouton **« do while »**

Une image contenant texte, logiciel, Police, Page web

Description générée automatiquement

Je commence par changer un peu le fichier html pour que l’affichage ressemble à celui de la maquette. Voici le nouveau **« body »** du fichier :

<body onload="initCtrl()">

    <div id="container">

      <p>

        Cliquez sur les boutons ci-dessous pour tester les différentes sortes de

        boucles en JavaScript.

      </p>

      <button id="for">For</button>

      <button id="while">While</button>

      <button id="do">Do while</button>

      <p id="info">&nbsp;</p>

    </div>

  </body>

On peut voir que chaque bouton aura donc sa propre fonction qui fonctionnera avec une boucle différente à chaque fois.

Pour commencer, je vais utiliser des écouteurs pour chacun de mes boutons dan cet exercice. Mais comme le script est chargé avant le DOM, il pourrait me donner des erreurs car certains éléments pourraient ne pas être crée.

Je vais donc mettre tous les écouteurs dans la fonction **« onload »** de mon body pour les initialiser

function initCtrl() {

  document.getElementById("for").addEventListener("click", testerFor); // appel de la function testerFor avec un événement "click"

  let butWhile = document.getElementById("while");

  butWhile.addEventListener("click", testerWhile); // appel de la function testerWhile avec un événement "click"

  let butDo = document.getElementById("do");

  butDo.addEventListener("click", testerDoWhile); // appel de la function testerDoWhile avec un événement "click"

}

On va commencer par la boucle **« For ».** Elle va nous permettre d’itérer des éléments x fois. Avec cette boucle on sait combien de fois on veut et on va itérer notre élément.

Voici à quoi ressemble mon code :

function testerFor() {

  let info = document.getElementById("info");

  let div = document.createElement("div");

  div.innerHTML += "for (let i = 0; i < 5; i++) {...} <br>";

  for (let i = 0; i < 5; i++) {

    div.innerHTML += " i= " + i + "<br>";

  }

  div.innerHTML +=

    "--> A utiliser si on sait que l'on veut itérer x fois (x connu avant de commencer la boucle)";

  info.appendChild(div);

}

On peut aussi voir que j’ai utilisé une balise **« <br> »** qui sert à faire un retour à la ligne à l’affichage. J’ai testé directement la boucle avec l’affichage et la condition est que jusqu’à ce que i soit plus grand que 5 j’affiche sa valeur et je l’augmente de 1. Le chiffre **5** ne s’affichera donc jamais, si j’avais voulu l’afficher j’aurais dû mettre un **« <= »**

Pour la boucle **« while »**, on va l’utiliser si on n’a pas défini ce nombre d’itération ou si on ne sait pas combien d’itérations vont se faire.

Dans ma boucle ci-dessous, je vais donc afficher la valeur de **i** tant qu’elle est plus petite que **5** et l’augmenter d’un à chaque fois.

function testerWhile() {

  let info = document.getElementById("info");

  let div = document.createElement("div");

  let i = 0;

  div.innerHTML += "while(i<5){...} <br>";

  while (i < 5) {

    div.innerHTML += "i= " + i + "<br>";

    i++;

  }

  div.innerHTML +=

    "--> A utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle";

  info.appendChild(div);

}

Et finalement, pour la boucle **« do…while »,** On va aussi l’utiliser quand le nombre d’itérations nous est inconnu mais contrairement à la boucle **« while »** ici, on va obligatoirement passer au moins une fois dans la boucle.

Pour la boucle ci-dessous, je vais afficher la valeur de **i** tant qu’elle est plus petite que **5**. Si la valeur avait été plus grande, elle serait quand même affichée car comme dit au-dessus, un passage est obligatoire au minimum dans cette boucle.

function testerDoWhile() {

  let info = document.getElementById("info");

  let div = document.createElement("div");

  let i = 0;

  div.innerHTML += "do{...} while(i<5); <br>";

  do {

    div.innerHTML += "i= " + i + "<br>";

    i++;

  } while (i < 5);

**div.innerHTML +=**

    "--> A utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle mais avec un passage obligatoire";

  info.appendChild(div);

}

## Exercice 8

On va partir sur la base de l’exercice précédent pour tester les tableaux en Json. On commence par supprimer tout le code javascript de l’exercice précédent pour repartir de 0.

On va tester une imbrication de deux boucles sur les données d’un tableau de personne défini par un objet de type JSON.

Sa syntaxe ressemble fortement à du javascript. Ce qui lui donne un avantage car il sera donc plus compréhensible de ce point.

Voici un exemple pour illustrer tout ça

// JSON exemple

const personne = {

  prenom: "Jules",

  nom: "Tartampion",

  age: 25

};

Ci-dessus, on peut voir que par exemple, je donne les informations d’une personne.

On utilise l’opérateur **« in »** dans un **« for »** pour balayer tous les champs **« f »** (field) de l’objet. Exemple :

for (let f in personne) {

  console.log("field: " + f + ", valeur: " + personne[f]);

}

On peut aussi faire des tableaux d’objets, voici un exemple avec des personnes :

const json = {

  personnes: [

    {prenom: "John", nom: "Doe", age: 44},

    {prenom: "Anna", nom: "Smith", age: 32},

    {prenom: "Peter", nom: "Jones", age: 29}

  ]

};

On parcourt un tableau avec une boucle **« for »** comme en java.

for (let i = 0; i < json.personnes.length; i++) {

  let personne = json.personnes[i];

// ...

}

Donc pour en revenir à l’exercice, on va tester la théorie ci-dessus, on va faire une application qui va nous permettre de parcourir un tableau JSON.

Voici la maquette de l’application :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Et une fois que le bouton a été pressé on devrait avoir ceci :

Une image contenant texte, capture d’écran, affichage, Police

Description générée automatiquement

On commence par changer un peu le code html pour qu’il corresponde à la maquette. Voici le nouveau body de l’application :

<body>

  <div id="container">

    <p>Cliquez sur le bouton pour parcourir un tableau JSON</p>

    <button onclick="parcourirUnTableauJSON();">

Parcourir un tableau JSON

</button>

    <p id="info">&nbsp;</p>

  </div>

</body>

On peu voir que l’on va donc devoir créer une fonction **« parcourirUnTableauJSON() »** dans notre fichier javascript.

J’ai donc créé un tableau de personne que j’ai appelé **« lesPersonnes »**, je l’ai rempli avec les trois personnes de la consigne et ensuite j’ai fait une boucle for sur les personnes de mon tableau en faisant **« lesPersonnes.personnes ».** J’ai fait une variable **« personne »** à qui je donne la valeur de **« lesPersonnes.personnes[i] ».** Je vais prendre l’index de ces personnes représenté par **« i »** et je vais rajouter un point à côté pour que l’affichage soit le même que dans la consigne. Je vais ensuite faire une boucle dans ces nouvelles personnes et je vais simplement les afficher avec un espace. Pour finir, je rajoute un retour à la ligne avec la balise **« br »** et je rajoute mes lignes dans mon fichier html avec **« innerHTML ».**

Voici à quoi ressemble le code de ma fonction :

function parcourirUnTableauJSON() {

  //on déclare un tableau de personne et on le remplit

  const lesPersonnes = {

    personnes: [

      { prenom: "John", nom: "Doe", age: 44 },

      { prenom: "Anna", nom: "Smith", age: 32 },

      { prenom: "Peter", nom: "Jones", age: 29 },

    ],

  };

  let res = ""; // je fais une variable résultat pour pouvoir y stocker l'affichage de mes personnes

  for (i = 0; i < lesPersonnes.personnes.length; i++) {

    let personne = lesPersonnes.personnes[i];

    res += i + ". "; //je rajoute un "." après el numéro d'index pour l'affichage

    for (let f in personne) {

      res += personne[f] + " ";

    }

    res += "<br>"; //je fais un retour à la ligne

  }

  document.getElementById("info").innerHTML = res; // on rajoute les 3 lignes de mes personnes dans l'élément html avec l'id "info"

}

## Exercice 9

Dans l’exercice précédent, nous avons vu comment créer un objet avec la notation JSON. Il y a une 2e façon de faire en créant une instance à partir de la fonction **« Object ».** Ensuite, on peut définir des classes pour créer des objets, on a même 3 manières de faire.

Cet exercice est une démonstration de ces différentes manières de faire.

Pour commencer sans utiliser de classe, pour créer un objet on le déclare simplement comme suis :

const p1 = {

prenom: "Jules",

nom: "Tartampion",

age: 25

};

Et si on veut rajouter par exemple un attribut a cet objet **« p1 »** on fait :

p1.ville = "Fribourg" ;

Une autre manière de créer un objet, est d’utiliser la fonction **« object ».** On va donc faire des **« new() ».**

const p2 = new Object();

p2["prenom"] = "Juliette";

p2["nom"] = "Tartampion";

p2["age"] = 23;

Javascript gère les attributs d’un objet un peu comme des éléments d’un hashmap (clé-valeur). Grace à ça on peut ajouter, supprimer ou même modifier ceux-ci. Autre chose d’intéressant est que comme on l’a déjà mentionné, les variables sont non typées en javascript. Ça nous permet de stocker un peu tout ce que l’on veut dans un objet.

Avec ça, on va donc pouvoir surcharger des méthodes et/ou en ajouter. On peut par exemple surcharger la méthode **« toString »** en remplaçant l’attribut nommé **« toString »** de l’objet.

Voici une première manière de le faire qui est la plus courante :

p1.toString = function() {

return this.prenom + " " + this.nom;

};

Et une deuxième qui est plutôt rare à rencontrer :

p2["toString"] = function() { // très rare

return this.prenom + " " + this.nom;

};

On peut aussi créer un objet avec une **« Function Objects »,** ça ressemble presque à une fonction java, on peut aussi utiliser la méthode to string et on se rapproche de ce que l’on a vu plus haut.

function Personne(prenom, nom, age) {

this.prenom = prenom;

this.nom = nom;

this.age = age;

this.toString = function(){

return this.nom + " " + this.prenom + " (" + this.age + ")";

}

}

On peut donc maintenant instancier un objet avec un **« new »** comme en java, mais le problème revient le même qu’avant, la fonction **« toString »** fait partie de l’objet et elle est donc copié pour chaque nouvel objet. Malgré cela, il est possible e rajouter des fonctions par après.

Pour éviter la duplication des fonctions, if faut utiliser la programmation par prototypage grâce à l’attribut **« prototype »** qui se trouve dans tout objet JS. Voici un exemple et cette fois la fonction a disparu de l’objet, on s’approche de ce qui est fait en Java :

class Personne {

constructor(prenom, nom, age) {

this.prenom = prenom;

this.nom = nom;

this.age = age;

}

toString() {

return this.nom + " " + this.prenom + " (" + this.age + ")";

}

}

## Exercice 10

Voici la consigne de l’exercice :

Depuis votre outil de développement, veuillez copier l’exercice « ex03… » vers « ex10-js-poo ». Supprimez tout le code JavaScript, pour le reconstituer plus tard avec les informations qui vont suivre. Cet exercice est l’un des plus importants de l’apprentissage de JavaScript, car il concerne la **« programmation orientée objets »** (POO) et la structuration de l’application en **« modules-fichiers »** de type MVC.

Le but est ici de construire une application qui permette de gérer simplement les informations d’une liste de personnes (prénom, nom, âge) avec des opérations **« métier »** comme **« ajouter une personne »** ou **« supprimer une personne »** que nous mettrons dans un fichier **« worker.js ».** Pour gérer des personnes, il faut également disposer d’un bean **« Personne »** (à la Java) que nous stockerons dans un fichier **« personne.js »**.

Vous pouvez dès à présent créer ces 2 fichiers JavaScript, mais vides pour le moment.

Nous ferons toute la partie « contrôle » de l’affichage et les interactions utilisateur dans le contrôleur nommé **« indexCtrl.js »**

Voici à quoi ressemble la maquette :

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Je commence donc par créer mes deux fichiers, **« personne.js »** qui représentera mon bean personne et **« worker.js »** qui sera le worker de mon application.

Il faut rajouter dans la partie **« head »** du fichier html les informations qui vont me permettre de charger les deux fichiers JavaScript que j’ai créé.

J’ai donc juste rajouté les deux lignes suivantes dans mon fichier :

<script src="js/worker.js" ></script>

<script src="js/indexCtrl.js" ></script>

Il va ensuite falloir transformer noter formulaire. Donc, la balise **« form »** n’a plus besoin de propriété **« action »** car on a plus de fichier PHP à gérer.

Les propriétés **« name »** des balises **« input »** peuvent être modifiées en **« id »**, car on a plus à envoyer une requête HTTP de type GET ou POST, mais on devra récupérer ces informations dans le contrôleur avec des **« document.getElementById(id).value ».**

Il n’y aura plus de bouton de type **« submit »** non plus, car encore une fois, on a plus de formulaire à envoyer par HTTP(S).

Voici à quoi ressemble notre nouveau **« body »**:

<body>

<div id="container">

<form class="user-form">

<fieldset>

<legend>Veuillez introduire une nouvelle personne

ou sélectionner une existante&nbsp;:</legend>

<div class="field">

<label for="prenom">Prénom:</label>

<input type="text" size="20" id="prenom"

placeholder="prénom" required="required" autofocus>

</div>

<div class="field">

<label for="nom">Nom:</label>

<input type="text" size="20" id="nom"

placeholder="nom" required="required">

</div>

<div class="field">

<label for="nom">Âge:</label>

<input type="text" size="20" id="age"

placeholder="âge" required="required">

</div>

<div class="button">

<input type="button" value="Ajouter" onclick="ajouter();">

<input type="button" value="Supprimer" onclick="supprimer();">

<input type="reset" value="Nettoyer">

</div>

</fieldset>

</form>

<div id="info"></div>

</div>

</body>

Il est maintenant temps de regarder le diagramme d’activités, voici à quoi il ressemble :

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

On va commencer par faire le bean **« personne »**, comme en java les beans ont des propriétés et des méthodes. On va lui donner 3 attributs, un nom, un prénom et un âge qu'il va falloir écrire avec, en plus, une fonction (prototype) toString(). Chaque fois qu’on utilise un attribut dans le bean, on va devoir le précéder par le mot-clé **this**.

Voici à quoi ressemble mon bean :

function Personne(prenom, nom, age) {

  this.prenom = prenom;

  this.nom = nom;

  this.age = age;

}

Personne.prototype.toString = function () {

  return this.prenom + " " + this.nom + " " + this.age;

};

Ensuite, dans la couche métier on va devoir faire une fonction privée, qui va nous permettre d’ajouter et de supprimer une personne. La fonction prendra la forme de **« \_trouverPersonne(p) »**, On teste donc que si une personne doit être ajouté à la liste, on regarde si elle n’y est pas déjà et même chose si on doit en supprimer une, on teste si elle existe dans notre liste.

Cette fonction **« \_trouverPersonne »** peut retourner l’index (idx=0, 1, 2 ...) dans la liste de la personne recherchée ou éventuellement -1 si elle n’a pas été retrouvée. Cela permettra de la supprimer facilement avec la fonction **« .splice(idx, 1) »** disponible sur les tableaux. La fonction **« .push(p) »** permettra d’ajouter une personne à la liste.

On ne commence par créer notre liste de personnes :

// définition du modèle de données, créé au chargement du js dans le index.html

const personnes = [

new Personne("John", "Doe", 44),

new Personne("Anna", "Smith", 32),

new Personne("Peter", "Jones", 29)

];

On utilise la méthode « sort » pour trier nos personnes :

// premier tri de la liste de personnes

personnes.sort();

On fait une fonction privée pour retrouver l’index d’une personne dans le tableau. Je parcours mes personnes et je teste si l’index de la personne est égal à l’index que je recherche.

Voici à quoi ressemble cette première fonction privée :

 // fonction privée pour retrouver l'index d'une personne dans le tableau, -1 autrement

  // il faut comparer avec toString()

  function \_trouverPersonne(p) {

    let idx = -1;

    for (let i = 0; i < personnes.length; i++) {

      if (p.toString() === personnes[i].toString()) {

        idx = i;

        break;

      }

    }

    return idx;

  }

Je fais ensuite une deuxième fonction privée qui va me permettre d’ajouter une personne dans mon tableau. Je rajoute cette personne grâce à son index et je retrie mon tableau. Pour tester si elle existe, je me sers de la fonction que j’ai créée précédemment, si elle me retourne « -1 » cela veut dire que la personne n’existe pas dans mon tableau et que donc je peux l’ajouter.

Voici le résultat :

// ajouter une personne dans la liste des personnes si pas trouvée

  function ajouterPersonne(p) {

    let idx = \_trouverPersonne(p);

    if (idx < 0) {

      personnes.push(p);

      personnes.sort();

    }

  }

Finalement, pour supprimer une personne, je teste via la méthode **« trouverPersonne »** que la personne que je veux supprimer existe bien dans mon tableau, pour cela, la méthode doit me retourner l’index de la personne que je veux supprimer, si cette personne existe bien, je la supprime de mon tableau via **« splice() »**

/ supprimer une personne dans la liste des personnes si trouvée

  function supprimerPersonne(p) {

    let idx = \_trouverPersonne(p);

    if (idx >= 0) {

      personnes.splice(idx, 1);

    }

  }

Pour le contrôleur, on peut faire comme en java FX et el séparer en 3 parties. La première sera le démarrage de l’application avec ma fonction **« document.onreadystateChange »**, L'attribut **onreadystatechange** est en fait un événement qui se déclenche à chaque fois que l'attribut **readyState** change de valeur.

/\*

\* 1. DOM PRET : DEMARRAGE DE L'APPLICATION

\*/

document.onreadystatechange = function () {

if (document.readyState === "complete") {

\_afficherPersonnes();

}

};

En deuxième partie, les méthodes privées de lecture et d’écriture de la vue. On va donc retrouver une fonction qui av afficher mes personnes, une qui va afficher les infos d’une personne dans le formulaire et une qui va lire le contenu de ce qui a été entré dans le formulaire pour créer une nouvelle personne.

Je commence par la méthode qui va afficher la liste de données au bas de la vue, je fais une liste pour ça et je vais rajouter des éléments dans cette liste. Je parcours mon tableau via une boucle **« for »**, j’utilise la méthode **« selectionnerPersonne() »** quand je clique sur el bouton, je lui spécifie la personne courante de la boucle et ensuite j’affiche cette personne.

// affiche la liste des données au bas de la vue (avec du HTML généré)

function \_afficherPersonnes() {

  let txt = "<ul>";

  for (let i = 0; i < personnes.length; i++) {

    txt += '<li><a href="#" onclick="selectionnerPersonne(' + i + ');">' + personnes[i] + '</a></li>';

  }

  txt += "</ul>";

  document.getElementById("info").innerHTML = txt;

}

Pour afficher une personne, je me sers simplement de son « id » et je dis que c’est égal à l’attribut correspondant, le champ nom est égal à l’attribut nom etc...

// affiche les infos d'une personne dans le formulaire

function \_afficherInfosPersonne(p) {

  document.getElementById("prenom").value = p.prenom;

  document.getElementById("nom").value = p.nom;

  document.getElementById("age").value = p.age;

}

Pour lire les informations, je lis les informations entrées dans les champs de mon formulaire et avec ce qui est écrit je vais créer une nouvelle personne et l’afficher dans mon tableau.

/ lit le contenu des masques de saisie pour en faire une personne

function \_lireInfosPersonne() {

  let p = null;

  let prenom = document.getElementById("prenom").value;

  let nom = document.getElementById("nom").value;

  let age = parseInt(document.getElementById("age").value);

  if (prenom.length > 0 && nom.length > 0 && age > 0) {

    p = new Personne(prenom, nom, age);

  }

  return p;

}

La troisième partie, sont les méthodes publiques qui sont nécessaires à la vue., la première méthodes « selectionnerPersonne() », va me servir à récupérer les informations d’une personne, via une autre méthode.

// appelée depuis la vue pour afficher les données de la personne sélectionnné

function selectionnerPersonne(i) {

  \_afficherInfosPersonne(personnes[i]);

}

La deuxième fonction me sert à ajouter une personne. Je lis les infos de cette personne, je l’ajoute et finalement je l’affiche.

// appelée depuis la vue pour ajouter une personne

function ajouter() {

  let p = \_lireInfosPersonne();

  ajouterPersonne(p);

  \_afficherPersonnes();

}

Finalement, la dernière me sert à supprimer une personne sur le même principe que la fonction précédente.

// appelée depuis la vue pour supprimer une personne

function supprimer() {

  let p = \_lireInfosPersonne();

  supprimerPersonne(p);

  \_afficherInfosPersonne();

}

## Exercice 11

En partant de l’exercice 10, le défi d’ici est de transformer les modules de code de cet exercice en créant des classes pour **«Personne », « worker » et « Ctrl ».**

Ensuite, la seule grande difficulté est de savoir comment créer un objet contrôleur pour qu’il soit disponible pour la vue ? La solution est donnée dans la consigne, l’objet doit être stocké dans l’objet **« Window »** du navigateur par ce que les méthodes de **« indexCtrl »** sont appelés depuis « index.html ». L’objet **« window »** doit être ajouté à la variable ou on ne met rien et la variable devient globale. On ne fait surtout pas de let.

/\*

\* DOM PRET : DEMARRAGE DE L'APPLICATION dans le fichier indexCtrl.js

\*/

document.onreadystatechange = function () {

if (document.readyState === "complete") {

window.ctrl = new Ctrl(); // ou ctrl = new Ctrl();

ctrl.afficherPersonnes();

}

};

Important: il faut faire référence aux attributs et méthodes de la classe en utilisant le mot-clé **this**. Comme c’est le Ctrl qui crée le Worker, on peut accéder aux attributs en utilisant **this.wrk.personnes** ou **this.wrk.ajouterPersonne(p)** pour une méthode.

/\*

 \* Première ligne de la classe Ctrl

 \*/

class Ctrl {

  constructor() {

      this.wrk = new Worker();

      this.afficherPersonnes();

}

Je vais ensuite simplement mettre toutes les fonctions qu’il y avait en dessous dans ma classe Ctrl, le mot clé **« function »** va donc disparaitre. Il va falloir faire attention à où je dois appeler les choses depuis mon worker.

Voici à quoi cela ressemble :

/\*

 \* Première ligne de la classe Ctrl

 \*/

class Ctrl {

  constructor() {

    this.wrk = new Worker();

    this.afficherPersonnes();

  }

  /\*

   \* 2. METHODES PRIVEES DE LECTURE/ECRITURE DANS LA VUE

   \*/

  // affiche la liste des données au bas de la vue (avec du HTML généré)

  afficherPersonnes() {

    let txt = "<ul>";

    for (let i = 0; i < this.wrk.personnes.length; i++) {

      txt +=

        '<li><a href="#" onclick="ctrl.selectionnerPersonne(' +

        i +

        ');">' +

        this.wrk.personnes[i] +

        "</a></li>";

    }

    txt += "</ul>";

    document.getElementById("info").innerHTML = txt;

  }

  // affiche les infos d'une personne dans le formulaire

  afficherInfosPersonne(p) {

    document.getElementById("prenom").value = p.prenom;

    document.getElementById("nom").value = p.nom;

    document.getElementById("age").value = p.age;

  }

  // lit le contenu des masques de saisie pour en faire une personne

  lireInfosPersonne() {

    let p = null;

    let prenom = document.getElementById("prenom").value;

    let nom = document.getElementById("nom").value;

    let age = parseInt(document.getElementById("age").value);

    if (prenom.length > 0 && nom.length > 0 && age > 0) {

      p = new Personne(prenom, nom, age);

    }

    return p;

  }

  // appelée depuis la vue pour afficher les données de la personne sélectionnée

  selectionnerPersonne(i) {

    this.afficherInfosPersonne(this.wrk.personnes[i]);

  }

  // appelée depuis la vue pour ajouter une personne

  ajouter() {

    let p = this.lireInfosPersonne();

    this.wrk.ajouterPersonne(p);

    this.afficherPersonnes();

  }

  // appelée depuis la vue pour supprimer une personne

  supprimer() {

    let p = this.lireInfosPersonne();

    this.wrk.supprimerPersonne(p);

    this.afficherPersonnes();

  }

}

Attention à ne pas oublier de faire référence au « ctrl » dans l’html quand on va appeler les fonctions.

<input type="button" value="Ajouter" onclick="ctrl.ajouter();">

<input type="button" value="Supprimer" onclick="ctrl.supprimer();">

On va ensuite faire la classe **« personne »** sur le même principe que pour **« ctrl ».** On a toujours les mêmes attributs et notre **« toString ».**

class Personne {

  constructor(prenom, nom, age) {

    this.prenom = prenom;

    this.nom = nom;

    this.age = age;

  }

  toString() {

    return this.nom + " " + this.prenom + " (" + this.age + ")";

  }

}

Et finalement, on va s’occuper de la classe **« worker »,** on met noter tableau de personne dans son constructeur et on adapte les fonctions déjà existantes avec le mot clé **« this ».**

class Worker {

  constructor() {

    this.personnes = [

      new Personne("John", "Doe", 44),

      new Personne("Anna", "Smith", 32),

      new Personne("Peter", "Jones", 29),

    ];

    this.personnes.sort();

  }

  // fonction privée pour retrouver l'index d'une personne dans le tableau, -1 autrement

  \_trouverPersonne(p) {

    let idx = -1;

    for (let i = 0; i < this.personnes.length; i++) {

      if (p.toString() === this.personnes[i].toString()) {

        idx = i;

        break;

      }

    }

    return idx;

  }

  // ajouter une personne dans la liste des personnes

  ajouterPersonne(p) {

    let idx = this.\_trouverPersonne(p);

    if (idx < 0) {

      this.personnes.push(p);

      this.personnes.sort();

    }

  }

  // supprimer une personne dans la liste des personnes

  supprimerPersonne(p) {

    let idx = this.\_trouverPersonne(p);

    if (idx >= 0) {

      this.personnes.splice(idx, 1);

    }

  }

}

## Exercice 12

Dans cet exercice, nous allons voir les différentes manières d’écrire une fonction.

La première méthode ci-dessous, est exécutée et affiche 1 dans la console avec un simple **« console.log »**, rien de très compliqué.

function a() {

let val = 1;

console.log(val) ;

}

a();

A noter que si on écrit la dernière ligne avant la fonction, cela fonctionne aussi !

La deuxième méthode nous permet de déclarer une fonction dite **« anonyme »** on se sert d’une variable **« b »** et pas directement du mot clé **« function ».**

function a() {

let val = 1;

console.log(val) ;

}

a();

La troisième méthode, nous permet de déclarer une fonction **« flèche »** ou en anglais **« arrow function »** pour faire ça on se sert du caractère **« => »** pour faire une flèche et indiquer comment on arrive au résultat d’une variable par exemple.

// Déclaration d’une fonction flèche (arrow function) ; 3 cas de figure

let c = (a,b) => a + b ;

console.log(c(2,5)) ; // = 7

let d = val => val\*val ; // avec un paramètre

console.log(d(5)); // = 25

let e = (a,b) => { // avec plusieurs instructions

let somme = a+b;

return somme;

};

console.log(e(5,7)); // = 12

Il est, quelquefois, nécessaire de créer une fonction et l’exécuter directement : IIFE **« Immediately-Invoked Function Expression»**. Voici deux manières d’écrire une fonction JS IIFE:

(function() {

let val = 3;

console.log(val) ;

})() ; // => exécution et affichage de 3 dans la console

(() => {

let val = 4;

console.log(val) ;

})() ; // => exécution et affichage de 4 dans la console via une fonction flèche

Pour l’exercice, j’ai réalisé un **« additionneur »,** j’ai trois boutons sur mon interface et quand j’appuie dessus, leur valeur s’incrémente de 1.

Pour réaliser cette petite application j’ai fait un fichier html simple avec les 3 boutons :

<html>

    <!--

   But :  page index de l'exe 12

   Auteur : Malori Burgy

   Date :   22.05.23 / V1.0

   -->

  <!-- entête de la page -->

   <head>

      <meta charset="UTF-8" />

      <meta name="viewport" content="width=device-width, initial-scale=1.0" />

      <link rel="icon" href="#"/>

      <link rel= "stylesheet" href= "styles/main.css" >

      <script type="text/javascript" src="js/indexCtrl.js" async></script>

      <title>Ex12</title>

   </head>

   <!-- corps de la page -->

   <body>

    <div id="container">

      <p>Cliquez pour incrémenter les compteurs</p>

      <button id="button1" onclick="actionCompteur(1);">Compteur No 1</button>

      <button id="button2" onclick="actionCompteur(2);">Compteur No 2</button>

      <button id="button3" onclick="actionCompteur(3);">Compteur No 3</button>

    </div>

  </body>

</html>

En ce qui concerne le fichier javascript, j’ai réalisé une fonction **« additionne »** qui va incrémenter la valeur de mes boutons de 1 à chaque fois que je clique dessus. Pour que je puisse faire une fonction pour 3 boutons, je me suis servi des index, selon l’index qui est cliqué j’incrémente le bouton correspondant.

function additionne(){

  let count=0;

  return function(){

    return count++;

  };

}

let mesBoutons = [additionne(), additionne(), additionne()];

function actionCompteur(idx){

  let val = mesBoutons[idx-1].call();

  document.getElementById("button"+idx).innerHTML="Compteur No "+idx+": <b>"+val+"</b>";

}

## Exercice 13

Dans cet exercice, nous allons introduire le concept de cookies. Par l’intermédiaire de cookies, il est possible d’écrire sur le poste de l’internaute des informations de manière permanente. Les cookies qui se présentent comme de petits fichiers texte peuvent entreposer un nombre limité de données mais cette technique peut se révéler précieuse pour conserver des contenus de variables mémoires réutilisables au fil de la navigation au travers des nombreuses pages de votre site web (conservation d’identifiants, de mots de passe même si cela est dangereux, de préférences utilisateur...).

Il est évidemment possible sur maintenant presque tous les sites d’interdire que les cookies soient utilisables pour cibler nos préférences sur un site de vêtements par exemple.

En Javascript, on peut créer un cookie avec **« document.cookie »** il nous suffit de l’affecter à une chaine de caractères. Il existe une multitude de paramètres pour définir les cookies.

Dans le cadre de cet exercice, on va retenir les 3 principaux :

**Le nom :** un même site peut enregistrer plusieurs cookies différents il faut donc savoir les identifier

**Le contenu :** c’est l’élément qu’on veut stocker dans ce cookie et qu’on récupérera plus tard

**La date d’expiration :** c’est la date à partir de laquelle le cookie cessera d’exister.

Voici un exemple de création d’un cookie :

document.cookie = "cookie\_exercice\_13=" + contenu + "; expires=" + dateExpiration.toUTCString() + "; path=/";

Pour récupérer ce qui est stocké dans le cookie, il faut lire le contenu de l’objet **« document.cookie »** il nous sera ensuite possible d’extraire les informations dont on a besoin.

Pour l’exercice, il est question de récupérer le nom et le prénom que l’utilisateur saisira dans un champ texte via un cookie. L’enregistrement du cookie est déclencher quand l’utilisateur appuie sur le bouton **« enregistrer »** La date d'expiration du cookie doit être de +3 minutes par rapport à la date et l'heure actuelles. Le nom du cookie set "cookie\_exercice\_13".

Au démarrage de la page, il faut vérifier si un cookie existe. Si c'est le cas, il faut lire le contenu du cookie afin de récupérer le prénom et le nom qui y sont stockés. Une fois ces 2 informations récupérées, il faut les réafficher dans le formulaire.

Voici la maquette de la vue :

Une image contenant texte, capture d’écran, Rectangle, Police

Description générée automatiquement

Je rempli donc le fichier javascript fournie en me servant des commentaires présents dans le code et de la théorie ci-dessus.

Je récupère les valeur des champs « noms » et **« prénoms »,** je crée un objet json avec les valeurs que je viens de récupérer. Je transforme cet objet json en chaine de caractères et j’utilise la méthode **« setCookie »** pour créer mon cookie.

function saveData() {

  // Récupérer les valeurs contenues dans les champs "prenom" et "nom".

  let nom = document.getElementById("nom").value;

  let prenom = document.getElementById("prenom").value;

  // Créer un objet JSON "personneJson" contenant le prénom et le "nom".

  let personneJson = { nom: nom, prenom: prenom };

  // Convertir l'objet JSON "personneJson" en chaîne de caractères.

  let stringPersonne = JSON.stringify(personneJson);

  //nom du cookie

  // Appeler la méthode "setCookie" en passant en paramètre la chaîne de caractères "stringPersonne" convertie et en précisant le délai de 3 minutes.

  setCookie(stringPersonne, 3);

}

Je génère dans la fonction ci-dessous, je génère une date d’expiration, je crée d’abord une date pour ça et ensuite je lui rajoute les minutes que j’ai en paramètres. Ces deux choses ensemble me feront ma date d’expiration. Je crée le cookie avec el contenu que j’ai en paramètre et la date d’expiration que je viens de créer.

function setCookie(contenu, minutes) {

  // Générer une date d'expiration. Il doit s'agir de la date et de l'heure actuelles + le nombre de minutes passé en paramètre.

  const d = new Date();

  d.setTime(d.getTime() + minutes \* 60000);

  let expire = "expires=" + d.toUTCString();

  // Créer le cookie.

  document.cookie ="monCookie=" + contenu + "; " + expire + ";path=/";

  console.log("Données enregistrées dans le cookie");

}

Pour ma fonction **« getCookie »** on va d’abord tester si le cookie existe bien regardant si sa taille est supérieure à 0. Ensuit on va séparer noter cookie où il y a un égal et on va transformer ce qu’il se trouve à droite de cela en objet Json. J’affiche mon cookie pour pouvoir m’aider dans le débugga mais cela n’est pas obligatoire. Je fini donc par juste remplir mes champs avec les attributs de mon objet json et si on refresh la page les champs auront garder en mémoire ce qu’il y a été tapé dedans.

function getCookie() {

  if (document.cookie.length > 0) {

    let cookieSplit = document.cookie.split("=");

    let cookieJson= JSON.parse(cookieSplit[1]);

    console.log("ContenuTxt : " + cookieSplit);

    document.getElementById("prenom").value = cookieJson.prenom;

    document.getElementById("nom").value = cookieJson.nom;

  }

  console.log("Data retrieved from cookie");

}

## Exercice 14

Pour cet exercice, nous allons aborder les bases de JQuery, ce que c’est et à quoi ça sert.

Donc, JQuery est une librairie qui va simplifier l’interaction entre Html et JavaScript. Elle à été créer pour simplifier les manipulations du DOM et pour simplifier la vie des développeurs en général. JQuery est très bien documenté, il est facile de trouver des informations en ligne, elle comporte une grande communauté de développeurs aussi ce qui la rends intéressante. Elle est aussi compatible avec de nombreux navigateurs et une multitude de plugins est disponible.

Un objet JQuery est précéder du signe **« $ »,** en réalité, c’est une fonction JavaScript. On peut rechercher des éléments dans une page à l’aide d’un sélecteur css par exemple un **« div ».** Don si on note **« $(« div ») »** cela va nous retourner tous les éléments **« div »** on va pouvoir par après par exemple lui rajouter une classe avec un **« addclass(..) ».**

Voici ci-dessous un exemple qui va rechercher tous les **« div »** et qui va leur rajouter une classe qui s’appelle ici **« demo »**

[$](http://api.jquery.com/jQuery)("div").[addClass](http://api.jquery.com/addClass)("demo");

Pour manipuler une page html on doit attendre qu’elle soit prête et qu’elle ait été chargé entièrement. On va utiliser en JQuery l’évènement **« ready »**

$(document).[ready](http://api.jquery.com/ready)(function(){   
    // le code jQuery doit se trouver ici   
});

Parfois une sélection par sélecteurs CSS n'est pas suffisant. JQuery propose une suite de méthodes pour parcourir le DOM.

Il en existe plusieurs : .parent(), .next(), .prev(), .children(), .siblings() ...

Voici un exemple :

[$](http://api.jquery.com/jQuery)("button").[parent](http://api.jquery.com/parent)().[css](http://api.jquery.com/css)("border", "3px solid red");

On peut aussi enchainer les actions sur un set d’éléments. Chaque méthode jQuery retourne un set d'élément jQuery pour de futures actions (à moins qu'une valeur soit retournée).

Exemple :

[$](http://api.jquery.com/jQuery)("div").[hide](http://api.jquery.com/hide)();   
[$](http://api.jquery.com/jQuery)("div").[hide](http://api.jquery.com/hide)().[css](http://api.jquery.com/css)("color","blue");   
[$](http://api.jquery.com/jQuery)("div").[hide](http://api.jquery.com/hide)().[css](http://api.jquery.com/css)("color","blue").[remove](http://api.jquery.com/remove)();

On peut faire pleins d’autres choses avec JQuery, on peut rajouter des événements pour faire des animations ou changer des couleurs etc…

Voici quelques exemples :

Ici on peut rajouter quelque chose après un éléments quand mon bouton est cliqué :

[$](http://api.jquery.com/jQuery)("a[target=\_blank]")   
  .[after](http://api.jquery.com/after)("<img src='images/open.png'/>");

On peut aussi appondre du texte :

[$](http://api.jquery.com/jQuery)("a[target=\_blank]")   
  .[append](http://api.jquery.com/append)("(mon texte)");

On peut rajouter du css à un certain élément :

[$](http://api.jquery.com/jQuery)("li a").[css](http://api.jquery.com/css)({   
  color: "red",   
  fontWeight: "bold"   
});

On peut sélectionner certains éléments HTML pour pouvoir les modifier :

[$](http://api.jquery.com/jQuery)("<li><a></a></li>")   
  .[find](http://api.jquery.com/find)("a")   
    .[attr](http://api.jquery.com/attr)("href", "http://www.emf.ch/")   
    .[html](http://api.jquery.com/html)("EMF")   
  .[end](http://api.jquery.com/end)()   
  .[appendTo](http://api.jquery.com/appendTo)("ul");

On peut même faire des formulaires avec l’événement **« submit »** :

[$](http://api.jquery.com/jQuery)("form").[submit](http://api.jquery.com/submit)(function(){   
  if ( [$](http://api.jquery.com/jQuery)("#name").[val](http://api.jquery.com/val)() === "" ) {   
    [$](http://api.jquery.com/jQuery)("span.help").[show](http://api.jquery.com/show)();   
    return false;   
  }   
});

Et comme dit plus haut on peut animer des choses :

[$](http://api.jquery.com/jQuery)("div.block").[animate](http://api.jquery.com/animate)({   
  fontSize: "2em",   
  width: "+=20%",   
  backgroundColor: "green"   
});

On peut cacher ou montrer des choses sur notre interface :

[$](http://api.jquery.com/jQuery)("div.block").[hide](http://api.jquery.com/hide)("slow", function(){   
  $(this).[show](http://api.jquery.com/show)("slow");   
});

Avec un plugin nommer « Ajax », il nous est possible du lire du contenu XML, sur le même principe que les méthodes précédentes :

[$](http://api.jquery.com/jQuery)("div.block").[hide](http://api.jquery.com/hide)("slow", function(){   
  $(this).[show](http://api.jquery.com/show)("slow");   
});

On peut aussi lire du JSON :

$.[getJSON](http://api.jquery.com/jQuery.getJSON)("file.json", function( obj ) {   
    for ( var prop in obj ) {   
        [$](http://api.jquery.com/jQuery)("ul").[append](http://api.jquery.com/append)(   
            "<li>" + prop + ": " + obj[prop] + "</li>");   
    }   
});

On peut charger des fichiers avec la méthode **« .load() » :**

[$](http://api.jquery.com/jQuery)("div.load").[load](http://api.jquery.com/load)("file.html");

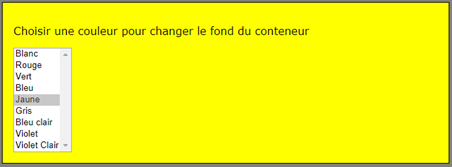
Pour rajouter un plugin, il nous suffit de mettre son lien dans une balise **« script »** comme pour un fichier css :

<script src='http://code.jquery.com/jquery.js'></script>

## Exercice 15

On va partir de l’exercice 1 et changer un peut la vue via JQuery.

Au démarrage de l’application la couleur sélectionner par défaut est jaune :



Et à chaque fois qu’on va sélectionner une nouvelle couleur le fond va s’adapter.

Une image contenant texte, capture d’écran, vert, logiciel

Description générée automatiquement

Le body de l’exercice était fourni et il ressemblait à ça :

<body>

    <div id="container">

      <form>

        <p id="titre">Choisir une couleur pour changer le fond du conteneur</p>

        <select

          size="9"

          id="couleurs"

          onChange="ctrl.changerCouleur(couleurs.value);"

        >

          <option value="white">Blanc</option>

          <option value="red">Rouge</option>

          <option value="green">Vert</option>

          <option value="blue">Bleu</option>

          <option selected value="yellow">Jaune</option>

          <option value="#F0F0F0">Gris</option>

          <option value="#0FABCD">Bleu clair</option>

          <option value="#FB33BB">Violet</option>

          <option value="#CDB0DC">Violet Clair</option>

        </select>

      </form>

    </div>

  </body>

On peut voir ma liste déroulante et les noms des couleurs de chaque élément de cette liste.

Dans mon code JavaScript, je suis partie sur un modèle avec classe pour mon contrôleur. Je me suis ensuite occupé de la fonction **« changerCouleur »,** je lui passe en paramètre la couleur qui est sélectionné dans la liste et je change la couleur de fond selon cette couleur. Je fais attention si les couleurs sont du rouge, du bleu ou du vert, le texte sera écrit en blanc, pour ça j’ai juste fait une condition et je change la couleur de ma police :

$().ready(function () {

  ctrl = new Ctrl();

  let couleur = $("#couleurs").val();

  console.log(couleur);

  ctrl.changerCouleur(couleur);

});

class Ctrl {

  constructor() {}

  changerCouleur(couleur) {

    $("#container").fadeOut(1000, function () {

      $("#container").css("background-color", couleur);

      if (couleur === "red" || couleur === "blue" || couleur === "green") {

        $("#container").css("color", "white");

      } else {

        $("#container").css("color", "black");

      }

      $("#container").slideToggle(1000);

    });

  }

}

## Exercice 16

**[EXERCICE NON FINI]**

Dans cet exercice, nous avions une interface graphique qui effectuait plusieurs actions via Jquery.

Voici à quoi ressemble l’interface :

Une image contenant texte, Police, nombre, ligne

Description générée automatiquement

Pour la première partie, il fallait simplement cacher ou afficher du texte. Pour réaliser cela, j’ai utilisé mes boutons et quand je clique dessus, cela va cacher ou afficher le texte. Pour le dernier bouton j’ai utilisé l’option **« toggle »** qui me permet d’afficher et cacher le texte quand j’appuie deux fois dessus par exemple.

$("#btnShow").click(function () {

    $("#div\_1").show();

  });

  $("#btnHide").click(function () {

    $("#div\_1").hide();

  });

  $("#btnToggle").click(function () {

    $("#div\_1").toggle();

  });

J’ai utilisé la même méthode pour **« plier »** et **« déplier »** mon texte en utilisant **« slideUp »** et **« slideDown »**

//plier et déplier

  $("#btnPlier").click(function () {

    $("#div\_2").slideUp();

  });

  $("#btnDeplier").click(function () {

    $("#div\_2").slideDown();

  });

Et finalement, avec le fondu, j’ai utilisé **« fadeIn »** et **« fadeOut »**

//fondu

  $("#btnFonduShow").click(function () {

    $("#div\_3").fadeIn();

  });

  $("#btnFonduHide").click(function () {

    $("#div\_3").fadeOut();

  });

Pour la deuxième partie, j’ai dû rajouter des classes à des éléments et en enlever. Pour ajouter une classe CSS j’ai utilisé **« addClass ».** Pour ne prendre que le premier élément d’une liste, je me suis servi de son index **« li :eq(0) »** et pour le dernier élément, j’ai utilisé **« last() ».**

  //"Ajouter une classe au premier élément": ajoute la classe CSS boldBlueText au premier élément de la liste.

  $("#btnSetClass").click(function () {

    $("li:eq(0)").addClass("boldBlueText");

  });

  //"Supprimer une classe au premier élément": supprimer la classe CSS boldBlueText du premier élément de la liste.

  $("#btnRemoveClass").click(function () {

    $("li:eq(0)").removeClass("boldBlueText");

  });

  //"Ajouter/Supprimer une classe au dernier élément": ajoute la classe CSS boldBlueText au dernier élément de la liste

  //si elle n'y est pas et supprime cette même classe CSS si elle est présente. Vous ne devez pas utiliser l'instruciton if.

  $("#btnToggleClass").click(function () {

    $("li").last().toggleClass("boldBlueText");

  });

Pour le dernier bouton, cela c’est compliquer un peu, j’ai du sélectionné tous les éléments qui commençaient par un chiffre et pas un point et en plus seulement ceux qui étaient impair.

J’ai d’abord défini que ma balise **« li »** devait être dans un élément **« ol »** car les sous éléments de ma liste sont dans un **« ul »** et en serons donc pas pris en compte et il faut préciser que c’est tous les éléments impair grâce au mot **« even »** qui veut dire impair en anglais.$

//"Souligner les éléments impaires": ajoute ou supprime la classe CSS underlineClass au éléments impaires de la liste.

  //Seul les éléments de la liste numérotée doivent être pris en considération, pas ceux de la liste avec un point devant.

  $("#btnPairesClass").click(function () {

    $("ol> li").even().toggleClass("underlineClass"); //ol>li = tout les li dans le ol

  });

## Exercice 17

Web service

REST architecture générale

SOAP

Comment faire un GET, avec quel outil et détaillez un exemple de GET sur Postman

en REST, comment faire un POST, avec quel outil et détaillez un exemple de POST sur Postman

Groupe 5 : en REST, comment faire un PUT, avec quel outil et détaillez un exemple de PUT sur Postman

en REST, comment faire un DELETE, avec quel outil et détaillez un exemple sur Postman

## Exercice 18

Dans cet exercice, nous allons voir qu’on peut aussi créer des Webservices en php. En partant de la base de l’exercice 1, on va tester tout ça.

Je charge les deux fichiers php que je vais utiliser en ligne. Le premier est un webservice de type POST retournant du html.

<?PHP

// convert\_temp\_p\_xml.php

// WebService Conversion de température POST et réponse XML

// Naël Telfser

//

header('Content-Type: application/xml');

$temp = intval($\_POST['Temperature'], 10);

$from = $\_POST['FromUnit'];

$to = $\_POST['ToUnit'];

if ($from === 'C') {

    $tempf = $temp \* 9 / 5 + 32;

} else if ($from === 'F') {

    $tempf = ($temp - 32) \* 5 / 9;

}

$result = "<temperature>" . $tempf . "</temperature>";

echo $result;

?>

Et le deuxième est de type GET et retourne du JSON, normalement on privilégierait cette deuxième méthode. Premièrement parce que comme l'appel du web service ne modifie pas la base de données et que l'on peut l'appeler aussi souvent que l'on veut, on doit utiliser GET et la réponse JSON (JavaScript Object Notation) est plus simple à traiter en JavaScript.

<?php

    // convert\_temp\_g\_json.php

    // WebService Conversion de température GET et réponse JSON

    //

    header('Content-Type: application/json');

    $temp = intval($\_GET['Temperature'], 10);

    $from = $\_GET['FromUnit'];

    $to = $\_GET['ToUnit'];

    if($from === 'C'){

        $tempf = $temp\*9/5+32;

    } else if($from === 'F') {

        $tempf = ($temp-32)\*5/9;

    }

    $result = array('temperature' => $tempf);

    echo json\_encode($result);  // fonction php pour transformer des données au format JSON

?>

On va ensuite changer la base de l’exercice 1 pour que l’interface corresponde à la maquette ci-dessous :

Une image contenant texte, Rectangle, Police, ligne

Description générée automatiquement

Voici à quoi ressemble le code du nouveau body :

<body onload="initCtrl()">

    <div id="container">

      <h3>Convertisseur de température</h3>

      <div class="form-group">

        <label for="celsius1">°C:</label>

        <input

          type="text"

          size="20"

          id="celsius1"

          name="celsius1"

          placeholder="une température °C"

          autofocus

          onkeyup="ctrl.celsius2Fahrenheit();"

        />

        <label for="fahrenheit1">°F:</label>

        <input

          type="text"

          size="20"

          id="fahrenheit1"

          name="fahrenheit1"

          disabled="disabled"

        />

      </div>

    </div>

  </body>

On va pouvoir rencontrer une erreur qui va nous parler de **« cors »** qui veut dire Cross-origin Ressource Sharing. Cette erreur nous indique que certaines requêtes qui proviennent de ources « non sûres » peuvent être bloquées.

Dans noter cas ça va un peut être problématique pour le fonctionnement de noter application. Pour remédier à cela, il y a deux solutions.

Premièrement dans l’entête de noter fichier PHP on peut rajouter la ligne :

header('Access-Control-Allow-Origin: \*');

Qui va donc contrer cette norme. Il existe aussi une extension chrome du nom de **« CORS »** mais on va plutôt privilégier l’entête dans le fichier.

J’ai réalisé deux fichier JavaScript qui vont me permettre d’utiliser mes deux fichier php. Un qui va s’appeler httpServ qui va se charger d’effectuer les requêtes « ajax »

/\*

  But : js de l'exercice 18

  Auteur : Burgy Malori

  Date :   05.06.23 / V1.0

\*/

class HttpServ {

  constructor() {}

  celcius2Fahrenheit(degres, successCallback) {

    let url =

      "https://307.burgym.emf-informatique.ch/Exercices/Exercice\_18/php/POST.php";

    let param = "Temperature=" + degres + "&FromUnit=C&ToUnit=F";

    // envoi de la requête

    $.ajax(url, {

      type: "POST",

      contentType: "application/x-www-form-urlencoded; charset=UTF-8",

      data: param,

      success: successCallback,

      error: this.centraliserErreurHttp,

    });

  }

  // Ajouter ceci, ôter le errorCallback dans le ajax et son paramètre.

  // Dans l'indexCtrl, appellez cette méthode avec une fonction callback

  centraliserErreurHttp(httpErrorCallbackFn) {

    $.ajaxSetup({

      error: function (xhr, exception) {

        let msg;

        if (xhr.status === 0) {

          msg = "Pas d'accès à la ressource serveur demandée !";

        } else if (xhr.status === 404) {

          msg = "Page demandée non trouvée [404] !";

        } else if (xhr.status === 500) {

          msg = "Erreur interne sur le serveur [500] !";

        } else if (exception === "parsererror") {

          msg = "Erreur de parcours dans le JSON !";

        } else if (exception === "timeout") {

          msg = "Erreur de délai dépassé [Time out] !";

        } else if (exception === "abort") {

          msg = "Requête Ajax stoppée !";

        } else {

          msg = "Erreur inconnue : \n" + xhr.responseText;

        }

        httpErrorCallbackFn(msg);

      },

    });

  }

}

Et un autre qui sert juste à faire les conversions degrés en fahrenheit.

$().ready(function () {

  ctrl = new Ctrl();

  httpServ = new HttpServ();

});

class Ctrl {

  constructor() {

  }

  celsius2Fahrenheit() {

    // Lire les degrés du formulaire

    let degres = parseFloat($("#celsius1").val());

    httpServ.celcius2Fahrenheit(degres, this.OKCelsius2Fahrenheit, this.KOCelsius2Fahrenheit);

  }

  KOCelsius2Fahrenheit(xhr) {

    let erreur = xhr.status + ': ' + xhr.statusText

    alert('Erreur - ' + erreur);

  }

  OKCelsius2Fahrenheit(data) {

    let temp = $(data).find("temperature").text();

    let affiche = temp !== "NaN" ? temp : "";

    $("#fahrenheit1").val(affiche);

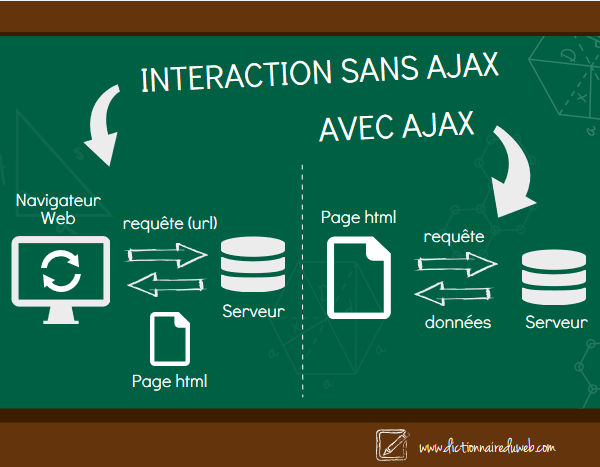
  }

}

## Exercice 20

Dans cet exercice, nous devions réparer une application qui était déjà presque fonctionnelle.

On a utilisé ici **« ajax »** qui nous permet de directement interagir entre une page web et un serveur, sans nécessairement passer par un navigateur.



Voici la syntaxe d’une fonction ajax :

$(document).ready(function(){

$.ajax({

//L'URL de la requête

url: "une/url/au/choix",

//La méthode d'envoi (type de requête)

method: "GET",

//Le format de réponse attendu

dataType : "json",

})

On peut très bien aussi faire des requêtes de type **« POST »** et c’est ce qui est fait dans l’exercice.

Elle prend une multitude de paramètres, mais les plus importants sont :

* **url** : URL de la requête. Seule option strictement obligatoire ;
* **method**(valeur par défaut : GET) : Permet de préciser la méthode d’envoi de la requête (GET, POST ou plus rarement PUT, DELETE, etc.) .
* **data :** Contient les données à envoyer au serveur. Si ces données ne sont pas au format chaine de caractères, elles seront converties en chaine .

Voici ls changements que j’ai fait dans le code :

Dans htppServicejs j’ai changé le lien vers mon fichier php :

// Uploade votre propre fichier PHP et adaptez l'URL ci-dessous.

    let url = "https://307.burgym.emf-informatique.ch/307/Exercices/Exercice\_20/php/login20.php";

Dans indexctrl j’ai rempli la fonction **« loadCompte »** pour pouvoir changer de vue , ici on peut voir que j’appelle la fonction **« chargerVue »** qui si on v avoir, va logiquement charger du contenu sur ma page de base:

loadCompte() {

    this.vue.chargerVue("compte", function(){

      new CompteCtrl();

    })

  }

Dans httpService, il manquait l’information du domaine. On doit préciser **« identifiant.domaine »** pour qu’on puisse recevoir l’information de celui-ci sous forme lisible. Si on ne met pas cela, on risque de se retrouver avec quelque chose comme **« [object] »**

let param =

      "username=" +

      identifiant.username +

      "&password=" +

      identifiant.password +

      "&domaine=" +

      identifiant.domaine +

      "&mail=" +

      identifiant.mail +

      "&langue=" +

      identifiant.langue;

# Projet

## Introduction du projet

### Définition: but, objectifs et fonctionnement désiré de votre projet

Dans ce projet, j’aimerais pouvoir créer une application web qui propose des menus de restaurants, il sera possible de choisir des plats selon des catégories et d’ensuite trouver via une map où se trouve ce restaurant.

### Explications API

L’api ci-dessous, fournit toutes informations des menus et des boissons proposées dans ce restaurant fictif :

[igdev116/free-food-menus-api: Free API 🥪 (github.com)](https://github.com/igdev116/free-food-menus-api)

LA deuxième API me permet d’intégrer une carte a mon application et de placer des points sur celle-ci :

https://www.openstreetmap.org/

### Site de référence

[igdev116/free-food-menus-api: Free API 🥪 (github.com)](https://github.com/igdev116/free-food-menus-api)

https://www.openstreetmap.org/

### Fonctionnement: requête, réponse, format

Pour faire une requête avec cette api je dois utiliser la syntaxe suivante :

https://free-food-menus-api-production.up.railway.app/<params>

Les paramètres sont a remplacé avec ce que je veux rechercher par exemple « drinks » pour la liste de toutes les boissons.

On peut voir ci-dessous que le format des réponses est en json :

Une image contenant texte, ligne, nombre, logiciel

Description générée automatiquement

### Exemple d’utilisation

Ici je cherche dans mon api la liste de toutes les boissons disponibles :

https://free-food-menus-api-production.up.railway.app/Drinks

Une image contenant texte, ligne, nombre, logiciel

Description générée automatiquement

On peut donc voir que cette requête me retourne plusieurs choses :

* **L’id** de la boissson pour l’identifier
* **L’image** de cette boisson
* **Le nom**
* **Le « dsc »** qui est la desription de cette boisson
* **Le prix** de la boisson
* **Le « rate »** qui correspond à la note de ce plat
* **Le « country »** le pays où se trouve le restaurant

## Analyse

### Diagramme de use cases / Explications des cas

Voici mon diagramme « use cases », donc l’utilisateur arrive sur le site, les données donc les plats sont triés par type, boissons, burgers, desserts etc…L’utilisateur va pouvoir choisir un burger par exemple, et la localisation du restaurant d’où vient cette recette va s’afficher sur une nouvelle page.

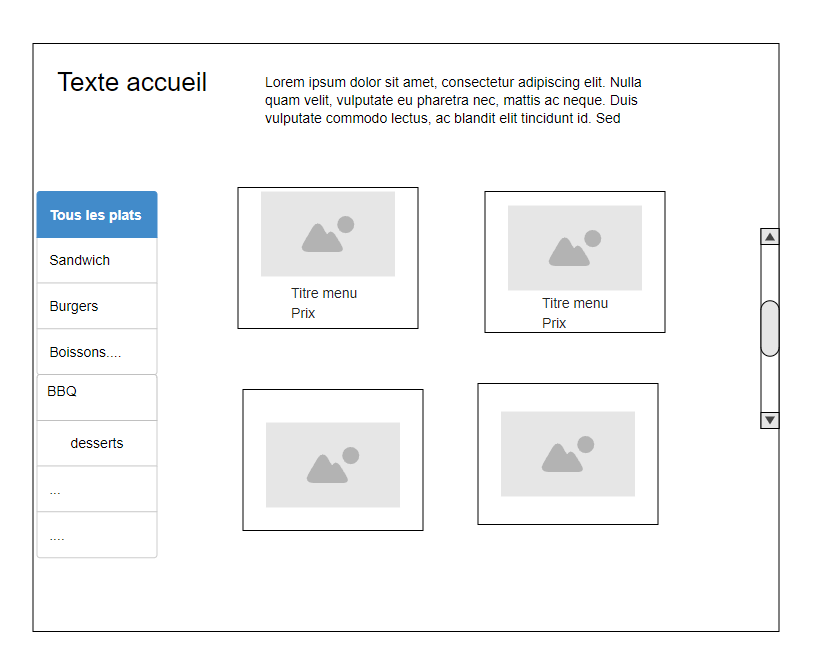
Une image contenant texte, diagramme, ligne, capture d’écran

Description générée automatiquement

### Maquette

Pour ma maquette, je veux rester dans quelque chose d’assez sobre, Une interface claire, les plats apparaitront dans des sortes de petites bulles, alignées, quand on passe la souris dessus on peut aussi voir le nombre d’étoiles que ce plat a. Pour la page de la map, quelque chose de simple aussi, on affiche la map avec le restaurant qui correspond l’adresse et on affiche évidemment son adresse. En dessous de la map sera aussi afficher les autres plats contenus dans la base de données de ce restaurant.

Voici la page d’accueil avec les « filtres » selon les types de plats :



Quand on passe la souris sur une photo pendant un certain temps on va pouvoir voir les notes de ce plat :

Une image contenant texte, capture d’écran, diagramme, conception

Description générée automatiquement

Et la page de la map qui va afficher sur la carte l’emplacement du restaurant ainsi que les informations sur l’adresse le nom et le pays. En dessous on va pouvoir retrouver les autres plats de ce même restaurant qui sont dans la base de données

Une image contenant texte, diagramme, capture d’écran, Rectangle

Description générée automatiquement

## Conception

### Diagramme de navigation

## Implémentation

### HTML / CSS (buts des fichiers, éventuellement des extraits de code avec explications)

### Javascript général (buts des fichiers, extraits de code avec explications)

### Javascript spécifiques (buts des fichiers, extraits de code avec explications)

### Descente de code complète pour une action qui fait appel à un service Web

# Conclusion