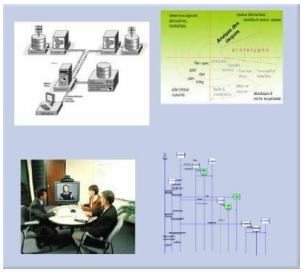
EXERCICES - DÉCOUVERTE DU LANGAGE JAVASCRIPT - SÉQUENCE "DÉVELOPPER DES PAGES WEB"

Appui technique

Introduction

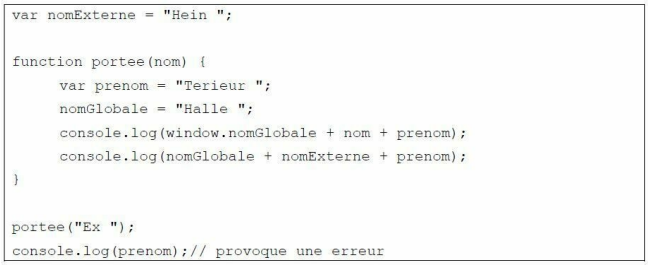


1. Fichier externe .JS

Le but de cet exercice est d'externaliser le code JavaScript dans un fichier .js puis de tester l'intégration de code Html dans les fonctions JavaScript.

1. Ouvrez le ficher fourni « **Mon_1er_code.html** » dans un navigateur ; un message doit s'afficher dans une boite de dialogue dès le chargement et un autre message apparaît au clic sur le bouton.
2. Ouvrez le fichier fourni « **Mon_1er_code.html** » dans un éditeur ou un EDI Web.
3. Créez un fichier JavaScript nommé « **Mon_1er_code.js** »
4. Déplacez le code JavaScript depuis le fichier Html vers le fichier externe JavaScript et importez ce fichier par l'instruction Html adéquate. Testez que tout se déroule de la même manière.
5. Au lieu d'écrire le titre <h2> de la page en Html, faites-le écrire au même endroit par JavaScript grâce à la méthode **document.write(...)**. Testez que tout se déroule de la même manière.
6. Modifiez une fonction **alert(...)** pour y passer en paramètre également du code Html (par exemple pour afficher le message en gras ou en élément **<h2>**). Que remarquez-vous ?

2. La Portée



```
var nomExterne = "Hein ";

function portee(nom) {
      var prenom = "Terieur ";
      nomGlobale = "Halle ";
      console.log(window.nomGlobale + nom + prenom);
      console.log(nomGlobale + nomExterne + prenom);
}

portee("Ex ");
console.log(prenom);// provoque une erreur
```

Code de l'image ci-dessus :

```
var nomExterne = "Hein ";
function portee(nom) {
var prenom = "Terieur ";
nomGlobale = "Halle ";
console.log(window.nomGlobale + nom + prenom);
console.log(nomGlobale + nomExterne + prenom);
}
portee("Ex ");
console.log(prenom);// provoque une erreur
```

Recopiez le code ci-dessus au sein d'une page Html valide et ouvrez le fichier dans un navigateur en affichant le débogueur. Observez les messages affichés en console.

Modifiez le code fourni afin que « **console.log(prenom)** » ne provoque pas d'erreur.

3. L'objet Arguments

En vous inspirant de la version proposée dans le support d'apprentissage, créez au sein d'une page Html valide une fonction sans paramètre qui gère automatiquement la liste des paramètres éventuels via l'objet **arguments** de façon à calculer le périmètre d'un polygone quelconque :

- Pas de paramètre : erreur
- 1 seul paramètre = carré (→périmètre = 4 x côté)
- 2 paramètres = rectangle (→périmètre = 2 x longueur + 2 x largeur)
- 3 paramètres = triangle (→périmètre = somme des 3 côtés)
- Supérieur à 2 = le nombre d'arguments correspond au nombre de côtés du polygone (→périmètre = somme des côtés)

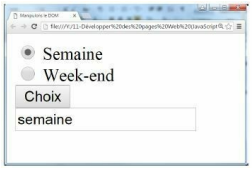
Les résultats peuvent être affichés en console ou boîte d'alerte ; ils contiennent la désignation du type de forme détecté et le périmètre calculé.

Testez et contrôlez les différents cas de figure (avec des valeurs simples). Placez des points d'arrêt ou exécutez pas à pas pour vérifier le bon déroulement du code.

4. Manipulons le dom

Créez une page Html contenant 2 radio-buttons exclusifs l'un de l'autre, un bouton simple, et un champ texte (pour le moment la balise de structure Html **<form>** importe peu et on peut l'ignorer).

Lorsque l'on clique sur le bouton **Choix**, le texte associé au radio-bouton sélectionné est recopié dans le champ texte.



Vous utiliserez l'attribut Html « **onclick** » du bouton **Choix** pour appeler une fonction JavaScript (interne ou externe) qui passera en revue les différents boutons-radio de la page.

Remarque

L'usage de l'attribut « **onclick** » ajoute du JavaScript intrusif maintenant déprécié...

Testez

À l'aide du débogueur du navigateur, placez un point d'arrêt dans la boucle et observez comment JavaScript représente les éléments Html sous forme d'objets dotés de propriétés et méthodes.

5. Le post-it

Créez une page Html valide contenant :

- un élément ** ** identifié **« postIt »** placé en absolu et invisible au démarrage (CSS) correspondant à la zone de texte sur fond jaune.
- un bouton « Affiche » permet de rendre visible la **div** « **postIt** » et d'y insérer le texte suivant : « Vous avez cliqué sur le bouton 'Affiche' ! »
- un bouton « Cache » permet de cacher la **div** « **postIt** ».

Le texte « Merci de bien vouloir me survoler » permet également d'afficher la **div** « **PostIt** » lors du survol de la souris, et d'y insérer le texte suivant :

« C'est gentil de me survoler... »

Quand la souris sort du texte, la **div** se cache.

Manipulez par JavaScript les attributs **.style.visibility** pour la div « **postIt** » et **.innerHTML** pour modifier le contenu de la div **« postIt ».**

Résultats à obtenir :

Tout le code JavaScript pourra être externalisé dans un script séparé de la page Html.

Implémentez par JavaScript la capture des événements nécessaires grâce à une fonction en auto-exécution dès le chargement de la page.

6. Le chrono

Réalisez un petit chrono qui affiche un champ texte **()** et 3 boutons **Start**, **Pause** et **Stop.**

- L'appui sur le bouton **Start** démarre le chrono.
- L'appui sur le bouton **Pause** arrête puis redémarre le chrono.
- L'appui sur le bouton **Stop** arrête et remet à zéro le chrono.

Tout le code JavaScript sera écrit de manière non intrusive en JavaScript 'moderne'.

Il s'agit de développer l'application pas-à-pas en testant à chaque étape.

Variables globales nécessaires :

- Références des objets JavaScript correspondant aux 3 boutons et au **span**
- 1 variable pour référencer un timer JavaScript
- 1 variable pour compter le nombre de secondes écoulées (initialisée à 0)

Une première fonction auto-exécutable permet d'ajouter une propriété '**paramTps**' aux objets boutons **Start** et **Pause**, et de capturer l'événement clic sur le bouton **Start** ; la fonction associée ('**startChrono**') à cet événement réalise les traitements :

- Désactiver la capture de l'événement **clic** sur ce bouton **Start.**
- Masquer ce bouton **Start** et afficher les 2 autres boutons (par modification des classes de styles CSS associées aux objets **button**).
- Activer un **timer** JavaScript associé à une fonction anonyme qui calcule le temps écoulé et le convertit en nombre d'heures, minutes et secondes (voir code fourni).

Nota Bene

Pour ajouter une propriété à un objet JavaScript, il suffit de l'initialiser :

```
// ajoute une propriété à l'objet button
btnStart.paramTps = tpsEcoule;
```

Code de la fonction de calcul du temps écoulé :

```
// algo de calcul de nombre haures, minutes et secondes écoulées
var startTime = new Date();
decompte = setInterval(function() {
      // 1- Convertir en secondes :
      var seconds = Math.round(
            (new Date().getTime() - startTime.getTime()) / 1000
            + e.target.paramTps); // e représente l'event déclencheur
            // e.target représente l'objet déclencheur
            // ici : bouton start ou bouton pause
            // (cette prop a été ajoutée aux boutons)
      // 2- Extraire les heures:
      var hours = parseInt( seconds / 3600 );
      seconds = seconds % 3600; // secondes restantes
      // 3- Extraire les minutes:
      var minutes = parseInt( seconds / 60 );
      seconds = seconds % 60; // secondes restantes
      // 4- afficher dans le span
      chronoP.innerHTML = ajouteUnZero(hours)
            +":"+ajouteUnZero(minutes)
            +":"+ajouteUnZero(seconds);
      // 5- incrémenter le nombre de secondes
      tpsEcoule += 1;

}, 1000); // fin de function anonyme dans setInterval()
```

La petite fonction **ajouteUnZero(temps)** reste à écrire ; elle permet simplement d'afficher systématiquement chaque nombre sur 2 chiffres ('04' et non '4') par simple concaténation du « 0 » manquant.

Testez à l'aide du débogueur et mettez au point jusqu'à ce que cette première étape se passe correctement (les boutons **Pause** et **Stop** restent inactifs).

Activez la capture des événements **clic** sur chacun des boutons **Pause** et **Stop** ; associez-les aux fonctions **'pauseChrono'** et **'stopChrono'** et écrivez déjà les structures de ces fonctions.

Écrivez le contenu de la fonction '**pauseChrono**' de manière à :

- Désactiver le **timer**
- Modifier la fonction associée à l'événement **clic** du bouton **Pause** (suppression puis ajout) de manière à ce qu'il puisse relancer le décompte à l'aide de la fonction **'startChrono** ' quand l'utilisateur cliquera à nouveau sur ce bouton
- Mémoriser le temps écoulé dans la propriété **paramTps** du bouton **Pause**

Testez la mise en pause et la reprise.

Écrivez le contenu de la fonction '**stopChrono**' de manière à :

- Désactiver le **timer**
- Désactiver la capture des événements **clic** des boutons **Pause** et **Stop**
- Réactiver la capture de l'événement **clic** du bouton **Start**
- Réinitialiser le temps écoulé et l'affichage en **span**
- Masquer les boutons inutiles

Testez l'application complète.

Variante pour aller plus loin

- Pour cet entraînement, on s'est appliqué à activer/désactiver systématiquement la capture des événements sur les boutons ; mais un bouton masqué (ou désactivé) ne risque pas de déclencher un événement ! On peut encore ajuster ces captures d'événements au strict nécessaire.
- Modifier les couleurs du texte affiché en fonction de l'état du chrono (arrêt, pause, en cours)
- Modifier les libellés des boutons plutôt que de les masquer et jouer sur les fonctions associées aux captures d'événements

7. Le contrôle de champ

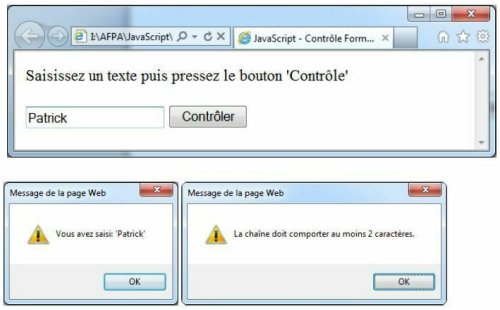
Réalisez un petit formulaire Html valide qui demande la saisie d'un texte puis qui contrôle le texte saisi quand l'utilisateur clique un bouton :

- Si le nombre de caractères saisi est inférieur à 2, affichez une alerte et terminer.
- Sinon, affichez le texte saisi et déclenchez l'action associée au form Html (par exemple http://www.afpa.fr).

Testez les 3 méthodes pour capturer l'événement clic du bouton :

- L'attribut « **onclick** » Html
- La propriété **onclick** sur l'objet du DOM représenté par le bouton.
- La méthode **addEventListener (' click ', ...)** sur l'objet bouton.

Résultat à obtenir :



Variantes pour aller plus loin

Le texte saisi doit correspondre à une adresse email probable ; il s'agit ici, côté client Web, de vérifier la forme du texte, pas l'existence de cette adresse sur un serveur de mail.

Mais la forme d'une adresse email reste assez compliquée... En voici les principales règles de gestion :

- 1° règle : saisie obligatoire
- 2° règle : doit contenir un caractère '@' mais pas en début ni en fin de chaîne
- 3° règle : doit contenir au moins un caractère '.' (point) mais pas en début ni en fin de chaîne
- 4° règle : ne doit contenir que des caractères lettre, chiffre, souligné, point, arobas
- autres règles : il doit y avoir au moins un point après le arobas, il peut ne pas y en avoir avant...

Certaines de ces règles peuvent être implémentées par Html (obligatoire), d'autres feront l'objet de traitements JavaScript.

On peut adopter plusieurs attitudes :

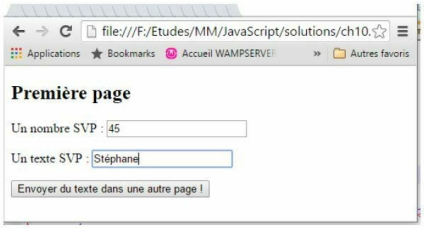
- contrôles par algorithmes détaillés qui aboutissent à des messages clairs et précis ;

- contrôle global à l'aide d'une 'expression régulière' (on trouve de nombreux masques sur le Web...) ;
- contrôles fins avec messages clairs et précis en mixant ces 2 techniques (contrôles par expressions régulières sur des parties de l'adresse, elle-même découpée par algorithmes ou expressions régulières...)

8. Le passage d'information

Réalisez une petite application qui utilise 2 pages Web comme ci-dessous :

Dans la 1ère page, vous demandez à saisir un texte (pour cet exercice, omettez la balise de structure **<form>**).



Vous récupérez le texte et le nombre saisis dans des variables que vous insérez dans une nouvelle url pour appeler la 2ème page : **...page2.html?clé1=valeur1& clé2=valeur2**

La 2ème page affiche ce que vous avez saisi précédemment :

En pratique

- Utilisez l'objet JavaScript **location.**
- Pour accepter tous types de caractères à l'intérieur d'une url valide, utilisez les fonctions **encodeURI()** et **decodeURI().**
- Pour récupérer une partie de l'url, vous pouvez utiliser la méthode **split ()** des objets JavaScript **string .**

Rappel

L'url contient l'adresse de la page cible. On peut lui ajouter des informations sous la forme de paires 'clé=valeur' sans espace ni caractères spéciaux, tout comme le fait un navigateur pour soumettre un formulaire en 'method get'.

Pour se faire, il faut tout d'abord ajouter le caractère '?' suivi des paires 'clé=valeur' séparées par le caractère '&'.

Testez en utilisant aussi bien l'appel de la méthode **location.replace()** que l'affectation de la propriété **location.href** et observez bien le comportement du navigateur.

9. L'objet Navigator

Affichez dans une page Html, par des alertes ou dans des zones de texte, les propriétés de l'explorateur qui interprète votre page. Les affichages seront déclenchés par clic sur des boutons.

Exemple d'image-écran

Exécutez cette page sur vos différents navigateurs et relevez les variantes en recherchant comment JavaScript peut vous aider à identifier le type de navigateur de l'utilisateur.

Recherchez sur le Web des 'trucs et astuces' JavaScript pour détecter le type de browser de l'utilisateur et confrontez ces solutions avec vos conclusions.

Fin de l'exercice

CRÉDITS

Œuvre collective de l'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Équipe de conception (IF, formateur, médiatiseur)

Damien Bin- formateur

Benoit Hézard - formateur

Chantal Perrachon – Ingénieure de formation

Crédits

OEUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la Direction de l'ingénierie

DATE DE MISE À JOUR

09/09/2021