JS Browser BOM (window, screen, location, history, etc.)

Table des matières

I. Contexte

Durée : 1 heure

Prérequis : maîtrise des concepts fondamentaux de JS et de POO

Environnement de travail : Replit

Contexte

Vous connaissez les principes fondamentaux de la programmation en JavaScript. Le concept de programmation orientée objet vous est par ailleurs bien moins étranger. JavaScript permet d'ajouter de l'interactivité et du dynamisme à un site internet. C'est aussi une technologie qui apporte des fonctionnalités efficaces de traitement des données.

Comme vous le savez, le Document Object Model permet d'accéder aux différents éléments du document HTML. Il représente donc une parcelle entre les scripts et les pages du site, en prenant une forme orientée objet.

Dans ce cours nous parlerons d'un point fondamental : JS Browser BOM (pour Browser Object Model). JS Browser BOM est une collection d'objets JavaScript fournis par les navigateurs web. Les objets fournis permettent aux développeurs de manipuler et de contrôler les fenêtres du navigateur, les écrans, les adresses URL et les historiques de navigation des utilisateurs. Dans ce cours, nous parlerons de plusieurs objets de JS Browser BOM.

Nous verrons des exemples concrets d'utilisation, avec des scripts JS que vous pourrez tester via votre interface Replit. Si vous rencontrez des bugs, étant donné que nous allons travailler sur la gestion de fenêtres, n'hésitez pas à créer un projet sur un IDE comme Visual Studio Code. Ce cours sera composé par ailleurs d'exercices de code ainsi que de vidéos d'explication. Pensez aussi à consulter la documentation Mozilla dédiée aux points abordés.

Attention

Pour avoir accès au code et à l'IDE intégré de cette leçon, vous devez :

1) Vous connecter à votre compte sur https://replit.com/ (ou créer gratuitement votre compte)

2) Rejoindre la Team Code Studi du module via ce lien :

https://replit.com/teams/join/kffyqqqiagcezrjugixbetjackpqhpgo-javascript-studi

Une fois ces étapes effectuées, nous vous conseillons de rafraîchir votre navigateur si le code ne s'affiche pas.

En cas de problème, redémarrez votre navigateur et vérifiez que vous avez bien accepté les cookies de connexion nécessaires avant de recommencer la procédure.

Pour accéder au code dans votre cours, cliquez sur le nom du lien Replit dans la fenêtre. Par exemple :

II. Comprendre l'objet window

A. Comprendre l'objet window

Définition **Window**

L'objet **window** est l'objet de niveau supérieur dans le BOM. C'est un objet global. Il représente la fenêtre du navigateur et fournit des fonctionnalités pour contrôler la taille, la position et le contenu de la fenêtre. L'objet **window** contient le fameux objet DOM (Document Object Model). Chaque fenêtre aura donc son DOM accessible via l'objet **window**.

Étant l'objet global par défaut en JS, toutes les propriétés et méthodes de l'objet **window** peuvent être utilisées sans écrire le mot clé **window**. Mais, il est intéressant de l'écrire afin d'éviter les confusions avec d'autres variables ou fonctions (c'est une bonne pratique de code).

Préparation de l'environnement

Pour bien comprendre, nous allons créer dans notre répertoire courant un fichier HTML **index.html** comme celui-ci :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport">
    <title>Document</title>
</head>
<body>
    <p>index.html</p>
    <button id="bouton">button</button>
    <script src="script.js"></script>
</body>
</html>
```

Un fichier **destination.html** :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport">
    <title>Document</title>
</head>
<body>
    <p>destination</p>
    <button id="bouton2">button</button>
</body>
</html>
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-1

[cf.]

Et un fichier **script.js** qui contiendra nos scripts.

Ouvrir / fermer la fenêtre du navigateur web

L'objet **window** permet d'ouvrir une fenêtre spécifique, mais aussi de fermer la fenêtre web actuelle. Pour cela, nous pouvons utiliser les méthodes **`open()`** et **`close()`**.

Voici comment utiliser la méthode `open()` :

```
window.open(/*url*/);
```

Cette ligne de code permet d'ouvrir le lien passé comme argument dans une fenêtre. Cependant, en règle générale, si une fenêtre est ouverte par défaut avec cette méthode, l'ouverture de la fenêtre sera bloquée par le navigateur (pop-up bloqué). Pour résoudre ce problème, nous pouvons par exemple déclencher l'ouverture en fonction d'un Event Listener qui écoute quand le **button** ayant pour id **bouton** est cliqué.

```
const button = document.getElementById("bouton");

button.addEventListener('click', () => {
  window.open("destination.html");
});
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-2

[cf.]

Nous pouvons voir que, s'il n'y a pas de bug, la fenêtre s'ouvre bien lorsque nous cliquons sur le bouton. Nous pouvons passer plusieurs autres arguments, comme le nom de la fenêtre et ses options, parmi lesquelles figure sa dimension (en px) :

```
const button = document.getElementById("bouton");

button.addEventListener('click', () => {
  window.open("destination.html", 'destination', 'width=500, height=500');
});
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-3

[cf.]

Dans cet exemple, la fenêtre du nom de « destination » sera ouverte avec une taille de 500 / 500 px (si le navigateur n'est pas en plein écran). Il y a d'autres options spécifiables comme la présence de scrollbars, d'une toolbar (barre de menu), etc.

Pour fermer la fenêtre courante, il va nous falloir appeler la méthode `close()`. Pour cela, nous allons créer un Event Listener qui déclenchera l'appel lorsque le button ayant pour id `bouton2` sera cliqué. Mais pour cela, il nous faut récupérer l'élément HTML de la fenêtre ouverte avec le DOM de son objet window. Par ailleurs, il va nous falloir créer un Event Listener qui sera appelé une fois que la fenêtre « destination » sera bien ouverte pour qu'il n'y ait pas de bugs. Voilà comment faire :

```
const button = document.getElementById("bouton");

button.addEventListener('click', () => {
  let fenetre = window.open("destination.html", 'destination', 'width=500, height=500');

  fenetre.addEventListener('load', () => {
    let button2 = fenetre.document.getElementById('bouton2');

    button2.addEventListener('click', () => {
      fenetre.close();
    });
  });
});
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-4

[cf.]

Pour bien comprendre, détaillons les étapes :

- Dans la fonction anonyme du premier Event Listener, nous définissons une variable `fenetre` sur la valeur de retour de la méthode `window.open()` correspondant à une référence vers la fenêtre qui s'ouvre.
- Puis nous définissons un Event Listener qui déclenchera une fonction quand la fenetre sera chargée (évènement « load »). Dans cette fonction, nous récupérons l'élément HTML ayant pour id boutton2 en stockant sa référence dans la variable button2. Pour cela, nous utilisons le DOM (via le mot clé document) de l'objet window dont la référence est stockée dans la variable fenetre.
- Puis, toujours dans cette fonction, nous définissons un Event Listener qui déclenchera une fonction quand button2 sera cliqué.
- Cette dernière fonction fermera la fenêtre (`fenetre.close()`).

Nous pouvons voir que ce code fonctionne correctement.

Méthode alert()

La méthode `alert()` est une méthode que vous connaissez déjà bien. En général, nous l'utilisons sans le mot clé window, car comme nous l'avons dit, toutes les propriétés et méthodes de window peuvent être utilisées sans spécifier le mot clé, étant l'objet global par défaut. Cependant, c'est une bonne pratique de le spécifier afin de bien mettre en avant que c'est une méthode de l'objet window.

Exemple **Méthode alert()**

```
1 const button = document.getElementById("bouton");
2
3 button.addEventListener('click', () => {
4   window.alert("Message"); //ou alert("Message")
5 });
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-5

[cf.]

Dans cet exemple, une alerte se déclenche lorsque nous cliquons sur le button ayant pour id bouton.

Méthode prompt()

La méthode `window.prompt()` permet d'afficher une fenêtre avec une entrée utilisateur. La valeur passée par l'utilisateur peut être récupérée très facilement en stockant la valeur de retour de l'appel dans une variable. Prenons un exemple assez simple :

Exemple

```
1 const button = document.getElementById("bouton");
2
3 button.addEventListener('click', () => {
4   let mot = window.prompt("Quel mot afficher ?");
5   window.alert(mot);
6 });
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-6

[cf.]

Dans cet exemple, lorsque le bouton est cliqué, un prompt est affiché et l'utilisateur peut insérer un texte. La variable `mot` est définie sur la valeur de retour du prompt. Une alerte affiche le contenu de la variable `mot` qui contient donc la chaîne renseignée par l'utilisateur. Comme second argument, nous pouvons passer la chaîne qui sera renseignée par défaut dans le prompt :

```
1 const button = document.getElementById("bouton");
2
3 button.addEventListener('click', () => {
4   let mot = window.prompt("Quel mot afficher ?", "mot de votre choix");
5   window.alert(mot);
6 });
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-7

[cf.]

Maintenant que nous avons vu quelques exemples de méthodes de l'objet window, parlons de quelques sous-objets de window.

B. Exercice

Question 1

[solution n°1 p.17]

Créez un code permettant d'afficher une boîte de dialogue avec le message « *Bonjour, bienvenue sur mon site !* » lorsque la page HTML est chargée.

```
// code
```

Question 2

[solution n°2 p.17]

Créez un code permettant de demander à l'utilisateur son nom grâce à une boîte de dialogue et afficher un message de bienvenue dans la console avec le format : « *Bienvenue {prenom}* ». Exécutez le code en spécifiant le prénom « *Peter* » dans le prompt.

```
// code
```

Question 3

[solution n°3 p.17]

Créez un code permettant d'ouvrir une nouvelle fenêtre avec une URL « https://www.google.com[1] », avec la taille de 500/500 px quand le bouton `button` est cliqué.

```
let button = document.createElement('button');

button.innerHTML = "button";

document.body.appendChild(button);

//code
```

Question 4

[solution n°4 p.17]

Créez un code permettant de fermer la fenêtre actuelle lorsque le bouton `close` est cliqué.

```
let close = document.createElement('button');

close.innerHTML = "close";

document.body.appendChild(close);

// code
```

Question 5

[solution n°5 p.17]

Créez un code permettant de demander à l'utilisateur d'entrer une URL et ouvrez une nouvelle fenêtre avec cette URL quand le bouton `open` est cliqué :

```
let open = document.createElement('button');

open.innerHTML = "open";

document.body.appendChild(open);

// code
```

1 https://www.google.com/

III. Les sous-objets de window

A. Les sous-objets de window

L'objet window.screen

En JavaScript, l'objet screen représente l'écran sur lequel s'affiche le navigateur, donc de votre ordinateur, téléphone ou tablette. Il contient des informations sur la résolution de l'écran, la taille de l'écran et la profondeur de couleur.

Voici quelques-unes des propriétés de l'objet screen :

- **`screen.width`** : la largeur de l'écran en pixels,
- **`screen.height`** : la hauteur de l'écran en pixels,
- **`screen.availWidth`** : la largeur disponible de l'écran en pixels, c'est-à-dire la largeur de l'écran moins la barre des tâches et autres éléments de l'interface utilisateur,
- **`screen.availHeight`** : la hauteur disponible de l'écran en pixels, c'est-à-dire la hauteur de l'écran moins la barre des tâches et autres éléments de l'interface utilisateur,
- **`screen.colorDepth`** : la profondeur de couleur de l'écran en bits,
- **`screen.pixelDepth`** : la profondeur de couleur de l'écran en pixels.

Exemple **L'objet screen**

Nous pouvons afficher dans la console les différentes informations concernant l'écran :

```
1 console.log(screen.width);
2 console.log(screen.height);
3 console.log(screen.availWidth);
4 console.log(screen.availHeight);
5 console.log(screen.colorDepth);
6 console.log(screen.pixelDepth);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-13

[cf.]

Comme vous l'avez compris, `screen` est un sous-objet de window. Nous pouvons donc accéder aux propriétés de screen en spécifiant le mot window avant screen :

```
1 console.log(window.screen.width);
2 console.log(window.screen.height);
3 console.log(window.screen.availWidth);
4 console.log(window.screen.availHeight);
5 console.log(window.screen.colorDepth);
6 console.log(window.screen.pixelDepth);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-14

[cf.]

L'objet window.location

En JavaScript, l'objet **`location`** représente l'URL de la page web actuelle et fournit des propriétés dont des méthodes pour interagir avec cette URL.

Voici quelques-unes des propriétés de l'objet location :

- **`location.href`** : contient l'URL complète de la page web,
- **`location.protocol`** : contient le protocole utilisé pour accéder à la page web (par exemple « *http :* » ou « *https :* »),
- **`location.host`** : contient le nom d'hôte (nom de domaine) de la page web,
- **`location.pathname`** : contient le chemin d'accès de la page web, sans le nom d'hôte.

Voici quelques méthodes de l'objet location :

- **`location.assign()`** : permet de charger la page web dont l'URL est passée comme argument. Il se produit la même chose quand on modifie la valeur de `location.href`. Elle ajoute la nouvelle URL dans l'historique de navigation.
- **`location.replace()`** : permet de remplacer la page web actuelle par celle dont l'URL est passée comme argument. À la différence de la méthode précédente, la méthode `location.replace()` remplace l'URL courante par la nouvelle URL dans l'historique du navigateur. Il n'y a donc pas un ajout d'une entrée dans l'historique de navigation, mais un remplacement d'une entrée par une autre.
- **`location.toString()`** : permet de récupérer l'URL sous la forme d'une chaîne de caractères.

Exemple **L'objet location**

Voici quelques exemples d'utilisation de propriétés de l'objet location.

```
1 //Affichage de propriétés de window.location (le mot window est facultatif)
2
3 console.log(window.location.href);
4 console.log(window.location.protocol);
5 console.log(window.location.host);
6 console.log(window.location.pathname);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-15

[cf.]

Voici maintenant un exemple d'appel de la méthode `location.assign()` :

```
1 window.location.assign('destination.html');
```

Lien replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-16

[cf.]

Voici un exemple d'appel de la méthode `location.replace()` :

```
1 window.location.replace('destination.html');
```

Lien replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-17

[cf.]

Voici un exemple d'appel de la méthode `location.toString()` :

```
1 console.log(location.toString());
```

Lien replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-18

[cf.]

L'objet window.history

L'objet **`history`** expose plusieurs méthodes et propriétés qui permettent de naviguer vers les pages précédentes ou suivantes dans l'historique du navigateur, d'ajouter ou de remplacer des entrées dans l'historique, de vérifier le nombre d'entrées dans l'historique, etc.

La propriété **`length`** permet de connaître le nombre d'entrées dans l'historique de navigation (nombre de pages dans l'historique, voir la vidéo suivante).

Voici quelques méthodes utiles de l'objet **`window.history`** :

- **`history.back()`** : charge la page précédente dans l'historique du navigateur,
- **`history.forward()`** : charge la page suivante dans l'historique du navigateur,
- **`history.go(n)`** : charge la page située n positions dans l'historique du navigateur.

Avant de voir quelques exemples concrets d'utilisation de méthodes de `window.history`, nous allons apporter des modifications à l'architecture de notre répertoire.

Méthode **Modifier l'architecture du répertoire pour utiliser window.history**

En premier lieu, nous allons créer un fichier js du nom de destination.js. Ce script sera importé par la page HTML destination.html. Pour cela, nous allons modifier le fichier destination.html de manière à modifier le bouton 2 et à importer destination.js :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport">
    <title>Document</title>
</head>
<body>
    <p>destination</p>
    <button id="back">back</button>
    <script src="destination.js"></script>
</body>
</html>
```

Nous modifions au passage index.html afin d'ajouter un bouton « *back* » (pour revenir à la page d'avant) et afin de modifier notre premier bouton :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport">
    <title>Document</title>
</head>
<body>
    <p>index.html</p>
    <button id="destination">destination</button>
    <button id="forward">forward</button>
    <script src="script.js"></script>
</body>
</html>
```

Voyons maintenant un exemple d'utilisation des méthodes `back()` et `forward()`.

Exemple **L'objet window.history**

Voici un exemple où nous appelons les méthodes `back()` et `forward()`. Ces méthodes vont nous permettre de naviguer dans l'historique. Au niveau de notre script.js :

```
let destination = document.getElementById('destination');

const forward = document.getElementById('forward');

destination.addEventListener('click', () => {
destination = window.location.assign('destination.html');
});

forward.addEventListener('click', () => {
window.history.forward();
});
```

Et pour ce qui est de destination.js :

```
const back = document.getElementById('back');

back.addEventListener('click', () => {
  window.history.back();
});
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-19

[cf.]

Pour bien comprendre, retraçons les étapes :

- Dans script.js, nous récupérons la référence des deux boutons. Puis nous ajoutons des Event Listener. Le bouton destination va déclencher un `window.assign` vers destination.html. Le bouton forward va déclencher un `window.history.forward()`. Pour bien comprendre, cette dernière méthode réalisera la même opération que la flèche de navigation du navigateur : « → ».
- Si nous cliquons sur le bouton destination, le fichier destination.html est ouvert et une entrée est ajoutée dans l'historique. Le fichier destination.js est exécuté. Nous récupérons dedans le bouton back et nous lui affectons un Event Listener qui appellera la méthode `window.history.back`. Cette méthode réalisera la même opération que la flèche de navigation du navigateur : « ← ». Nous pouvons donc cliquer dessus pour revenir en arrière dans l'historique de navigation.
- Quand nous sommes à nouveau sur index.html, nous pouvons utiliser le bouton forward pour aller en avant dans l'historique de navigation, nous nous retrouvons alors à nouveau sur la page destinatinon.html.

Nous pouvons constater que ce système n'aurait pas fonctionné si nous avions utilisé avec le bouton destination la méthode `window.replace`, qui n'aurait pas inséré de nouvelle entrée dans l'historique de navigation. Par ailleurs, si nous chargeons la page et cliquons directement sur le bouton forward, rien ne se passera puisque l'historique ne comporte pas d'entrées positionnées après l'entrée courante. C'est exactement le même principe que quand nous utilisons les flèches de navigation d'un navigateur.

Maintenant que nous avons compris un peu mieux le fonctionnement de l'objet `window.history`, parlons d'un dernier objet qui fera l'objet de notre attention dans ce cours : `window.document.cookie`.

L'objet window.document.cookie

Comme vous le savez, le web d'aujourd'hui regorge des fameux cookies. On pourrait penser que les cookies sont des outils servant uniquement à harceler les utilisateurs de pubs envahissantes, mais c'est loin d'être le cas.

Un cookie est un fichier texte stocké sur l'ordinateur d'un utilisateur par un site web. Les cookies sont souvent utilisés pour stocker des informations qui peuvent être récupérées par le site web à une date ultérieure. Les informations stockées dans les cookies peuvent être utilisées pour personnaliser l'expérience utilisateur sur le site web, ou pour suivre les utilisateurs et collecter des données d'utilisation.

Donc les cookies vont aussi permettre de mémoriser votre statut de connexion sur un site, d'afficher automatiquement un site web dans votre langue, etc. Dans votre parcours de développeur, ils seront plus qu'indispensables à utiliser dans cette visée, car ils vont vous permettre de stocker des informations qui seront utilisables sur n'importe quelle page d'un domaine, donc du site web.

Méthode **Créer et récupérer la valeur de cookies**

Pour créer un cookie, nous pouvons utiliser l'objet cookie de JavaScript en procédant ainsi :

```
document.cookie = "clé=value ; expires=date_d'expiration ; path=chemin ; domain=domaine ; secure"
```

Voici une explication des différentes informations à renseigner :

- **clé** : le nom du cookie.
- **value** : la valeur à stocker dans le cookie.
- **date_d'expiration** : la date à laquelle le cookie expire. Si cette date n'est pas spécifiée, le cookie est considéré comme un cookie de session et sera supprimé lorsque le navigateur sera fermé.
- **chemin** : le chemin associé au cookie. Si cette option n'est pas spécifiée, le cookie est associé au chemin de la page courante.
- **domaine** : le domaine associé au cookie. Si cette option n'est pas spécifiée, le cookie est associé au domaine de la page courante.
- **secure** : si cette option est spécifiée, le cookie ne sera envoyé que sur une connexion HTTPS.

Prenons un exemple, ajoutons deux cookies (username et mdp) dans notre fichier script.js :

```
const destination = document.getElementById('destination');

const forward = document.getElementById('forward');

destination.addEventListener('click', () => {
  const destination = window.location.assign('destination.html');
});

forward.addEventListener('click', () => {
  window.history.forward();
});

//création d'un objet Date pour créer une date d'expiration :

const dateExp = new Date();

dateExp.setMonth(dateExp.getMonth() + 1); //nous ajoutons 1 mois à la date

//nous interrogeons l'utilisateur sur son username et son mdp avec window.prompt()

const username = prompt("Username");

const mdp = prompt("Mot de passe");

//creation des cookies

document.cookie = `username=${username} ; expires=${dateExp.toUTCString()}`;

document.cookie = `mdp=${mdp} ; expires=${dateExp.toUTCString()}`;
```

Nous interrogeons donc l'utilisateur sur son username et son mdp. Puis nous créons des cookies pour le domaine actuel qui seront stockés 1 mois, avec les valeurs renseignées par l'utilisateur. Notons que nous utilisons de préférence la méthode `toUTCString()` pour récupérer la date de notre objet Date. Maintenant, nous pourrons accéder à nos cookies sur n'importe quelle page du domaine. Par exemple, modifions destination.js pour récupérer les valeurs des cookies :

```
const back = document.getElementById('back');

back.addEventListener('click', () => {
  window.history.back();
});

//document.cookie est une chaîne qui contient tous les cookies séparés par "; "
//Nous utilisons la méthode split pour créer un tableau contenant les cookies

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

//Nous utilisons la méthode split pour séparer les clés des valeurs dans un forEach et ajouter les clés et valeurs à un tableau identifiants

let identifiants = new Array();

tableauId.forEach((ligne) => {
    let tab = ligne.split("=");
    identifiants.push([tab[0], tab[1]]);
});

//Nous pouvons afficher le tableau dans la console

console.log(identifiants);
```

Lien replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-20

[cf.]

Dans cet exemple, nous récupérons dans la page destination.html la valeur des cookies. Nous créons un tableau qui contiendra les clés / valeurs des cookies, et nous l'affichons dans la console. Nous pouvons constater que les cookies créés sont accessibles à toutes les pages du domaine, ce qui n'est évidemment pas le cas des variables basiques que nous déclarons dans un fichier js.

B. Exercice

Question 1

[solution n°6 p.18]

Utilisez l'objet screen pour afficher la largeur de l'écran de l'utilisateur dans la console.

```
console.log(/*code*/);
```

Question 2

[solution n°7 p.18]

Utilisez l'objet location pour rediriger l'utilisateur vers la page « *google.com*[1] » en ajoutant une entrée dans l'historique, quand le bouton open est cliqué.

```
let open = document.createElement('button');

open.innerHTML = "open";

document.body.appendChild(open);

// code
```

Question 3

[solution n°8 p.18]

Utilisez l'objet location pour rediriger l'utilisateur vers la page « *google.com*[2] » sans ajouter d'entrée dans l'historique, quand le bouton open est cliqué.

```
let open = document.createElement('button');

open.innerHTML = "open";

document.body.appendChild(open);

// code
```

Question 4

[solution n°9 p.18]

Utilisez l'objet `window.document.cookie` pour stocker un cookie avec une durée de validité de 1 heure. Sa clé est username et sa valeur « *user1* ».

```
// code

console.log(document.cookie);
```

Question 5

[solution n°10 p.19]

Utilisez l'objet cookie pour récupérer la valeur du cookie stocké dans l'exercice précédent.

```
let expirationDate = new Date();
expirationDate.setHours(expirationDate.getHours() + 1);

document.cookie = "username=user1; expires=" + expirationDate.toUTCString();

let valeurDuCookie;

// code

console.log(valeurDuCookie);
```

IV. Essentiel

Le JavaScript Browser BOM (Browser Object Model) est un ensemble d'objets qui permettent d'interagir avec le navigateur web. Nous avons vu que le BOM se compose principalement de l'objet window et de ses sous-objets.

Dans la première partie de ce cours, nous avons étudié les méthodes les plus courantes de l'objet window : `open()`, `close()`, `alert()` et `prompt()`. Ces méthodes permettent d'ouvrir et de fermer des fenêtres du navigateur, ainsi que d'afficher des messages d'alerte ou de demander des entrées de données à l'utilisateur.

Dans la seconde partie, nous avons vu comment interagir avec les sous-objets de window, tels que screen, location, history et document.cookie. Nous avons appris comment accéder à des informations telles que la résolution de l'écran, l'URL de la page, l'historique de navigation, les cookies stockés sur le navigateur, etc.

---

1 https://www.google.com/
2 https://www.google.com/

Il est important de comprendre que le JavaScript Browser BOM peut être utilisé à des fins malveillantes, comme l'installation de logiciels malveillants ou la collecte de données à l'insu de l'utilisateur. Par conséquent, il est essentiel d'utiliser ces fonctions avec précaution et de mettre en place des mesures de sécurité pour protéger les utilisateurs.

En conclusion, le JavaScript Browser BOM est un outil puissant pour interagir avec le navigateur web et améliorer l'expérience utilisateur. Il permet d'accéder à des données concernant la fenêtre de l'utilisateur et les données en entrée. Il permet par ailleurs de contrôler les pages et l'historique de navigation. C'est donc un outil puissant qu'il faut savoir maîtriser et sécuriser afin d'éviter les piratages.

V. Auto-évaluation

A. Exercice

Vous cherchez à créer un système d'authentification rapide avec un identifiant et un mot de passe sans passer par un formulaire HTML. L'authentification se fera via deux prompts interrogeant l'utilisateur sur son identifiant et son mot de passe. Attention, ce système ne sera pas du tout conventionnel ni sécurisé, c'est simplement un exemple pour utiliser les notions développées.

Question 1

[solution n°11 p.19]

- Créez un projet avec une page connexion.html où les prompts seront lancés.
- Créez une page traitement.html ainsi que deux scripts : connexion.js et traitement.js.
- Créez une page connected.html contenant simplement un paragraphe « *vous êtes connectés* » et une page incorrect.html avec un paragraphe « *l'identifiant ou mot de passe est incorrect* ».
- Importez dans connexion.html le script connexion.js et dans traitement.html le script taitement.js.
- Créez dans connexion.js le système de prompt et créez des cookies stockant l'identifiant et le mot de passe à partir des entrées utilisateur pendant 1 heure.

Question 2

[solution n°12 p.19]

Utilisez l'objet `window.location` pour ouvrir la page traitement.html depuis le script connexion.js une fois qu'un bouton connexion est cliqué, en ajoutant une nouvelle entrée dans l'historique de navigation. Le bouton connexion devra être créé avec JS. Créez dans traitement.js un script permettant de vérifier si les informations de connexion correspondent à :

- identifiant : TonyStark
- mdp : starktower

Si c'est bien le cas, quand un bouton « *espace utilisateur* » est cliqué, ouvrir dans un nouvel onglet la page connected.html sinon la page incorrect.html. Le bouton espace utilisateur devra être créé avec JS. La fenêtre ouverte devra avoir une dimension de 400 / 400 px.

B. Test

Exercice 1 : Quiz

[solution n°13 p.19]

Question 1

Quelle est la méthode de l'objet **window** qui permet d'ouvrir une nouvelle fenêtre de navigateur ?

- ○ alert()
- ○ prompt()
- ○ open()

Question 2

Quelle est la méthode de l'objet **window** qui permet de fermer la fenêtre de navigateur actuelle ?

- close()
- exit()
- quit()

Question 3

Quel est le sous-objet de l'objet **window** qui permet de manipuler l'historique de navigation ?

- location
- history
- screen

Question 4

Quelle méthode de l'objet **window** permet d'afficher une boîte de dialogue pour saisir une valeur ?

- prompt()
- alert()
- confirm()

Question 5

Quel sous-objet de l'objet **window** permet de récupérer l'URL de la page courante ?

- location
- history
- screen

Solutions des exercices

p. 7 Solution n°1

Sortie attendue : Dans une boîte de dialogue : « *Bonjour, bienvenue sur mon site !* »

```
window.addEventListener('load', function() {
alert("Bonjour, bienvenue sur mon site !");
});
```

Lien replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-8

[cf.]

p. 7 Solution n°2

Sortie attendue : « *Bienvenue Peter* »

```
let nom = prompt("Quel est votre nom ?");
console.log("Bienvenue " + nom);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-9

[cf.]

p. 7 Solution n°3

```
let button = document.createElement('button');

button.innerHTML = "button";

document.body.appendChild(button);

button.addEventListener('click', () => {
  window.open("https://www.google.com", 'google', 'width=500, height=500');
})
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-10

[cf.]

p. 7 Solution n°4

```
let close = document.createElement('button');

close.innerHTML = "close";

document.body.appendChild(close);

close.addEventListener('click', () => {
  window.close();
})
```

Lien replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-11

[cf.]

p. 7 Solution n°5

```
let open = document.createElement('button');

open.innerHTML = "open";

document.body.appendChild(open);

let url = prompt("Entrez l'URL :");

open.addEventListener('click', () => {
  window.open(url);
});
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-12

[cf.]

p. 13 **Solution n°6**

```
console.log(window.screen.width);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-21

[cf.]

p. 14 **Solution n°7**

```
let open = document.createElement('button');

open.innerHTML = "open";

document.body.appendChild(open);

open.addEventListener('click', () => {
  window.location.assign('https://www.google.com');
})
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-22

[cf.]

p. 14 **Solution n°8**

```
let open = document.createElement('button');

open.innerHTML = "open";

document.body.appendChild(open);

open.addEventListener('click', () => {
  window.location.replace('https://www.google.com');
});
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-23

[cf.]

p. 14 **Solution n°9**

Sortie attendue : « *username=user1* »

```
let expirationDate = new Date();
expirationDate.setHours(expirationDate.getHours() + 1);

document.cookie = "username=user1; expires=" + expirationDate.toUTCString();

console.log(document.cookie);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-24

[cf.]

p. 14 Solution n°10

```
let expirationDate = new Date();
expirationDate.setHours(expirationDate.getHours() + 1);

document.cookie = "username=user1; expires=" + expirationDate.toUTCString();

let valeurDuCookie;

let tableau = document.cookie.split('=');

valeurDuCookie = tableau[1];

console.log(valeurDuCookie);
```

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-25

[cf.]

p. 15 Solution n°11

Voici le projet mis en place :

[cf. GDWFS-060-AW - Cas pratique 1-20230614T100622Z-001.zip]

p. 15 Solution n°12

Voici le projet mis en place :

[cf. GDWFS-060-AW - JS Browser BOM (window - screen - Location - History, etc)-20230919T120706Z-001.zip]

Lien Replit : https://replit.com/@javascript-studi/JS-Browser-BOM-Code-26[1]

[cf.]

Exercice p. 15 Solution n°13

1 https://replit.com/@javascript-studi/JS-Browser-BOM-Code-26

Question 1

Quelle est la méthode de l'objet **window** qui permet d'ouvrir une nouvelle fenêtre de navigateur ?

- ○ alert()
- ○ prompt()
- ◉ open()

La méthode **open()** permet d'ouvrir une nouvelle fenêtre de navigateur. Elle prend en paramètre l'URL à charger et le nom de la fenêtre (optionnel). Nous pouvons aussi passer des options comme la taille de la fenêtre, la présence de scrollbars, etc.

Question 2

Quelle est la méthode de l'objet **window** qui permet de fermer la fenêtre de navigateur actuelle ?

- ◉ close()
- ○ exit()
- ○ quit()

La méthode **close()** permet de fermer la fenêtre de navigateur actuelle.

Question 3

Quel est le sous-objet de l'objet **window** qui permet de manipuler l'historique de navigation ?

- ○ location
- ◉ history
- ○ screen

L'objet **history** permet de manipuler l'historique de navigation dans le navigateur. Il permet notamment de naviguer en arrière et en avant dans l'historique.

Question 4

Quelle méthode de l'objet **window** permet d'afficher une boîte de dialogue pour saisir une valeur ?

- ◉ prompt()
- ○ alert()
- ○ confirm()

La méthode **prompt()** permet d'afficher une boîte de dialogue pour saisir une valeur. Cette boîte de dialogue contient un message (optionnel) ainsi qu'un champ de saisie et deux boutons : « ***OK*** » et « ***Annuler*** ».

Question 5

Quel sous-objet de l'objet **window** permet de récupérer l'URL de la page courante ?

- ◉ location
- ○ history
- ○ screen

Le sous-objet **location** permet de récupérer l'URL de la page courante. Il permet également de modifier l'URL de la page courante avec des méthodes comme **assign()** ou **replace()**.