307 -

Rapport personnel

Date de création : 15.05.2023  
Version 1 du 12.06.2023

Brodard Clara

|  |  |  |
| --- | --- | --- |
|  |  | Module du 15.05.2023 au 13.06.2023 |

Table des matières

1. Introduction 4

1.1. Objectifs 4

2. Exercices 4

2.1. Exercice1 4

2.1.1. Consigne 4

2.1.2. Exécution 5

2.1.3. Documentation / ce que je retiens 6

2.2. Exercice 2 6

2.2.1. Consigne 6

2.2.2. Exécution 7

2.2.3. Documentation / Ce que je retiens 8

2.3. Exercice 3 9

2.3.1. Consigne 9

2.3.2. Exécution 9

2.3.3. Documentation / ce que je retiens 11

2.4. Exercice 4 11

2.4.1. Consigne 11

2.4.2. Réalisation 13

2.4.3. Documentation / ce que je retiens 14

2.5. Exercice 5 14

2.5.1. Consigne 14

2.5.2. Réalisation 14

2.5.3. Documentation / ce que je retiens 14

2.6. Exercice 6 14

2.6.1. Consigne 14

2.6.2. Réalisation 15

2.6.3. Documentation / ce que je retiens 16

2.7. Exercice 7 16

2.7.1. Consigne 16

2.7.2. Réalisation 17

2.7.3. Documentation / ce que je retiens 17

2.8. Exercice 8 17

2.8.1. Consigne 17

2.8.2. Réalisation 18

2.8.3. Documentation / ce que je retiens 19

2.9. Exercice 9 – demo sur les objets 19

2.10. Exercice 10 – programmation orienté objets 19

2.11. Exercice 11 – programmation orienté classe 19

2.12. Exercice 12 – fonctions et IIFE 19

2.13. Exercice 13 – les cookies 20

2.14. Exercice 14 – Bases de jquery 20

2.15. Exercice 15 – Jquery début 20

2.16. Exercice 16 – utilistation de jquery 20

3. WebServices découverte (ex17) 20

3.1. WebService REST 20

3.2. WebService SOAP 20

3.3. GET 21

3.4. POST 21

3.5. PUT 21

3.6. DELETE 21

4. Ex20 – login etc 21

4.1.1. Consigne 21

4.1.2. 21

5. PROJET 21

5.1. But 21

5.2. Explications API 22

5.2.1. Site de référence 22

5.2.2. Fonctionnement : requête, réponse, format 22

5.2.3. Exemple d’utilisation 29

5.3. Analyse 29

5.3.1. Diagramme de use cases 29

5.3.2. Maquettes 30

5.4. Conception 31

5.4.1. Diagramme de navigation 31

5.5. Implémentation 32

# Introduction

Ce module est une introduction à javascript, et à l’utilisation de Jquery et des API dans la programmation web. Le module est séparer en deux partie, une première avec des exercices divers pour apprendre à maitriser la matière et une deuxième partie de projet.

## Objectifs

* Développer le caractère fonctionnel des pages Web interactives conformément aux données du problème.
* Développer une maquette pour la saisie et la présentation des données compte tenu des aspects ergonomiques.
* Choisir les éléments de formulaire appropriés pour la réalisation des données du problème, et garantir la validation des données entrées.
* Programmer l’application de manière modulaire et conformément aux directives de codification.
* Définir et en mettre œuvre des cas de tests appropriés pour des pages Web interactives et documenter dans le procès-verbal de tests.

# Exercices

Dans ce chapitre, les différents exercices effectuer en cours.

## Exercice1

### Consigne

Utilisation d'un modèle

Un dossier modèle (un template) est disponible ci-dessous et sera utile pour les premiers exercices. Il faudra compléter les entêtes à chaque exercice. Vous pouvez adapter ce template comme bon vous semble.

Travail à réaliser

Pour ce premier exercice, veuillez créer ou copier, avec votre outil de développement, les sources contenues dans le fichier *« exercice\_1.zip »* ci-dessous dans le dossier principal *« /exercices/exercice\_1 »*.

Adaptez le texte (votre nom et prénom) et sa mise en forme (gras) ainsi que la couleur du bouton en modifiant à bon escient les 3 fichiers à disposition. La ligne du bas s'affiche lors de l'appui du bouton.

Maquette de la vue

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Code JS lancé à la fin du chargement de la page

|  |
| --- |
| **<body onload="initCtrl()">** |

Si vous avez besoin d'initialiser certaines choses dans le JavaScript, mais que cela n'est possible que si la page est chargée (DOM créé entièrement), il est possible de lancer une fonction en fin de chargement. On peut aussi déplacer le lancement du script en fin de body au lieu du head. Observez la méthode **initCtrl()** qui ajoute un écouteur par programmation.

HTML avec écouteur

Les lignes en commentaire sont identiques mais l’écouteur est en dur dans le code HTML, mais c’est à vous de l’ajouter ! Cette situation est très courante, mais il faut favoriser l’autre solution. Jouez avec les commentaires pour tester l’une ou l’autre solution.

Hébergement

Adaptez les en-têtes de vos différents fichiers et uploader votre exercice 1 sur votre domaine personnel de l'école.

### Exécution

Le body contient un "onload=initCtrl()”. Cela va permettre d’effectuer le code JavaScript de cette fonction lors du chargement de la page.

<body onload="initCtrl()">

  <div id="container">

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

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

    <p id="info">&nbsp;</p>

  </div>

</body>

La fonction initCtrl() va ajouter aux éléments qui ont l’id « testez » un écouteur d’événement. Il se fera sur un click, et exécutera la fonction testez(). La fonction testez() va modifier les éléments ayant l’id « info ». Il va modifier le texte HTML.

function initCtrl() {

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

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

}

function testez() {

   document.getElementById("info").innerHTML =

      "C'est <b>Clara Brodard</b> qui a pressé le bouton !";

}

La deuxième manière d’exécuter ce code est :

<body>

    <div id="container">

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

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

      <p id="info">&nbsp;</p>

    </div>

  </body>

Le “onclick” sur le bouton est un écouteur. Il n’y aura donc pas besoin de la fonction initCtrl() et va directement faire la fonction « testez() »

### Documentation / ce que je retiens

Pour ajouter un écouteur, on utilise une fonction avec le code suivant :

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

Document va rechercher dans le document où on fait appel à la méthode. GetElementById va rechercher les éléments avec l’id sélectionné. addEventListener va jouter l’écouteur. On ajoute « click » et la méthode à exécuter lors du déclenchement de l’écouteur.

Pour ajouter directement l’écouteur sur le bouton, on ajoute onclick=la méthode à exécuter.

Pour exécuter un code javascript lorsqu’une page à fini de charger, on met sur le body « onload= la fonction à exécuter ».

DOM (Document Object Model) est un model de la construction de la page. C’est une structure en arbre, qui va définir l’étendu d’action du code JS sur le HTML.

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

Description générée automatiquement

## Exercice 2

### Consigne

On vous demande de compléter le conteneur pour afficher un composant formulaire (form) HTML5 pour simuler une opération de login. Pour le moment, le test du login se fera localement en JavaScript *(***valides : *« admin »* quelle que soit la casse pour l’utilisateur et *« emf123 »* pour le mot de passe**).

Le but sous-jacent de cet exercice est de bien séparer les différents fichiers de la vue d’une application entre son **contenu** en HTML (des composants dans une page web), la **présentation** en CSS (le design) et la **logique** métier en JavaScript.

Maquette

Une image contenant texte, logiciel, capture d’écran

Description générée automatiquement

Si on introduit le nom d’utilisateur et le mot de passe correctement :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Si la validation n’est pas correcte (nom d’utilisateur ou mot de passe) :

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

La fonction Javascript alert() ouvre un popup d'information, confirm() ouvre un popup de confirmation ou non et prompt() ouvre un popup d'input, qui permet à l'utilisateur de saisir du texte.

### Exécution

HTML

<body>

   <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>

On a l’exécution d’un code JS avec l’input button onclick= « validerUtilisateur() ». Ici, le bouton doit récupérer les données des inputs pour vérifier qu’ils sont corrects.

CSS

.user-form .field{

  padding: 1em;

  background-color: lightgrey;

}

.user-form .field label{

  display: inline-block;

  width:10em;

}

.user-form .field .input {

  display: inline-block;

}

.user-form input[type=button]{

  width: 7em;

  border-radius: 1em;

  background-color: greenyellow;

  margin: 1em;

}

JS

// teste nom utilisateur et mot de passe

function validerUtilisateur() {

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

  if (username.toLowerCase == "admin".toLowerCase && document.getElementById("password").value == "12345emf") {

    alert("validation ok");

    console.log(document.getElementById("username").value+" se connecte")

  } else {

    alert("utilisateur ou mot de passe incorrect");

    console.log("connexion échouée")

  }

}

Lors d’une véritable identification, on ne va pas tester en local, mais du coté backend. Ici, ce qu’il faut retenir, ce sont les alert() qui affiche des popups et console.log() qui permet d’afficher dans la console.

### Documentation / Ce que je retiens

* Placeholder permet d’afficher un message dans l’emplacement input de texte quand celui-ci est vide. Cela permet à l’utilisateur de savoir ce qu’il doit y écrire.
* Autofocus permet que l’utilisateur puisse directement écrire dans l’input après le chargement de la page.
* Alert() permet d’afficher une popup d’information. Confirm() permet d’afficher une popup avec un choix oui / non. Prompt() permet d’afficher une popup avec un input (text).
* Pour récupérer le texte des input password et username, on utilise document.getElementById(«nomid»).value.
* La différence entre button et input type=button est que <button> peut avoir du contenu, alors que <input> ne le peut pas (c'est un élément nul). Bien que le texte du bouton d'une <input> puisse être spécifié, on ne peut pas ajouter de balisage au texte ou insérer une image.
* Pour écrire dans la console du navigateur, on utilise console.log(« message »). C’est l’équivalent d’un shout en java. On va pouvoir mieux débuguer ou vérifier que notre code s’exécute correctement. Pour ouvrir la console de chrome, on fait un clic droit sur la page, puis inspecter. On a ensuite l’onglet console.

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

Description générée automatiquement

* Pour définir une méthode en javascript, on l’annonce avec junction nomDeLaMethode(){ code à exécuter ;}.
* Pour créer une variable, on fait avec let nomVariable = élément. Les variables peuvent avoir des types, qui sont donnés automatiquement. String, Number, Bigint, Boolean, Undefined, Null, Symbol, Object, Array, Date.
* On peut le scipt Javascript avant le /body afin d’éviter les ralentissements. On peut le charger dans le header, mais cela risque de faire des ralentissements, sauf depuis HTML5 qui a introduit l’attribut async comme balise script. Cela permet d’indiquer au navigateur qu’il n’y a pas de changement du DOM durant le chargement.
* Pour convertir un texte en minuscule, on a la fonction .toLowerCase.

## Exercice 3

### Consigne

Le but de l'exercice est :

* D’améliorer un peu la vue en décolorant fortement la bordure du *« fieldset »* (lightgray) et en centrant et stylisant le bouton ;
* De faire la validation des informations de login par un petit script en PHP (au lieu du JavaScript) sur un serveur Apache à installer sur votre hébergement EMF. Il est aussi présent sur [**http://galleyp.emf-informatique.ch**](http://galleyp.emf-informatique.ch)*.*

Concrètement, vous allez répondre aux questions suivantes en testant chaque nouvelle partie :

1. Comment centrer le bouton de validation et le styliser ?
2. Comment envoyer le formulaire à un script PHP qui fera la validation ?

### Exécution

On ajouter dans le form action et méthode, qui permettent de faire le lien avec le fichier PHP. On a remplacé le bouton par un submit, qui lui, contrairement au type bouton, annonce l’échange avec un fichier REST.

<body>

   <form id="container" class="user-form" action="http://brodardc.emf-informatique.ch/exercices307/ex3/php/login.php" method="POST">

      <fieldset>

         <legend>Identification:</legend>

         <div class="field">

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

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

         </div>

         <div class="field">

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

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

         </div>

         <div class="button">

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

         </div>

      </fieldset>

   </form>

</body>

Le php récupère les informations du formulaire et les tests, avant d’annoncer si le test a réussi ou échoué.

<?PHP

/\*

    But :    login avec PHP

    Auteur : brodardc

    Date :   15.05.2023 / V1.0

\*/

// 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>";

  }

}?>

Ensuite, il fallait remplacer POST par GET. On le remplace dans la méthode.

<form id="container" class="user-form" action="http://brodardc.emf-informatique.ch/exercices307/ex3/php/login.php" method="GET">

Et dans le PHP.

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

### Documentation / ce que je retiens

Les types de boutons :

* type="button" : ce type crée un simple bouton cliquable sans fonctionnalité particulière.
* type="submit" : ce type crée un bouton qui envoie un formulaire vers un serveur lorsqu'il est cliqué. Le visiteur sera conduit à la page indiquée dans l'attribut ***action***.
* type="reset" : ce type crée un bouton qui réinitialise les valeurs d'un formulaire à leurs valeurs par défaut.
* type="image" : ce type crée un bouton qui est représenté par une image spécifiée dans l'attribut src.
* type="checkbox" : ce type crée un bouton qui peut être coché ou décoché.
* type="radio" : ce type crée un bouton qui peut être sélectionné dans un groupe de boutons liés.
* type="file" : ce type crée un bouton qui permet à l'utilisateur de sélectionner un fichier sur son ordinateur.

Un code php commence avec <?PHP et fini avec ?>.

On récupère les informations avec les flux GET ou POST avec le nom des balises, et en la mettant dans une variable.

Pour savoir si une variable est vide, on utilise isset. Pour mettre un string en minuscule, on utilise strtolower. Pour renvoyer quelque chose vers le client, on utilise la commande Echo.

On trouve du javascript dans ce PHP (<script>).

## Exercice 4

### Consigne

Le site <http://jsfiddle.net/> propose un outil très simple pour tester du JavaScript à partir de code HTML + CSS. Le tout est appelé un *« fiddle »* que l’on peut traduire en français par une *« combine »* (ou une astuce). Je vous conseille donc :

* De créer un compte gratuit chez eux (pour pouvoir sauver ou partager des fiddle)
* De tester un premier fiddle avec un affichage du fameux *« hello world »* dans un popup JavaScript (ce que vous savez déjà faire) :

Une image contenant logiciel, texte, Logiciel multimédia, Icône d’ordinateur

Description générée automatiquement

Vous pouvez noter le codage très souple, puisqu’aucune balise HTML n’a précisé que nous utilisions du JavaScript.

L'avantage de cette solution est qu'il est possible de tester du code HTML, CSS ou Javascript rapidement sans devoir mettre en place des fichiers et ouvrir son IDE.

Créez ensuite une petite application Web. Grâce à de l'HTML et du CSS, créez un carré bleu de 100px de côté. Implémentez ensuite une fonction ***messageAlert*** qui affichera un popup avec le message "Quelqu'un à cliqué sur le carré bleu" lorsqu'on clique sur le carré contenu dans le rendu de la page.

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

Description générée automatiquement

Sauvegardez votre ***fiddle*** dans JSFiddle. Vous devez avoir une session ouverte.

Ajoutez le lien vers votre ***fiddle*** dans la page d'accueil du module 307, pour l'exercice 4. Vous pouvez récupérer l'URL de votre fiddle en allant dans les options de votre compte JSFiddle:

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

Description générée automatiquement

### Réalisation

Première partie avec le test d’affichage dans la console avec du JavaScript.

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

Description générée automatiquement

Deuxième partie avec la combinaison du html, cdd et javaScript. Le site internet est utile pour afficher les 3 codes en même temps, et avoir un retour direct de l’exécution du code.

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

Description générée automatiquement

### Documentation / ce que je retiens

JSFiddle est un outil en ligne pour contrôler et debugger son code. Il est pratique, car on peut directement voir les 3 codes JS, HTML et CSS, ainsi que le rendu final. Cet outil va nous permettre de tester des bouts de codes.

## Exercice 5

### Consigne

Résumez l’exercice avec les concepts clés dans votre RP. Mettez directement le code Javascript que vous aurez produit et le résultat. Il est impératif que votre code contienne des commentaires explicants les différentes instructions utilisées.

### Réalisation

// 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(`${a} + ${b}  = ${a+b}`);

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

console.log(`${a} - ${b}  = ${a-b}`);

console.log(`${a} \* ${b}  = ${a\*b}`);

console.log(`${a} / ${b}  = ${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 `…${…}…`;

console.log(`${a} ${b}`);

// 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(`${a} AND ${b} = ${a && b}`);

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

console.log(`${a} OR ${b} = ${a||b}`);

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

a = new Date();

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

b= new Date();

b.setDate(b.getDate() - 61);

// 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(`a = ${a.toLocaleDateString("ch-fr")}`);

console.log(`b = ${b.toLocaleDateString("ch-fr")}`);

// 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";

let c = true;

let d = new Date();

let e;

console.log(`le type de la variable a est : ${typeof a}`);

console.log(`le type de la variable b est : ${typeof b}`);

console.log(`le type de la variable c est : ${typeof c}`);

console.log(`le type de la variable d est : ${typeof d}`);

console.log(`le type de la variable e est : ${typeof e}`);

### Documentation / ce que je retiens

La meilleure manière de déclarer une variable est avec le mot let.

On utilise le mot const à la place de let et le nom de la constante est en majuscule.

La différence entre let et var est que var laisse beaucoup de liberté, voir trop, avec la possibilité de déclarer plusieurs fois la même variable. Elle est pourtant encore beaucoup utilisé, bien que moins recommander.

Les types primitifs en javascript sont String, Number, Boolean, Null, undefined, Symbol et Bigint.

La commande ci-dessous permet de savoir le type d’une variable.

console.log(`le type de la variable a est : ${typeof a}`)

## Exercice 6

### Consigne

Depuis votre outil de développement, copier l’exercice « exercice\_1 » vers « exercice\_6 ».

Nous avions déjà testé le « IF », testons maintenant le « SWITCH », avec une petite application qui doit afficher le jour de la semaine directement en remplacement du contenu d’une balise HTML existante.

Exemple tableau :

const tab = [12, 45, 56];

  const noms = ["Alice", "Bob", "Clara", "Dave"];

  console.log(noms[0]);  // affichage d'Alice

  console.log(noms.length);  // affichage de 4

  noms.push('Eric');        // ajoute Eric à la fin du tableau

  noms.unshift('AA');     // ajoute AA au début du tableau

  noms.pop();            // efface le dernier élément du tableau

  noms.shift();           // efface le premier élément du tableau

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

    console.log(i + " : " + noms[i]);   // 0 : Alice …

  }

  for (let i in noms) {

    console.log(i + " : " + noms[i]);        // 0 : Alice …

  }

  noms.forEach(function (nom) {

    console.log(nom);  // Alice …

  });

  noms.forEach(nom => console.log(nom));    // Alice …

  noms.forEach((nom, i) => console.log(i + " : " + nom));    // 0 : Alice …

### Réalisation

Afficher le jour de la semaine grâce à un switch :

function afficherJourSemaine(){

  let retour;

  switch (new Date().getDay()) {

    case (0):

      retour = "dimanche";

      break;

    case (1) :

      retour = "lundi";

      break;

    case (2):

      retour = "mardi";

      break;

    case (3):

      retour = "mercredi";

      break;

    case(4):

      retour = "jeudi";

      break;

    case(5):

      retour = "vendredi";

      break;

    case(6):

      retour = "samedi";

      break;

    default:

      break;

  }

  document.getElementById("info").innerHTML =

      "Aujourd'hui, on est "+retour;

}

Affichage jour de la semaine avec tableau :

function jourSemaineTab(){

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

  document.getElementById("info2").innerHTML =

    "Aujourd'hui, on est " + jourSemTab[new Date().getday()] + " (tableau)";

}

### Documentation / ce que je retiens

Pour récupérer le numéro du jour (0=dimache), on fait : new Date().getDay().

Pour insérer du texte dans une balise html depuis le code javascript, on utilise document.getElementById(«idBalise »).innerHTML = « texte ».

Pour créer des tableaux :

* const cars = []; //vide
* cars[0]= "Saab";  
  cars[1]= "Volvo";  
  cars[2]= "BMW"; //ajoute dans la case 2
* const cars = ["Saab", "Volvo", "BMW"]; //creation du tableau avec //directement les valeurs
* Pour ajouter un élément au début du tableau : .unshift(élément)
* Pour ajouter un élément à la fin du tableau : .push(élément)
* Pour supprimer le premier élément du tableau : .shift()
* Pour supprimer le dernier élément du tableau : .pop()

Pour récupérer une donnée d’un tableau, on peut faire un FOR comme en JAVA (voir exemple plus haut). Pour afficher toutes les valeurs d’un tableau, on peut faire un FOR EACH. noms.forEach(nom => console.log(nom));

## Exercice 7

### Consigne

Nous allons tester toutes les sortes de boucles possibles en JavaScript avec une application qui ressemble à la précédente du point de vue de l’insertion d’informations dans du HTML par l’entremise de JavaScript.

### Réalisation

La boucle FOR

function testerFor(){

  document.getElementById("info").innerHTML="for (let i=0, i<5, i++){...}";

  for (let i = 0; i < 5; i++) {

    document.getElementById("info").innerHTML+=" i = "+i;

  }

  document.getElementById("infoFin").innerHTML="A utiliser si on sait que l'on veut itérer x fois (x connu avant de commencer la boucle)"

}

La boucle WHILE

function testerWhile(){

  document.getElementById("info").innerHTML="while (i<5){...}";

  let i=0;

  while (i<5) {

    document.getElementById("info").innerHTML+=" i = "+i;

    i++;

  }

  document.getElementById("infoFin").innerHTML="A utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle";

}

La boucle DOWHILE

function testerDoWhile(){

  document.getElementById("info").innerHTML="do {...} while (i<5)";

  let i=0;

  do {

    document.getElementById("info").innerHTML+=" i = "+i;

    i++;

  } while (i<5);

  document.getElementById("infoFin").innerHTML="A utiliser si on ne sait pas le nombre d'itérations au démarrage de la boucle mais avec un passage obligatoire";

}

### Documentation / ce que je retiens

Les boucles en JavaScript sont semblables aux boucles en Java. La boucle FOR doit être utilisé si l’on connait le nombre d’itération avant le début de la boucle. La boucle WHILE est faite si on ne sait pas le nombre d’itérations avant le démarrage de la boucle. Enfin, DO WHILE est à utiliser lorsque l’on ne sait pas le nombre d’itération au démarrage de la boucle mais avec malgré tout un passage obligatoire.

## Exercice 8

### Consigne

Nous allons tester une imbrication de 2 boucles sur les données d’un tableau de personnes défini par un objet de type JSON (JavaScript Object Notation).

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

Description générée automatiquement

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

Description générée automatiquement

 Il s’agit ici de créer la fonction « parcourirUnTableauJSON() » dont le but est de concaténer dans une variable « txt » tout le contenu d’un tableau JSON.

### Réalisation

function parcourirUnTableauJSON() {

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

  const json = {

    personnes: [

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

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

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

    ]

  };

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

    let personne = json.personnes[i];

    document.getElementById("info").innerHTML+=i+".";

    for (let f in personne) {

      document.getElementById("info").innerHTML+=" "+personne[f];

      }

      document.getElementById("info").innerHTML+="<br/>";

    }

  }

### Documentation / ce que je retiens

Les objets JSON sont très polyvalent, un peu comme un objet en java. JSON est facile à comprendre, indépendante des langages informatiques, peut stocker plusieurs types de données.

Pour créer un objet JSON

const personne = {

  prenom: "Jules",

  nom: "Tartampion",

  age: 25

};

Pour balayer tous les champs :

for (let f in personne) {

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

}

Un tableau de JSON :

const json = {

  personnes: [

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

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

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

  ]

};

Parcourir un tableau de JSON :

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

  let personne = json.personnes[i];

// ...

}

Pour afficher tous les champs de personne dans un tableau de personne :

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

    let personne = json.personnes[i];

    document.getElementById("info").innerHTML+=i+".";

    for (let f in personne) {

      document.getElementById("info").innerHTML+=" "+personne[f];

      }

      document.getElementById("info").innerHTML+="<br/>";

    }

## Exercice 9 – demo sur les objets

### Consigne

Créer différents objets de manières différentes.

### Réalisation

Objet JSON :

function creerDesObjetsJSON() {

  console.log("---------------JSON-------------------------");

  // créer un objet en JSON

  let p1 = {

    prenom: "Jules",

    nom: "Tartampion",

    age: 25,

  };

  console.log(p1);

  // ajouter deux méthodes

  p1.toString = function () {

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

  };

  p1["toStringAge"] = function () {

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

  };

  console.log(p1);

  // let txt = p1.toString() + "<br>" + p1.toStringAge();

  let txt = p1 + "<br>" + p1.toStringAge();

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

}

Créer un objet simple (rare), gère l’objet comme un Hashmap, on peut y stocker ce que l’on veut, la fonction toString permet d’afficher le contenu des objets :

function creerDesObjetsObject() {

  console.log("-----------------Object----------------------");

  // créer un objet avec la fonction Object

  const p1 = new Object();

  p1.prenom = "Juliette";

  p1.nom = "Tartampion";

  p1.age = 23;

  console.log(p1);

  // ajouter deux méthodes

  p1.toString = function () {

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

  };

  p1["toStringAge"] = function () {

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

  };

  console.log(p1);

  //        var txt = p1.toString() + "<br>" + p1.toStringAge();

  let txt = p1 + "<br>" + p1.toStringAge();

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

}

Créer un objet avec une fonction :

function Personne(prenom, nom, age) {

  this.prenom = prenom;

  this.nom = nom;

  this.age = age;

  this.toString = function () {

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

  };

}

function creerDesObjetsFonction() {

  console.log("-----------------objet fonction------------------");

  let p1 = new Personne("JuJu", "Tartampion", 40);

  console.log(p1);

  let p2 = new Personne("Julie", "Tartampion", 32);

  console.log(p2);

  let txt = p1 + "<br>" + p2;

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

}

Créer un objet avec une pseudo-classe (fonction prototype) :

function Professeur(prenom, nom, age) {

  this.prenom = prenom;

  this.nom = nom;

  this.age = age;

}

Professeur.prototype.toString = function () {

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

};

function creerDesObjetsClassePrototype() {

  console.log("-----------------prototype----------------------");

  let p1 = new Professeur("De Ju", "Tartampion", 25);

  console.log(p1);

  let p2 = new Professeur("De Jiette", "Tartampion", 24);

  console.log(p2);

  let txt = p1 + "<br>" + p2;

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

}

Créer un objet avec une classe :

class Eleve {

  constructor(prenom, nom, age) {

    this.prenom = prenom;

    this.nom = nom;

    this.age = age;

  }

  toString() {

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

  }

}

function creerDesObjetsClasse() {

  console.log("-----------------Class----------------------");

  let p1 = new Eleve("Julien", "Tartampion", 18);

  console.log(p1);

  let p2 = new Eleve("Julia", "Tartampion", 22);

  console.log(p2);

  let txt = p1 + "<br>" + p2;

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

}

## Exercice 10 – programmation orienté objets

### Consigne

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 final 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****».*

### Réalisation

Index controleur ;

/\*

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

 \*/

document.onreadystatechange = function () {

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

    \_afficherPersonnes();

  }

};

/\*

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

 \*/

// 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;

}

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

function \_afficherInfosPersonne(p) {

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

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

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

}

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

function \_lireInfosPersonne() {

  let p = null

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

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

}

/\*

 \* 3. METHODES PUBLIQUES NECESSAIRES A LA VUE

 \*/

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

function selectionnerPersonne(i) {

  return \_afficherInfosPersonne(personnes[i]);

}

// appelée depuis la vue pour ajouter une personne

function ajouter() {

  let p = \_lireInfosPersonne();

  ajouterPersonne(p);

  \_afficherPersonnes();

}

// appelée depuis la vue pour supprimer une personne

function supprimer() {

  let p = \_lireInfosPersonne();

  supprimerPersonne(p);

  \_afficherPersonnes();

}

Beans personne

/\*

 \* Beans personne

 \* @author brodardc

 \* @version 1.0 / 22.05.2023

 \*/

function Personne(prenom, nom, age){

  this.prenom = prenom;

  this.nom = nom;

  this.age = age;

};

Personne.prototype.toString = function() {

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

};

Worker

### Documentation / ce que je retiens

/\*

 \* Worker

 \* @author brodardc

 \* @version 1.0 / 22.05.2023

 \*/

//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)

];

// premier tri de la liste de personnes

personnes.sort();

// 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;

}

// 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();

  }

}

// supprimer une personne dans la liste des personnes si trouvée

function supprimerPersonne(p) {

  let idx = \_trouverPersonne(p);

  if (idx>=0) {

    personnes.splice(idx,1);

  }

}

## Exercice 11 – programmation orienté classe

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 12 – fonctions et IIFE

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 13 – les cookies

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 14 – Bases de jquery

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 15 – Jquery début

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 16 – utilistation de jquery

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 18

### Consigne

### Réalisation

### Documentation / ce que je retiens

## Exercice 20

### Consigne

### Réalisation

### Documentation / ce que je retiens

# WebServices découverte (ex17)

## WebService REST

Les webService REST sont des services en ligne avec la structure suivante :

* Client
* Server
* Uniform Resource Identifiers
* http Methods
  + get
  + post
  + put
  + patch
  + delete
* Représentations
  + Json
  + Xml
  + html
* Statelessness (n’enregistre pas les anciennes requêtes)
* Response Code

Les méthodes envoyées sont accessibles par le client et les méthodes doivent respecter l’architecture ci-dessus. C’est le type de webService que l’on va utiliser

## WebService SOAP

SOAP est la seconde architecture de webservice. Elle est un mélange de XML et http. Les messages SOAP sont des messages XML qui encapsulent des données à envoyer entre les applications. Le binding SOAP est comment les messages doivent être encodé pour le transport.

Une requête fonctionne de la manière suivante :

1. Le client envoi une requête SOAP au serveur en http
2. Le serveur reçoit la requête et extrait les informations
3. Le serveur traite la requête
4. La réponse est encapsulée et renvoyé au client
5. Le client reçoit la réponse SOAP, extrait les informations et traite les données reçues

Le WSDL est en gros un schéma XML, qui spécifie ce que doit contenir un message pour le serveur.

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

Description générée automatiquement

## GET

## POST

## PUT

## DELETE

# Ex20 – login etc

### Consigne

Trois pages sont chargées à tour de rôle dans un div de la page principale "index.html" via une fonction JQuery "load". Chaque page a son contrôleur. Le "service.js", c'est l'ancien httpServ avec, en plus, la méthode de chargement des vues.

Deux objets "indexCtrl" et "service" sont globaux donc atteignables depuis tous les fichiers JS. Lors du chargement des vues, on crée le contrôleur associé et on rajoute par programmation les écouteurs nécessaires pour le fonctionnement de la page. La bonne vieille méthode de mettre l'écouteur en dure dans le code html nécessite d'ajouter la référence sur l'objet devant la fonction (écouteur), comme cette référence n'est pas globale, cela génère une erreur et le mal-fonctionnement de l'appli.

Le projet est terminé à 95%, il reste deux grosses lacunes et un bug :

* La transmission des données au web service ne se fait pas.
* La page de création de compte ne se charge pas.
* Les données de création de compte transmises au web service sont incorrectes.

### 

# PROJET

## But

Le but de cette application sera de répertorier les livres de la série Game of Thrones, ainsi que des informations sur les différentes « maisons » de la série. Les maisons sont les différentes familles de pouvoir dans la série de livre.

## Explications API

### Site de référence

<https://anapioficeandfire.com/>

Cette API permet de récupérer :

* les livres : <https://anapioficeandfire.com/api/books>
* les personnages : <https://anapioficeandfire.com/api/characters>
* les maisons : <https://anapioficeandfire.com/api/houses>

Cette API n’a pas besoin d’authentification.

### Fonctionnement : requête, réponse, format

Les seules requêtes acceptées sont les GET. On peut rechercher tous les livres :

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Ou un seul livre :

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Description générée automatiquement

De même avec les personnages et les maisons.

Le format des livres est le suivant :

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

Description générée automatiquement

On peut également inclure un filtre :

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

Description générée automatiquement

Le format des personnages:

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

Description générée automatiquement

Pour filtrer :

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

Description générée automatiquement

Le format des maisons:

Une image contenant texte, capture d’écran, menu, document

Description générée automatiquement

Filtrer:

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

Description générée automatiquement

Les réponses de cette API est en json.

<https://thronesapi.com/>

Cette API répertorie les personnages de games of thrones et les continents. Cette API contient des images, c’est pour cela que je vais l’utiliser pour afficher les personnages sur mon site.

Récupérer une image :

<https://thronesapi.com/assets/images/theon.jpg>

récupérer tous les personnages

**https://thronesapi.com/api/v2/Characters**

Cette api permet aussi de post pour les personnages, mais on ne va pas utiliser cette possibilité.

Schéma personnages :

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

Description générée automatiquement

### Exemple d’utilisation

Rechercher les personnages vivants :

Get https://anapioficeandfire.com/api/characters?isAlive=true

Rechercher toutes les maisons:

Get <https://anapioficeandfire.com/api/houses>

Afficher l’image de theon :

GET https://thronesapi.com/assets/images/theon.jpg

## Analyse

### Diagramme de use cases

Une image contenant diagramme, croquis, texte, dessin

Description générée automatiquement

### Maquettes

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

Description générée automatiquement

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

Description générée automatiquement

## Conception

### Diagramme de navigation

Une image contenant texte, capture d’écran, Rectangle, Post-it

Description générée automatiquement

## Implémentation