Afpa

se former, avancer

Les Formulaires

Manipulation des formulaires

AFPA CENTRE DE POMPEY

Les formulaires

Zone de texte - input et textarea

required — oblige la saisie

value — valeur de la zone de texte

gestion du focus — focus : saisie en cours dans la zone de texte
blur : changement de cible provoque un blur sur l'ancienne zone de texte qui avait le focus

```
// Affichage d'un message contextuel pour la saisie du pseudo
pseudoElt.addEventListener("focus", function () {
    document.getElementById("aidePseudo").textContent = "Entrez votre pseud
});
// Suppression du message contextuel pour la saisie du pseudo
pseudoElt.addEventListener("blur", function (e) {
    document.getElementById("aidePseudo").textContent = "";
});
```

autocomplete="current-password"
indique : « ce champ contient le mot de passe *existant* d'un utilisateur déjà inscrit ».
Le navigateur *peut* proposer d'autocompléter si l'utilisateur a enregistré un mot de passe pour ce site.
autocomplete="new-password"
indique : « ce champ sert à créer un nouveau mot de passe ».
Le navigateur *ne proposera pas* de remplir un ancien mot de passe, et peut proposer d'en générer un nouveau.

Les formulaires

Les cases à cocher et boutons radios

Afpa

change indique lorsque l'utilisateur modifie son choix.

Case à cocher dispose d'une propriété checked

Radio Les attributs name et value

```
// Affichage de la demande de confirmation d'inscription
document.getElementById("confirmation").addEventListener("change", function (e)
{
    console.log("Demande de confirmation : " + e.target.checked);
});
```

Les formulaires

Liste déroulante et Formulaire

select + option

change est déclenché sur les modifications apportées à la liste.

Comme pour les boutons radio, la propriété e.target.value de l'événement change contient la valeur de l'attribut value de la balise option associé au nouveau choix, et non pas le texte affiché dans la liste déroulante.

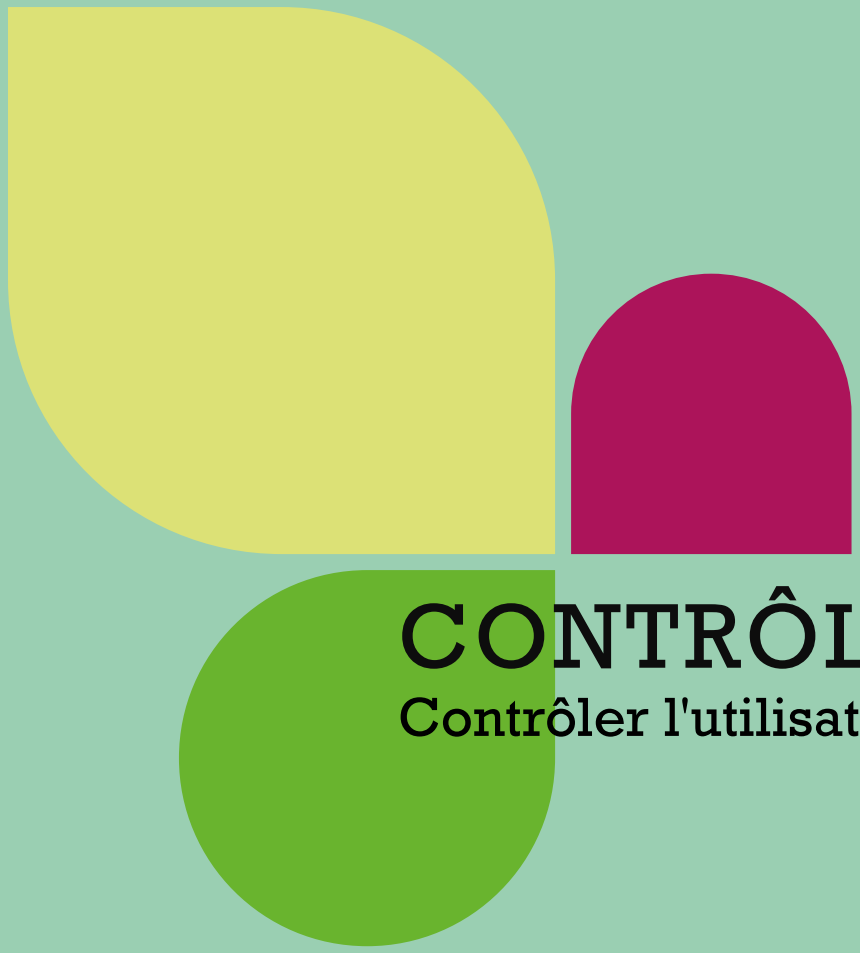
Formulaire

La balise form est l'élément à cibler pour accéder au contenu du formulaire par l'attribut elements rassemblant les champs de saisie

soumission

- input de type submit et input de type reset

C'est à partir de cette soumission que nous allons tester et contrôler la saisie de l'utilisateur et en cas d'erreur utiliser preventDefault



CONTRÔLE DE LA SAISIE

Contrôler l'utilisateur

Sur les éléments du formulaire

REGEX

Vous avez aussi la possibilité de mettre en place des patterns sur la balise input afin de mettre en place un contrôle sur la saisie à la validation.

L'attribut pattern peut être utilisé pour les champs de type text, tel, email, url, password, search.

Les REGEX vont nous aider à mettre en place des contrôles sur la saisie.

```
<p>
    <label for="mdp">Mot de passe</label> :
    <input type="password" name="mdp" id="mdp"
    pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
    title="doit contenir au moins 1 chiffre, 1 majuscule, 1 minuscule
    et au moins 8 caractères" required>
    <span id="aideMdp"></span>
</p>
```

Sur les différentes phases du formulaire

Contrôle des champs

Le contrôle de validité peut se faire de plusieurs manières, éventuellement combinables :

- Soit au fur et à mesure de la saisie d'une donnée
  - input
- Soit à la fin de la saisie d'une donnée
  - blur
- Soit au moment où l'utilisateur soumet le formulaire.
  - submit

Exemple

```
// Vérification de la longueur du mot de passe saisi
document.getElementById("mdp").addEventListener("input", function (e) {
    var mdp = e.target.value; // Valeur saisie dans le champ mdp
    var longueurMdp = "faible";
    var couleurMsg = "red"; // Longueur faible => couleur rouge
    if (mdp.length >= 8) {
        longueurMdp = "suffisante";
        couleurMsg = "green"; // Longueur suffisante => couleur verte
    } else if (mdp.length >= 4) {
        longueurMdp = "moyenne";
        couleurMsg = "orange"; // Longueur moyenne => couleur orange
    }
    var aideMdpElt = document.getElementById("aideMdp");
    aideMdpElt.textContent = "Longueur : " + longueurMdp; // Texte de l'aide
    aideMdpElt.style.color = couleurMsg; // Couleur du texte de l'aide
});

// Contrôle du courriel en fin de saisie
document.getElementById("courriel").addEventListener("blur", function (e) {
    var validiteCourriel = "";
    if (e.target.value.indexOf("@") === -1) {
        // Le courriel saisi ne contient pas le caractère @
        validiteCourriel = "Adresse invalide";
    }
    document.getElementById("aideCourriel").textContent = validiteCourriel;
});
```

Sur les différentes phases du formulaire

Suite

Sur la sortie du champs de saisie, ma fonction contrôle la saisie.

Contrôle du champs Email par exemple

```
// méthode sans REGEX
// Contrôle du courriel en fin de saisie
document.getElementById("courriel").addEventListener("blur", function (e) {
    var validiteCourriel = "";
    if (e.target.value.indexOf("@") === -1) {
        // Le courriel saisi ne contient pas le caractère @
        validiteCourriel = "Adresse invalide";
    }
    document.getElementById("aideCourriel").textContent = validiteCourriel;
});

// méthode avec REGEX
// Contrôle du courriel en fin de saisie
document.getElementById("courriel").addEventListener("blur", function (e) {
    // Correspond à une chaîne de la forme xxx@yyy.zzz
    var regexCourriel = /.+@.+\..+/;
    var validiteCourriel = "";
    if (!regexCourriel.test(e.target.value)) {
        validiteCourriel = "Adresse invalide";
    }
    document.getElementById("aideCourriel").textContent = validiteCourriel;
});
```

Requêtes HTTP
Communiquer avec un serveur

QUELQUES NOTIONS

Quelques notions théoriques qu'il faut connaître avant de se lancer dans la communication avec un serveur en JavaScript.

Service Web : programme s'exécutant sur un serveur accessible depuis internet et fournissant un service.

- Pour ce faire, il met à disposition une API.

API : Application Programming Interface - interface de communication avec les services Web, au travers de requêtes.

Requêtes : données qui respectent le protocole de communication et qui sont envoyées au serveur.

Protocoles : SMTP (envoi de mails), IMAP (réception de mails), HTTP et HTTPS, FTP, WebDAV, etc…

Requêtes HTTP asynchrones : AJAX (Asynchronous JavaScript and XML) est la technologie utilisée pour gérer ce type de requêtes.

- Cela permet de ne pas bloquer le navigateur pendant l'attente d'une réponse du serveur.

JSON : le format de données standard actuel pour l'échange des données sur le WEB.

LE PROTOCOLE HTTP(S)

HTTP : HyperText Transfert Protocol

- Communiquer avec un site internet, chargement des pages HTML, des styles CSS, etc…
- Envoie et récupération d'informations avec les formulaires.

- Méthodes :
  - GET : permet de récupérer des ressources
  - POST : permet de créer ou modifier une ressource
  - PUT : permet de modifier une ressource
  - DELETE : permet de supprimer une ressource
  - HEAD : demande des informations sur la ressource sans obtenir la ressource.
  - TRACE, OPTION, CONNECT…
- URL : l'adresse du service web à atteindre
- Données : les données qu'on envoie mais aussi qu'on reçoit
- Code HTTP : code numérique qui indique comment s'est déroulée la requête
  - 200 : tout s'est bien passé
  - 404 : la ressource n'existe pas
  - 500 : une erreur avec le service web
  - … https://fr.wikipedia.org/wiki/Liste_des_codes_HTTP

GET vs POST

GET

Cette méthode de requête existe depuis le début du Web. Elle est utilisée pour demander une ressource, par exemple un fichier HTML, au serveur Web.

- Paramètres URL

La requête GET peut recevoir des informations supplémentaires que le serveur Web doit traiter.

Ces paramètres d'URL sont simplement ajoutés à l'URL. La syntaxe est très simple :

- La chaîne de requête est introduite par un ? Et chaque paramètre est nommé, et composé donc d'un nom et d'une valeur : Nom=Valeur
- Si plusieurs paramètres doivent être inclus, ils sont reliés par un &
- un encodage des caractères spéciaux est effectué (exemple : @).

POST

Si vous souhaitez envoyer de grandes quantités de données, par exemple des images, ou des données confidentielles de formulaires au serveur, la méthode GET n'est pas idéale car toutes les données que vous envoyez sont écrites ouvertement dans la barre d'adresse du navigateur.

Dans ces cas, la méthode POST est la plus adaptée.

Cette méthode n'écrit pas les paramètres de l'URL, mais les ajoute à l'en-tête HTTP.

- Les requêtes POST sont principalement utilisées pour les formulaires en ligne.

Récupération des données du Formulaire

https://developer.mozilla.org/fr/docs/Web/API/window

Exemple

Depuis un formulaire

Soit le formulaire suivant :


```html
<form id="userForm" action="compte.html" method="get">
    <div class="form-group">
        <label for="nom">Nom :</label>
        <input type="text" id="nom" name="nom" required pattern="[A-Za-zÀ-ÖØ-öø-ÿ\s]+">
        <div class="error" id="nomError"></div>
    </div>
    <div class="form-group">
        <label for="prenom">Prénom :</label>
        <input type="text" id="prenom" name="prenom" required pattern="[A-Za-zÀ-ÖØ-öø-ÿ\s]+">
        <div class="error" id="prenomError"></div>
    </div>
    <div class="form-group">
        <label for="email">Email :</label>
        <input type="email" id="email" name="email" required>
        <div class="error" id="emailError"></div>
    </div>
    <div class="form-group">
        <label for="dateAnniversaire">Date d'anniversaire :</label>
        <input type="date" id="dateAnniversaire" name="dateAnniversaire" required>
        <div class="error" id="dateAnniversaireError"></div>
    </div>
    <div class="form-group">
        <label for="adresse">Adresse :</label>
        <input type="text" id="adresse" name="adresse" required>
        <div class="error" id="adresseError"></div>
    </div>
    <button type="submit">Soumettre</button>
</form>
```

window

Notre fenêtre en cours…

Lors de l'envoi du formulaire par la méthode GET, on envoie au travers de la requête HTTP de manière visible, les informations saisies dans notre formulaire.

Avec window.location, on obtient l'ensemble des propriétés disponibles.

- window représente notre page en cours
- location contient toutes les informations de la page courante

window.location .search

L'attribut search

L'attribut search contient l'ensemble des paramètres envoyés au serveur sous la forme

- ?param=valeur&param=valeur…

Ensuite, on peut aisément récupérer cette donnée dans une variable et en extraire les informations dans un tableau.

```
> let param = window.location.search
< undefined
> console.info(param);
? VM469:1
nom=boebion&prenom=jerome&email=jero%40free.fr&dateAnniversaire=2025-03-10&adresse=centre+afpa+de+Pompey
```

Récupération des données d'une URL

Côté Javascript

Maintenant que nous connaissons les éléments à cibler pour récupérer les données du formulaire, il nous reste à écrire un script Javascript.

Voici un exemple de récupération des données **de manière basique** au travers d'une URL.

- Les tests dans ce code permettent de remplacer les codes ascii que le formulaire utilise lors de l'envoi.
    - *Dans cet exemple, pour l'email par exemple il remplace : par le code ascii correspondant %40 et les espaces par +*

```
function getParamsURL() {

    // objet contenant les parametres
    let varParams = {};
    // recupération du l'url en retirant le ?
    let search = window.location.search.substring(1);
    // contrôle
    console.info(search);

    // decompostion des parametre en retirant le &
    let varSearch = search.split('&');
    // contrôle
    console.info(varSearch);

    // parcours des élements et contruction de mon objet
    for (let i=0; i < varSearch.length; i++) {
        // suppression du =
        let parameter = varSearch[i].split('=');
        // mise en forme du caractère ASCII @
        if (parameter[0] === "email") {
            parameter[1] = parameter[1].replace('%40', '@');
        }
        // mise en forme du caractère ASCII +
        if (parameter[0] === "adresse") {
            parameter[1] = parameter[1].replaceAll('+', ' ');
        }
        // contruction de la réponse
        varParams[parameter[0]] = parameter[1].toUpperCase();

    }
    // controle
    console.info(varParams);

    return varParams;
}
```

Affichage

Mise en forme

Ensuite dans la page de récupération, là où le formulaire nous redirige, nous n'avons plus qu'a afficher les données saisies.

```
document.addEventListener("DOMContentLoaded", function () {

    // appel de la fonction
    let params = getParamsURL();

    console.dir(params);

    // Mise en forme dans la page HTML
    document.getElementById("nom").textContent = params.nom;
    document.getElementById("prenom").textContent = params.prenom;
    document.getElementById("email").textContent = params.email;
    document.getElementById("dateAnniversaire").textContent = params.dateAnniversaire;
    document.getElementById("adresse").textContent = params.adresse;

});
```

FORMDATA

https://developer.mozilla.org/fr/docs/Web/API/FormData

FORMDATA

L'objet FormData a été standardisé et facilite grandement l'envoi vers un serveur.

- Il peut être utilisé indépendamment d'un formulaire, en lui ajoutant une à une les données à transmettre grâce à sa méthode append.
- Cette méthode prend en paramètres le nom et la valeur de la donnée ajoutée (clé/valeur).

Ensuite, on peut retrouver les données contenant dans notre objet FormData avec la méthode entries() qui retourne un itérator permettant d'accéder à l'ensemble des données.

Cet objet est souvent utilisé avec la méthode POST

```
document.getElementById('monFormulaire').addEventListener('submit', function(event) {
    event.preventDefault(); // Empêche l'envoi du formulaire

    // Crée un nouvel objet FormData à partir du formulaire
    const formData = new FormData(event.target);

    // Affiche les données du formulaire dans la console
    for (let [name, value] of formData.entries()) {
        console.log(`${name}: ${value}`);
    }
});
```

https://fr.javascript.info/formdata

MÉTHODES DISPONIBLES

Diverses méthodes sont disponibles avec formdata

https://developer.mozilla.org/fr/docs/Web/API/FormData

- append() : Ajoute une nouvelle valeur à une clé existante dans un objet FormData, ou ajoute la clé si elle n'existe pas encore.
- delete() : Supprime une paire clé/valeur d'un objet FormData.
- entries() : Renvoie un itérateur permettant de passer en revue toutes les paires clé/valeur contenues dans cet objet.
- get() : Renvoie la première valeur associée à une clé donnée à partir d'un objet FormData.
- getAll() : Renvoie un tableau de toutes les valeurs associées à une clé donnée à partir d'un objet FormData.
- has() : Renvoie un booléen indiquant si un objet FormData contient une certaine clé.
- key() : Renvoie un itérateur permettant de parcourir toutes les clés des paires clé/valeur contenues dans cet objet.
- set() : Renvoie un itérateur permettant de parcourir toutes les valeurs contenues dans cet objet.
- values() : Renvoie un itérateur permettant de parcourir toutes les valeurs contenues dans cet objet.

localStorage et session Storage

Au sein du navigateur

LOCALSTORAGE

- fonctionnalités JavaScript qui permet de stocker des données de manière persistante ou temporaire dans le navigateur web.
- Les données stockées dans localStorage restent disponibles même après la fermeture et la réouverture du navigateur.
- Les données stockées dans sessionStorage restent disponibles dans la session ou l'onglet du navigateur.
- utilisé pour stocker des données locales telles que des préférences utilisateur, des données de session ou des informations de configuration.
- accessibles uniquement via JavaScript et spécifiques au domaine et au protocole utilisés pour accéder à la page.

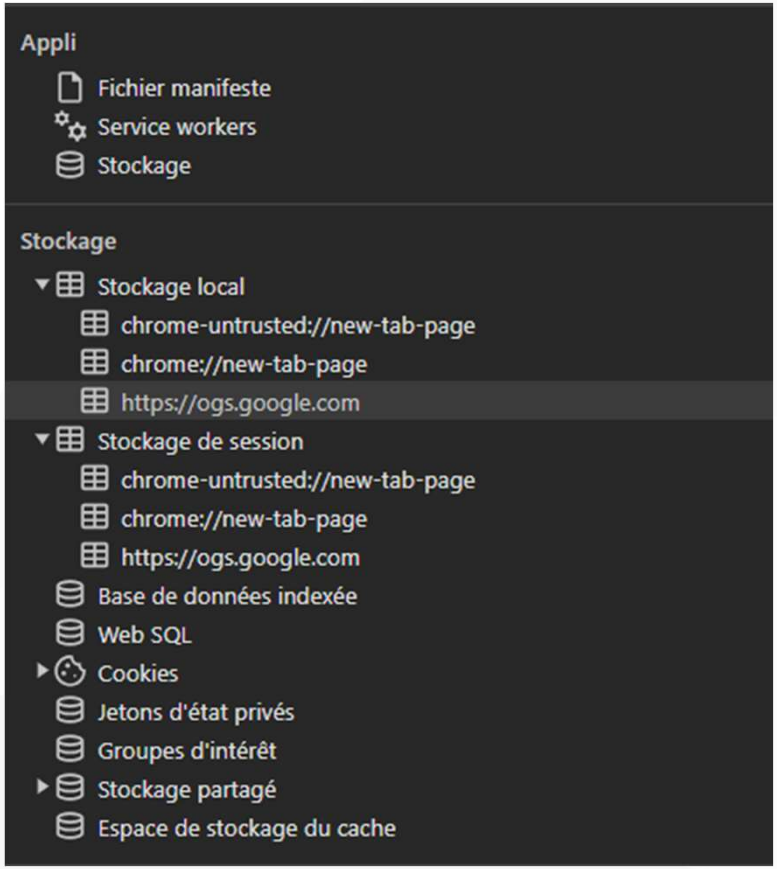
- Attention : En termes de sécurité web, l'utilisation doit être abordée avec prudence, car il peut présenter certains risques si les données sensibles sont mal gérées ou mal protégées.
  - Données sensibles : Ces données pourraient être accessibles à des scripts malveillants ou à d'autres attaques.
  - Injection de code : Ne stockez pas de données directement à partir d'entrées utilisateur sans validation et échappement appropriés. Cela pourrait permettre à des attaquants d'injecter du code malveillant dans les données stockées.
  - Taille limitée : il y a généralement une limite de taille (généralement quelques mégaoctets) par domaine. Stocker de grandes quantités de données peut affecter les performances du site web.
  - Utilisation sécurisée du HTTPS : Assurez-vous que votre site web utilise HTTPS pour chiffrer les données échangées entre le navigateur de l'utilisateur et votre serveur. Cela réduit le risque d'interception des données stockées.
  - Gestion des autorisations : Les navigateurs modernes demandent généralement la permission de l'utilisateur avant de stocker des données. Assurez-vous de respecter les préférences de l'utilisateur et de ne stocker que les données nécessaires.
  - Nettoyage des données : Évitez d'accumuler des données obsolètes ou inutiles Assurez-vous de nettoyer régulièrement les données qui ne sont plus nécessaires pour réduire les risques en cas de compromission.

DANS L'INSPECTEUR

Depuis l'inspecteur de votre navigateur, vous pouvez avoir accès aux différents espaces de stockage.

Depuis la console, vous pouvez également manipuler le localStorage ou la sessionStorage au travers des méthodes disponibles :

- getItem()
- removeItem()
- setItem()
- length()
- clear()



LOCALSTORAGE VS SESSIONSTORAGE

	localStorage	sessionStorage
Type d'objet	Objet JavaScript	
Portée	Domaine du site	Session de navigation
Durée de stockage	Persistant (reste jusqu'à ce qu'il soit explicitement supprimé)	Temporaire (disponible uniquement pendant la durée de la session)
Volume de stockage	Plusieurs Mo (selon le navigateur)	
Accessibilité	Tous les scripts de la même origine (domaine, protocole et port)	
Persistance	Persistant, reste jusqu'à ce qu'il soit explicitement supprimé	Temporaire, disparaît lorsque la session se termine (par exemple, lorsque l'onglet ou le navigateur est fermé)
Utilisation typique	Stockage de données utilisateur, telles que des préférences ou des données de configuration	Stockage de données temporaires nécessaires pendant la session de navigation de l'utilisateur, telles que des informations de session utilisateur ou des états temporaires
Sécurité	Doit être utilisé avec prudence pour éviter les fuites de données sensibles	
Exemple d'utilisation	Stockage des préférences utilisateur	Stockage des informations de session utilisateur

EXEMPLE

Reprenons notre exemple précédent en remplaçant la méthode basique par l'utilisation du formData et de localstorage

- Au niveau du formulaire, sur la soumission, je crée un élément formData à partir de mon formulaire.
- Ensuite, afin de pouvoir le transmettre, je dois effectuer une conversion en objet Javascript et le transformer en JSON (JSON.stringify).
  - *En effet, le localstorage ne peut contenir des objets complexes tel que formData.*

```
document
  .getElementById("userForm")
  .addEventListener("submit", function (event) {
    event.preventDefault();

    let isValid = checkForm();

    if (isValid) {
      // Stocker les informations dans le localStorage

      // creation du formData
      let formData = new FormData(event.target);
      // Affiche les données du formulaire dans la console
      for (let [name, value] of formData.entries()) {
        console.log(`${name}: ${value}`);
      }

      // Convertit les données du formulaire en objet
      // le localstorage ne peux stocker que du texte donc le JSON est parfait
      // Cela implique de transformer l'objet FormData en un objet JavaScript standard,
      // puis de le convertir en chaîne JSON avec JSON.stringify().
      const formDataObject = {};
      formData.forEach((value, key) => {
        formDataObject[key] = value;
      });


      // Sauvegarde les données dans localStorage
      localStorage.setItem("userInfo",JSON.stringify(formDataObject));

      // Rediriger vers la page de compte
      window.location.href = "compte.html";
    }
  });
```

EXEMPLE SUITE

Enfin sur la page de compte, je vais pouvoir récupérer les informations depuis le localstorage et reparcourir les données.

1. Lecture depuis le localstorage en fonction du nom donné et convertion en objet Javascript (JSON.parse)
2. Parcours de l'objet Javascript, pour l'afficher dans les bons éléments HTML.

```
// Récupérer les informations de l'utilisateur depuis le localStorage
// Lorsque vous récupérez les données depuis localStorage,
// vous devez les convertir de JSON en objet JavaScript avec JSON.parse().
const userInfoFormData = JSON.parse(localStorage.getItem("userInfo"));

// contrôle
console.info(userInfoFormData);

// si l'objet est bien présent dans le localstorage
// mise à jour des éléments html
if (userInfoFormData) {

  // parcours du formData
  for(const pair of Object.entries(userInfoFormData)) {

    // contrôle
    console.table(pair);

    // ciblage de l'element correspondant
    const tag = document.getElementById(pair[0]);
    // remplissage de la valeur de l'element
    tag.textContent = pair[1];
  }

} else {
  alert( 'Aucune donnée de formulaire trouvée.');
}
```

EXEMPLE DE CLASSE UTILITAIRE

Voici un exemple de classe Javascript contenant des méthodes d'accès sur localStorage et sessionStorage.

Pour pouvoir utiliser ces méthodes, il suffit de les importer :

```
// Importer les fonctions du module de stockage
import {
    saveTolocalStorage,
    saveToSessionStorage,
    getAllLocalStorageItems,
    getAllSessionStorageItems,
    clearLocalStorage,
    clearSessionStorage
} from './storage-utils.js';
```

```
// Fonctions utilitaires pour la gestion du stockage
export function saveTolocalStorage(key, data) {
    localStorage.setItem(key, JSON.stringify({
        ...data,
        date: new Date().toLocaleString()
    }));
}

export function saveToSessionStorage(key, data) {
    sessionStorage.setItem(key, JSON.stringify({
        ...data,
        date: new Date().toLocaleString()
    }));
}

export function getAllLocalStorageItems() {
    return Object.keys(localStorage)
        .map(key => ({
            key,
            value: JSON.parse(localStorage.getItem(key))
        }));
}

export function getAllSessionStorageItems() {
    return Object.keys(sessionStorage)
        .map(key => ({
            key,
            value: JSON.parse(sessionStorage.getItem(key))
        }));
}

export function clearLocalStorage() {
    localStorage.clear();
}

export function clearSessionStorage() {
    sessionStorage.clear();
}
```

Gestion des erreurs

TRY..CATCH…FINALLY

Peu importe notre niveau en programmation, nos scripts comportent parfois des erreurs.

- Elles peuvent être dues à nos erreurs, à une entrée utilisateur imprévue, à une réponse erronée du serveur et à mille autres raisons.

Généralement, un script “meurt” (s’arrête immédiatement) en cas d’erreur, en l’affichant dans la console.

La structure try..catch est identique à celle vu avec Java.

```
try {
  // code...
} catch (err) {
  // Gestion des erreurs
}
```

En cas d’erreur, JavaScript génère un objet contenant les détails à son sujet. L’objet est ensuite passé en argument à catch

Pour toutes les erreurs intégrées, l’objet d’erreur a deux propriétés principales :

- Name : Nom de l’erreur. Par exemple, pour une variable non définie, il s’agit de "ReferenceError".
- Message : Message textuel sur les détails de l’erreur.

Il existe d’autres propriétés non standard disponibles dans la plupart des environnements. L’un des plus largement utilisés et supportés est :

- Stack : Pile d’exécution en cours : chaîne contenant des informations sur la séquence d’appels imbriqués ayant entraîné l’erreur. Utilisé à des fins de débogage.

CEPENDANT, IL FAUT TENIR COMPTE DES COMPORTEMENTS DE NOTRE CODE, DÛ À L'ASYNCHRONE. LE COMPORTEMENT DE TRY CATCH EST SYNCHRONE.

Si une exception se produit dans le code "planifié", comme dans setTimeout, try...catch ne l'attrapera pas

```
try {
  setTimeout(function() {
    noSuchVariable; // le script mourra ici
  }, 1000);
} catch (err) {
  alert( "won't work" );
}
```

C'est parce que la fonction elle-même est exécutée ultérieurement, lorsque le moteur a déjà quitté la structure try...catch.

Pour capturer une exception dans une fonction planifiée, try...catch doit être à l'intérieur de cette fonction.

```
setTimeout(function() {
  try {
    noSuchVariable; // try...catch gère l'erreur !
  } catch {
    alert( "error is caught here!" );
  }
}, 1000);
```

LEVER NOS PROPRES EXCEPTIONS

L'instruction throw génère une erreur.

La syntaxe est la suivante :

- throw <error object>

Techniquement, on peut utiliser n'importe quoi comme objet d'erreur.

- Cela peut même être une primitive, comme un nombre ou une chaîne, mais il est préférable d'utiliser des objets, de préférence avec les propriétés name et message (pour rester quelque peu compatibles avec les erreurs intégrées).

JavaScript comporte de nombreux constructeurs intégrés pour les erreurs standards : Error, SyntaxError, ReferenceError, TypeError et autres.

Nous pouvons également les utiliser pour créer des objets d'erreur.

```
let error = new Error(message);
// ou
error = new SyntaxError(message);
error = new ReferenceError(message);

error = new Error("Houston, we have a problem o_O");

alert(error.name); // Error
alert(error.message); // Houston, we have a problem o_O

// JSON.parse génére une SynaxError
try {
  JSON.parse("{ bad json o_O }");
} catch (err) {
  alert(err.name); // SyntaxError
  alert(err.message); // Unexpected token b in JSON at position 2
}

let json = '{ "age": 30 }'; // données incomplètes

try {

  let user = JSON.parse(json); // <-- pas d'erreurs

  if (!user.name) {
    throw new SyntaxError("Incomplete data: no name"); // (*)
  }

  alert( user.name );

} catch (err) {
  alert( "JSON Error: " + err.message ); // JSON Error: Incomplete data: no name
}
```

LES ERREURS INTÉGRÉES

En JavaScript, il existe plusieurs types d'erreurs intégrées qui peuvent être levées pendant l'exécution d'un programme.

Voici quelques-unes des erreurs les plus courantes :

1. EvalError : Indique une erreur concernant la fonction globale eval(). Cette erreur n'est plus levée par le moteur JavaScript moderne, mais elle existe toujours pour des raisons de compatibilité.
2. RangeError : Levée lorsqu'une valeur n'appartient pas à l'ensemble ou à la plage de valeurs autorisées. Par exemple, cela peut se produire lorsque vous essayez de créer un tableau avec une taille négative.
3. ReferenceError : Levée lorsqu'on fait référence à une variable qui n'existe pas.
4. SyntaxError : Levée lorsqu'il y a une erreur dans l'analyse du code JavaScript. Cela se produit souvent à cause de fautes de frappe ou de syntaxe incorrecte.
5. TypeError : Levée lorsqu'une valeur n'est pas du type attendu. Par exemple, l'appel d'une méthode sur une valeur null ou undefined.
6. URIError : Levée lorsqu'une fonction globale manipulant des URI est utilisée de manière incorrecte. Par exemple, encodeURI ou decodeURI.
7. AggregateError : Introduite dans ES2021, elle est utilisée pour regrouper plusieurs erreurs en une seule. Elle est souvent utilisée avec les promesses pour capturer plusieurs rejets.
8. InternalError : Levée lorsqu'une erreur interne se produit dans le moteur JavaScript. Ce type d'erreur est rarement vu par les développeurs, car il est généralement lié à des bugs dans le moteur JavaScript lui-même.

PROPAGER UNE EXCEPTION

Dans l'exemple précédent, nous utilisons try...catch pour gérer des données incorrectes.

Mais est-il possible qu'une autre erreur inattendue se produise dans le bloc try {...} ?

- Comme une erreur de programmation (variable is not defined) ou quelque chose d'autre, pas seulement cette "donnée incorrecte".

Catch ne doit traiter que les erreurs qu'il connaît et "renvoyer" toutes les autres.

La technique "rethrowing" peut être expliqué plus en détail comme :

- Catch obtient toutes les erreurs.
- Dans le bloc catch (err) {...} nous analysons l'objet d'erreur err.
- Si nous ne savons pas comment le gérer, nous faisons throw err.

```
let json = '{ "age": 30 }'; // données incomplètes
try {

  let user = JSON.parse(json);

  if (!user.name) {
    throw new SyntaxError("Incomplete data: no name");
  }

  blabla(); // erreur inattendue

  alert( user.name );

} catch (err) {

  if (err instanceof SyntaxError) {
    alert( "JSON Error: " + err.message );
  } else {
    throw err; // propager (*)
  }

}
```

Afpa

Exercice :
créer un formulaire en HTML et un script permettant la récupération des informations qui s'afficheront dans une deuxième page.

[Bonus] : mettre en œuvre un appel à une API (Meteo, Leaflet ou Swapi)