Comprendre JavaScript

Que ce soit pour maitriser un nouveau langage de programmation ou pour faire vos connaissances en la matière, plongez dans les bases du JavaScript afin de découvrir sa passionnante histoire, intimement liée à l’essor du Web, et entamez l’écriture de vos premiers scripts. Vous dynamiserez ainsi votre site web et vous serez en mesure de l’agrémenter de toutes sortes d’interactions et de comportements originaux.

En 1995 apparait le LiveScript, un langage de script s’inspirant du Java et l’ancêtre de JavaScript, est rapidement intégré à Netscape 2 sous le nom de JavaScript, en hommage au populaire langage dont in s’est librement inspiré.

Retenez toutefois que les deux langages restent fondamentalement différents et servent des usages distincts tout en n’obéissant pas du tout à la même syntaxe.

L’année suivante, Microsoft développe une version similaire, le JScript, et l’intègre à Internet Explorer 3.

Netscape soumet alors sa version du langage à l’organisme central de normalisation, l’ECMA International, qui le standardise définitivement sous le nom d’**ECMAScript**. En claire ECMAScript est depuis la référence de base dont le JavaScript est la déclination la plus célèbre.

Retenez qu’il existe 2 familles principales de langages de programmation :

* Les **langages compilés**, pour lesquels le code source passe tout d’abord par un compilateur qui va le convertir.
* Les **langages interprétés**, avec lesquels votre code est directement lu.

Javascript appartient à cette dernière catégorie, c’est-à-dire que les scripts que vous développez sont directement lus par le navigateur, à travers son **interpréteur** **JavaScript** (V8 pour GoogleChrome, Chakra pour Internet Explorer et Microsoft Edge, SpiderMonkey pour Mozilla Firefox, JavaScriptCore pour Safari).

Avec HTML et CSS, JavaScript est la 3ème brique essentielle du développement Web. Il est possible de bâtir des applications totalement autonomes en JavaScript, au-delà du pur navigateur, grâce à des frameworks spécifiques.

Le JavaScript est un langage qui s’exprime coté client, c’est-à-dire directement dans le cadre du navigateur. Il s’oppose aux langages coté serveur, comme PHP, qui vont donc être interprétés par le serveur avant d’être renvoyés au navigateur.

Un script PHP demande au serveur Web d’extraire du contenu (liste des articles) depuis une base de données. Ce contenu brut est alors renvoyé vers le navigateur qui le met en forme à l’aide de balises HTML, de propriétés CSS et de codes JavaScript. HTML et CSS présente les informations, JavaScript gère les comportements et les interactions possibles.

Deux manières de définir le code JavaScript :

* En interne dans la page HTML à l’aide des balises <script> … </script>

*<html>*

*…*

*<body>*

*<script>*

*console.log(« Hello World ») ;*

*</script>*

*</body>*

*</html>*

* En appelant un fichier externe qui présente l’extension .js

*<script type= « text/javascript » src= « script.js »></script>*

Depuis HTML 5, il est possible d’omettre cet attribut type, la simple utilisation de la balise <script> sous-entend qu’il va s’agir de code JavaScript.

On a longtemps conseillé de placer cette déclaration dans l’entête de la page HTML, c’est-à-dire entre les balises <head> … </head>. Depuis les choses ont changées et désormais, il est parfois préférable de charger le fichier JavaScript en dernier lieu, avant la fermeture de la balise </body>

Depuis quelques années, il est possible de gagner du temps grâce aux services de « sandboxing ». Ce sont des services gratuits qui font cohabiter dans une fenêtre unique code HTML, CSS et JavaScript. Le plus célèbre : JSFiddle.net.

Sitôt connecté au site, vous découvrez une interface scindée en quatre blocs :

* + Un bloc en haut à gauche pour le HTML
  + Un bloc en haut à droite pour le CSS
  + Un bloc en bas à gauche pour le JavaScript

Le quatrième bloc fait apparaitre le résultat lors d’un clic sur le bouton run, dans la barre supérieure.

Pense-bête : le site W3Schools.com propose un récapitulatif très complet de l’ensemble des notions du JavaScript.

Définir des variables

Les variables constituent l’un des éléments les plus importants du langage : elles stockent tout type de données, des chaines de caractères comme des valeurs numériques ou d’autres formes de structures, et vous les manipulez pour réaliser des opérations ou afficher leur contenu. Il est primordial d’entamer votre découverte du JavaScript en les maîtrisant au mieux, elles vont soutenir toutes vos applications, des plus simples aux plus ambitieuses.

Les constantes ne voient pas leur valeur évoluer. Vous définissez ainsi des informations stables dans le temps, que vous réutiliserez à loisir.

Les variables sont susceptibles d’évoluer au fil de l’exécution de votre code.

*var prenom="David";*

*var nom="Dupont";*

*const label="Utilisateur";*

*alert(label+ " : "+prenom+" "+nom);*

il est possible de regrouper la définition de plusieurs variables sur une même ligne, en les séparant d’une virgule

*var prenom, nom, age=18;*

*prenom="Sophie";*

Les variables sont très variées. On retrouve

* Des chaines de caractères, que l’on définit entre guillemets ou apostrophes. On peut utiliser le caractère d’échappement \ pour éviter de casser la chaine et faire croire à JavaScript que la chaine est terminée.

*var titre='bienvenue à l\'académie';*

* Des valeurs numériques. Retenez que le séparateur en JavaScript est le point.

*var note=12.5;*

* Des valeurs booléennes, qui permet de vérifier si une condition est vraie (true) ou fausse (false).
* Des tableaux de données qui contiennent une liste de valeurs successives.

*var langages=["php","html","css"];*

* Des objets complets pour lesquels on associe une valeur à une série de propriétés. Ces éléments sont encadrés par des accolades

*var utilisateur={nom:"Dupont",prenom:"David",age:"18"};*

L’opérateur **typeof** renvoie le type d’une variable JavaScript

*var prenom="David";*

*var age=18;*

*var utilisateur={identifiant:"david",motdepasse:"azerty"};*

*alert(typeof(prenom));*

*alert(typeof(age));*

*alert(typeof(utilisateur));*

dans ce cas string, puis number, puis object

L’opérateur \*= va multiplier la valeur de gauche avec celle de droite.

*var num1=5;*

*num1 \*= 2;*

*alert(num1);*

Attention : l’opérateur + sert aussi à concaténer des chaines de caractères, c’est-à-dire les coller.

*var prenom="David";*

*var nom="Dupont";*

*var nomComplet = prenom+nom;*

La fonction prompt() sert à récupérer la saisie de l’utilisateur, dont le résultat de la saisie sera ensuite enregistré dans une variable.

*var tempFahr=prompt("La température en degrés Fahrenheit que vous souhaitez convertir ?");*

*var tempCelsius=(tempFahr-32)/1.8;*

*alert("La température en degré Celsius est : "+tempCelsius);*

Introduction aux boucles conditionnelles

Grace aux variables, vous stockez des informations et vous réalisez tout type de traitements. Mais pour aller plus loin, vous allez être confronté à la nécessité d’introduire des boucles conditionnelles : vous définissez ainsi différents cheminements en fonction de certaines valeurs, qui vont conduire à des traitements distincts. Il s’agit là encore d’un concept essentiel en programmation, qui se décline à la quasi-totalité des langages.

Le type booléen, susceptible de prendre deux valeurs true ou false, est tout particulièrement utilisé dans les boucles conditionnelles pour vérifier si le résultat d’un test est vrai ou faux.

*var resultat=12;*

*if (resultat >= 10){*

*alert("vous avez la moyenne");*

*} else {*

*alert("vous ferez mieux la prochaine fois");*

*}*

Opérateurs de comparaison, il en existe 8, qui renvoient un booléen (true ou false) :

==  égal à

!=  différent de

>  supérieur à

>= supérieur ou égal à

<  inferieur à

<= inférieur ou égal à

=== contenu et type égal à

!== contenu et type différent de

Ils se complètent d’opérateurs logiques :

&& et

|| ou

*var resultat=12;*

*if (resultat >= 10 && resultat < 16){*

*alert("vous avez la moyenne");*

*} else if (resultat >= 16) {*

*alert ("felicitation");*

*} else {*

*alert("vous ferez mieux la prochaine fois");*

*}*

Prenez l’habitude d’insérer des commentaires (double slash // ou encadrés des signes /\* et \*/)

*// voici un commentaire*

*/\* et encore un autre \*/*

*var resultat =17;*

**On utilise la condition switch lorsqu’une variable peut prendre plusieurs valeurs distinctes qui vont conduire à des embranchements différents**.

*var dateActuelle = new Date();*

*var jour = dateActuelle.getDay();*

*var nomJour;*

*switch (jour){*

*case 0:*

*nomJour="dimanche";*

*break;*

*case 1:*

*nomJour="lundi";*

*break;*

*case 2:*

*nomJour="mardi";*

*break;*

*case 3:*

*nomJour="mercredi";*

*break;*

*case 4:*

*nomJour="jeudi";*

*break;*

*case 5:*

*nomJour="vendredi";*

*break;*

*case 6:*

*nomJour="samedi";*

*break;*

*default:*

*nomJour="inconnu";*

*break;*

*}*

La méthode getDay() renvoie le numéro du jour courant dans la semaine (0 pour dimanche, 1 pour lundi, …). L’instruction break à la fin de chaque case sert à sortir de la boucle switch, pour éviter que les lignes suivantes s’exécutent.

Les boucles conditionnelles permettent aussi de répéter une série de traitements tant qu’une condition est remplie.

Cas de la boucle while (tant que)

*var compteur= 1;*

*while (compteur <=10) {*

*console.log("compteur :"+compteur);*

*compteur++*

*}*

Pensez bien à faire évoluer le compteur et à vérifier la condition afin d’éviter une boucle infini.

Cas de la boucle do … while (faire jusqu’à)

*var compteur= 1;*

*do {*

*console.log("compteur :"+compteur);*

*compteur++*

*} while (compteur <=10);*

Elle place le test conditionnel à l’issu du traitement. Les instructions sont donc obligatoirement exécutées au moins une fois avant que l’on vérifie si la condition est remplie.

Cas de la boucle for

*for (var compteur= 1; compteur <= 10; compteur++) {*

*console.log("compteur :"+compteur);*

*}*

Elle exécute une série de traitements pendant un certain nombre de répétitions. Vous l’utiliserez en particulier pour traiter l’ensemble des valeurs d’un tableau. Vous n’indiquez plus seulement une condition de base à vérifier, mais vous définissez un état initial et le niveau d’incrémentation.

On peut aussi décrémenter une valeur (compteur--).

En réalité, les différents blocs de la boucle for sont facultatifs. Par contre les ; sont obligatoires dans l’écriture de la boucle for

*var constructeurs = ["renault","peugeot","toyota"];*

*var i=0;*

*var nbr=constructeurs.length; //stockage dynamique de la longueur du tableau constructeurs*

*var texte="";*

*for (; i < nbr;) {*

*texte += constructeurs[i] +"\n"; //\n retour à la ligne*

*i++;*

*}*

*console.log(texte);*

Gérer les tableaux

Les tableaux figurent parmi les objets JavaScript les plus importants : vous les utiliserez sans cesse pour regrouper une série de valeurs évoluant autour d’un même thème, que vous pouvez extraire ou compléter à loisir. Prenez le temps de vous familiariser avec sa structure et ses méthodes, pour envisager le développement de programmes plus ambitieux.

Pour les objets littéraux, qui associent une série d’identifiants et de valeurs, il s’agit d’encadrer cette succession de valeurs par des accolades, en les séparant d’une virgule. On accède ensuite à la valeur de l’un des identifiants à l’aide de la notation point.

*var utilisateur = {prenom : "Jean", nom : "DUPONT", identifiant= "jdupont"} ;*

*alert (utilisateur.nom) ;*

C’est une solution très pratique pour établir une liste de propriétés sous forme de **couples**, que vous interrogez facilement. On parle également de **tableau** **associatif** et vous ajoutez de nouveaux éléments à l’aide de simples crochets.

utilisateur["motDePasse"] = "azerty" ;

Contrairement aux objets littéraux, les tableaux n’associent pas une série de propriétés à des identifiants. C’est une simple liste de valeurs rangées dans des cases que l’on peut pointer de manière individuelle. Toutes les valeurs ont le même type.

*var constructeurs = ["renault","peugeot","toyota"];*

On dit que cet array (le nom du tableau en anglais) compte trois items que l’on parcourt en indiquant leur index (qui permet de pointer vers un case du tableau) Attention l’index démarre à 0.

*alert(constructeurs[0]);*

**Parcourir et compléter un tableau**

Un site Web qui invite les utilisateurs à s’inscrire en créant leur propre compte.

Stockage de la liste des utilisateurs dans un tableau.

Inscription d’un utilisateur => vérification de son inexistence dans le tableau

*function verifier(utilisateur,adherents){*

*for(var i = 0; i < adherents.length; i++){*

*if (utilisateur == adherents[i]){*

*return "true";*

*}*

*}*

*}*

*var nouvelUtilisateur = prompt("Veuillez saisir votre identifiant");*

*var listeUtilisateur = ["jean","sophie","nicolas"];*

*if (verifier(nouvelUtilisateur,listeUtilisateur)){*

*alert("le compte exite déjà");*

*} else {*

*alert("bienvenue sur notre site");*

*listeUtilisateur.push(nouvelUtilisateur);*

*}*

La fonction verifier() prend deux arguments : une nouvel utilisateur et le tableau complet avec la liste de comptes déjà enregistrés. Cette fonction parcours le tableau jusqu’à *adherents.length* et pour chaque valeur dans le tableau *adherents[i]* la compare avec le nouvel utilisateur. En cas de succès la fonction renvoie « true ».

En dehors de la fonction, on définit la variable nouvelUtilisateur stockant la saisie de l’utilisateur. Si l’utilisateur n’est pas trouvé dans le tableau, on l’ajoute en fin de tableau. Sinon on avertit l’utilisateur de son existence.

La méthode **unshift()** ajoute un élément en début du tableau.

La méthode **shift()** supprime le premier élément du tableau.

La méthode **pop()** supprime le dernier élément du tableau.

Par ailleurs, vous pouvez créer facilement un tableau à partir d’une chaine de caractères à l’aide de la méthode **split()**. Signifie diviser. Il suffit de renseigner le séparateur à prendre en compte.

*var utilisateurs = "jean sophie nicolas";*

*var monTableau = utilisateurs.split(" ");*

*alert(monTableau.length);*

A l’inverse, vous créez une chaine de caractères à partir d’un tableau à l’aide de la méthode **join().**

*var listeUtilisateur = monTableau.join(‘-‘);*

*alert(listeUtilisateur);*

On sépare d’un trait d’union tous les éléments figurant dans le tableau crée précédemment.

Créer des fonctions

Lorsque vous effectuez des traitements redondants, il est impératif de les déclarer dans des fonctions spécifiques, afin de les appeler à chaque fois que vous en avez besoin.

Non seulement votre code gagne en cohérence, mais vous vous préparez également une « bibliothèque » prête à l’emploi, que vous pourrez réutiliser dans de nombreux projets ou communiquez à d’autres développeurs. L’union fait la force.

* En appelant alert(), par exemple, vous exécutez en réalité **une fonction native** du langage qui va afficher le message passé entre parenthèse à l’écran.

JavaScript propose de nombreuses fonctions natives, pour afficher un message utilisateur alert(), lui demander de saisir des valeurs prompt() ou d’afficher une boite de message avec les boutons OK et Annuler confirm(). Vous avez intérêt à les passer en revue (<https://www.w3schools.com/jsref/obj_window.asp>)

* Vous avez intérêt à identifier les traitements redondants dans votre code et à les isoler dans des fonctions que vous pouvez appeler à loisir.
* Il est possible de personnaliser les valeurs qu’elles traitent à l’aide d’arguments que l’on passe entre parenthèses
* Vous définissez une fonction personnalisée de la manière suivante :

*function maFonction (argument) {*

*// e code de la fonction*

*}*

La déclaration commence par le mot-clé function afin d’identifier le caractère particulier des instructions qui suivent. Le nom de la fonction vient ensuite. Cette structure ne sert qu’à définir une fonction. En l’état son code ne sera pas exécuté.

Exemple qui demande à l’utilisateur de saisir un nombre entier avant de calculer s’il est pair ou impair

*function parite (nombre) {*

*if (nombre%2 == 0) {*

*alert (nombre + « est pair ») ;*

*} else {*

*alert (nombre + « est impair ») ;*

*}*

*}*

*var nbrTest ;*

*nbrTest = parseInt (prompt (« indiquez le nombre à évaluer ») ;*

*parite (nbrTest) ;*

Pour vérifier si un nombre entier est pair ou impair, nous utilisons l’opérateur modulo, identifié par le signe %. En clair, si le résultat de l’opération nombre%2 vaut 0, c’est que le nombre est pair.

Dans l’appel de la fonction, nous utilisons 2 fonctions natives : parseInt() qui renvoie un entier (les valeurs à virgule flottante seront ainsi arrondies) et prompt() qui invite l’utilisateur à saisir une valeur.

**Utilisation de variables et de valeurs de retour**

* Il est primordial d’indiquer le même nombre d’arguments dans la définition de la fonction et dans son appel, sous peine de rencontrer une erreur.
* Vous pouvez avoir besoin de sortir des résultats de la fonction afin de les traiter par la suite ou de les passer à une autre fonction. C’est le rôle du mot-clé return, que vous placez à l’issue de la fonction et qui met fin à celle-ci.

Exemple : sortie d’un résultat

*function surface (longueur, hauteur) {*

*var aire = longueur\*hauteur ;*

*return aire ;*

*}*

*alert (« l’aire est de : «  + surface (20,10) + «  m2 ») ;*

L’ordre d’exécution est simple. On commence par déclarer la fonction surface, qui s’enregistre en mémoire sans s’exécuter. On appelle ensuite la fonction alert(), qui attend la réception de son argument unique. En clair, la fonction surface est bien exécutée avant alert et on rencontre donc pas d’erreur.

* Que se passe-t-il si l’on souhaite sortir plusieurs résultats d’une fonction ?

*function redimensionnement (largeur, hauteur) {*

*var largDest, hautDest ;*

*largDest = largeur/1.5 ;*

*hautDest = hauteur/1.5 ;*

*return { largeur : largDest, hauteur : hautDest } ;*

*}*

*var obj = redimensionnement (800,600) ;*

*alert (obj.largeur) ;*

*alert (obj.hauteur) ;*

Son véritable intérêt tient à l’utilisation du mot-clé return. On retourne ici un couple de variables, entre accolades, avec à chaque fois le nom de la clé et de sa valeur. On commence par définir une variable en lui associant l’appel de la fonction, avec les arguments nécessaires. On utilise ensuite la notation **point**. Ainsi obj.largeur va contenir la valeur de la largeur redimensionnée.

**Variables globales et variables locales**

Vous devez maitriser la **portée** des variables, c’est-à-dire leur champ d’action (on parle de **scope**, dans la terminologie anglophone). Lorsque certaines variables sont déclarées au sein des fonctions : on parle alors de variables locales, elles sont accessibles qu’au sein de cette portion de code et vous ne pouvez pas les appeler ou les utiliser en dehors de cette fonction.

Les variables déclarées en dehors des fonctions sont dites globales.

*Exemple variable globale Exemple variable locale*

*var msg = « bonjour » ; function parler () {*

*function parler () { var msg = « bonjour » ;*

*alert (msg) ; }*

*} parler() ;*

*parler() ; alert (msg) ;*

Les évènements

Les évènements correspondent à toutes les interactions et les actions que sont susceptibles d’effectuer les visiteurs de votre site. Clic sur un élément, survol, pression sur une touche du clavier, saisie dans un formulaire, envoi des champs remplis, …. Vous associez des fonctions et des traitements à tous ces évènements, ce qui ve contribuer à renforcer le dynamisme de vos pages.

Sur les pages Web, l’utilisateur s’attend à interagir avec tous les éléments qui les composent.

Qu’appelle-t-on un évènement ?

Il s’agit de toutes les interactions entreprises par les utilisateurs sur votre page, que ce soit à l’aide de leur souris, de leur clavier ou même de leur écran tactile.

Parmi les évènements les plus utiles, on peut citer le clic sur un élément de la page HTML, le survol de la souris, la pression sur une touche du clavier, le focus sur le champ d’un formulaire.

*<span onclick="alert('vous avez cliqué');">cliquez ici</span>*

Dans cet exemple, l’attribut onclick prévu en HTML exécute une fonction JavaScript lorque l’on clique sur l’élément correspondant. On peut associer cet attribut à n’importe que élément HTML et pas seulement à un bouton ou un lien hypertexte.

Il existe des dizaines d’attributs HTML de ce type.

onmousover : le survol d’un élément

onwheel : utilisation de la molette de la souris

onkeypress : la pression d’une touche sur le clavier

onfocus : la sélection d’un champ de formulaire

oncopie : la copie de texte dans le presse papier

Cependant je vous préconise d’utiliser plutôt le **gestionnaire d’évènements JavaScript**

1-apprendre à maitriser les sélecteurs JavaScript

La solution la plus simple consiste à associer des **id** aux éléments que vous ciblez, dans le code HTML.

Vous avez alors la possibilité d’appeler la méthode **getElementById()** de l’objet **document**, afin de pointer précisément vers un élément spécifique.

Vous exécuterez ensuite la méthode **addEventListener()**, en précisant le type d’évènement que vous souhaitez écouter.

*<input type="text" placeholder="Entrez un montant en euros" id="euro">*

*<input type="submit" value="convertir" id="conversion">*

*<div id="resultat"></div>*

*<script>*

*var taux=1.1059;*

*var element = document.getElementById("conversion");*

*var source = document.getElementById("euro");*

*element.****addActionListener****("click",function(){*

*var montant = source.value;*

*document.getElementById("resultat").innerHTML = montant\*taux +"$";*

*});*

*</script>*

Nous avons ici associé une fonction anonyme au gestionnaire d’évènements.

*<!DOCTYPE html>*

*<html>*

*<head><title>essai</title></head>*

*<body>*

*<div id="resultat"></div>*

*<div id="survol">survolez-moi*

*<div id="survol2">survolez-moi aussi</div>*

*</div>*

*<div id="curseur"></div>*

*<script>*

*var survol=document.getElementById("survol");*

*var resultat=document.getElementById("resultat");*

*survol.addEventListener("mouseover",function(e){*

*resultat.innerHTML="L'événement "+e.type+" concerne l'id: "+e.target.id;*

*});*

*var position=document.getElementById("curseur");*

*document.addEventListener("mousemove",function(e){*

*position.innerHTML="Curseur x: "+e.clientX+" px<br/>Curseur y: "+e.clientY+" px";*

*});*

*</script>*

*</body>*

*</html>*

Nous associons ici un argument **e** à la fonction anonyme comprise dans les deux gestionnaire d’évènements afin de pointer vers le même objet **Event.**

Dans le 1er cas, avec l’évènement mouseover (survol) on utilise les propriétés type pour récupérer la nature de l’évènement et target.id pour récupérer l’identifiant de l’élément l’ayant déclenché.

Dans le 2ème cas, on surveille le déplacement de la souris avec mousemove. La propriété clientX renvoie la position du curseur de la souris selon l’axe des abscisses et clientY la position du curseur de la souris selon l’axe des ordonnées (point d’origine en haut à gauche)

Déployer des formulaires interactifs

Les formulaires constituent l’un des éléments incontournables des sites web : brisez leur monotonie en leur associant des comportements dynamiques et utilisez le gestionnaire des évènements pour procéder à une série de validations. Vous aurez ainsi l’assurance que les visiteurs les remplissent correctement, en leur présentant éventuellement des messages d’avertissement, et vous n’obtiendrez que les valeurs attendues.

Pour renforcer les interactions et distribuer au mieux les valeurs saisies par les utilisateurs, avant de les traiter, vous devez vous reposer sur du code JavaScript.

Les expressions régulières, une technique qui dépasse le seul cadre du JavaScript mais qui permet notamment de valider des champs en confrontant la saisie de l’utilisateur à un motif auquel elle doit se conformer.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<form id="formulaire">

Email <input type="text" id="mail" />

<input type="submit" value="Envoyer" />

</form>

<script>

var form = document.**getElementById**("formulaire");

form.addEventListener("submit",function(e){

var email = document.getElementById("email");

if (email.value == "") {

email.**value** = "Veuillez saisir votre email";

**e.preventDefault();**

}

});

</script>

</body>

</html>

On utilise la méthode getElementById() afin de pointer vers le formulaire et on associe un écouteur d’évènements qui sera déclenché lors de son envoi (évènement submit).

A l’aide de la propriété value, on récupère la valeur du champ : si elle est vide, on indique à l’utilisateur qu’il n’a pas rempli le champ et on empêche l’envoi du formulaire avec la méthode preventDefault().

On évite ainsi les allers-retours entre le navigateur et le serveur, avec une réactualisation de la page (c’est-à-dire la perte des champs déjà remplis).

**Utiliser les propriétés des formulaires**

L’une des propriétés les plus utiles des formulaires : value. En l’associant à un sélecteur de champ, comme getElementById(), vous récupérez la valeur de l’élément.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<form id="form">

Suppléments :

<input type="checkbox" class="supp" name="repas" value="salade" />Salade<br/>

<input type="checkbox" class="supp" name="repas" value="tomate" />Tomate<br/>

<input type="checkbox" class="supp" name="repas" value="oignon" />Oignon<br/>

<input type="button" id="valider" value="Valider" />

</form>

<div id="commande"></div>

<script>

var valider = document.getElementById("valider");

valider.addEventListener("click",function(e){

var commande = document.**getElementsByClassName**("supp");

var texte="";

var i;

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

if (commande[i].checked) {

texte=texte+commande[i].value+" ";

}

}

document.getElementById("commande").innerHTML="Vos suppléments : "+texte;

});

</script>

</body>

</html>

On utilise la méthode getElementsByClassName() afin de sélectionner tous les éléments définis avec la classe supp. Il s’agit donc des 3 cases à cocher. Au sein d’une boucle for(), on parcourt le tableau des valeurs ainsi relevées et on vérifie si leur propriété checked est vraie. On compose ainsi progressivement une chaine de caractères avec la valeur des cases cochées et on l’affiche dans le bloc div

Les listes déroulantes disposent également de propriétés spécifiques :

option, pour relever les différentes options

selectedIndex, pour récupérer l’indice de la valeur sélectionnée (le premier élément est à l’indice 0)

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<form id="form">

<select id="ville">

<**option**>Votre ville</option>

<option>Poitiers</option>

<option>La Roche Sur Yon</option>

</select>

</form>

<div id="resultat"></div>

<script>

var ville = document.getElementById("ville");

ville.addEventListener("**change**",function(){

document.getElementById("resultat").innerHTML=ville.options[ville.**selectedIndex**].value;

});

</script>

</body>

</html>

L’évènement change se déclenche lorsque l’on sélectionne une autre valeur. On récupère ensuite l’indice de l’option sélectionnée à l’aide des propriétés options et selectedIndex. Puis nous affichons en toutes lettres la valeur correspondante.

Le basculement vers le champ suivant, c’est-à-dire la perte de focus, se traduit par le fait de quitter un champ spécifique.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<form id="form">

<input type="text" id="champ" />

</form>

<div id="resultat"></div>

<script>

var champ = document.getElementById("champ");

champ.addEventListener("focus",function(){

**champ.style.backgroundColor= "green";**

});

champ.addEventListener("blur",function(){

champ.style.backgroundColor= "red";

});

</script>

</body>

</html>

Ici deux évènements réciproques :

* focus qui se déclenche lorsque l’on sélectionne le champ
* blur qui se déclenche lorsque l’on sort du champ

Chaines de caractères

Les chaines de caractères figurent parmi les types de données que vous utiliserez le plus fréquemment. Elles s’accompagnent de nombreuses méthodes pour les mettre en forme, les combiner, extraire du contenu ou rechercher des termes spécifiques : apprenez à les maitriser afin de réaliser tout type de traitements sur les site Web, en particulier pour assurer une plus grande cohérence dans la saisie des utilisateurs.

Le JavaScript n’est pas un langage au typage fort.

*var chaine1="54";*

*var chaine23=54;*

*alert(chaine1+12);*

*alert(chaine2+12);*

Le 1er résultat de l’addition va conduire à une nouvelle concaténation : 5412

Le 2ème résultat de l’addition donnera : 66

Pour convertir une valeur numérique en chaine, utilisez la méthode **toString()**.

*alert(chaine2.toString()+12);*

**Opérations sur les caractères et les chaines**

*var chaine="Bonjour";*

*alert(chaine.toLowerCase());*

*alert(chaine.length);*

*alert(chaine.charAt(1));*

toLowerCase() : met les caractères de la chaine en minuscules

toUpperCase() : met les caractères de la chaine en majuscules

length : renvoie la longueur de la chaine de caractères (nombre de caractères espace compris)

charAt(n) : renvoie le caractère situé à la position n dans la chaine de caractères (le comptage des caractères débute à 0.

Pour mettre l’initial d’un mot en majuscule et le reste en minuscule. La méthode **slice(a,b)** découpe une chaine de caractères à partir de a jusqu’au caractère b (facultatif).

*alert(chaine.charAt(0).toUpperCase()+chaine.slice(1).toLowerCase());*

Il est possible de rechercher des chaines entières à l’aide de la méthode **indexOf().** Celle-ci renvoie la position de la première occurrence de la chaine recherchée et retourne -1 si la recherche est infructueuse. Attention : cette méthode est sensible à la casse

*if (chaine.indexOf("JavaScript")!== -1) {*

*alert("chaine trouvée");*

*} else {*

*alert("chaine non trouvée");*

*}*

La méthode **split()** découpe la chaine concernée en un pur tableau JavaScript. En guise d’argument, indiquez le separateur

*var chaine="comment ça va";*

*var coupe=chaine.split("");*

*alert(coupe[2]); //affiche m*

*var espace=chaine.split(" ");*

*alert (espace[2]); //affiche va*

*var decoupe = chaine.split("ça");*

*alert (decoupe[0]); //affiche comment*

*alert (decoupe[1]); //affiche va*

*alert (decoupe[2]); //renvoie undefined*

Les méthodes substr() et substring() sont relativement proches.

Pour **substr(),** on indique le caractère de départ et la longueur de la sous-chaine à extraire.

Pour **substring()**, on précise les caractères de départ et de fin

*var email="identifiant@fournisseur.fr";*

*var decoupe = email.indexOf("@");*

*var fournisseur = email.substring(decoupe+1);*

*alert(fournisseur.split(".")[0].toLowerCase());*

Il est possible d’enchainer les appels des méthodes avec la notation point.

Les dates et le temps

En JavaScript, la notion de temps intervient de deux manières : vous utilisez tout d’abord l’objet Date pour afficher, calculer ou manipuler tout type de dates, qu’ils s’agissent d’années, de jours, de mois ou même de millisecondes, et vous disposez par ailleurs de fonctions visant à décaler ou au contraire à répéter à intervalles réguliers vos traitements.

Pour créer une date vous devez donc manipuler le constructeur de l’objet Date qui dispose d’une série de méthodes et de propriétés.

var date = new Date() ;

alert(date) ;

Vous récupérez ici en réalité l’heure courante (par défaut). Il s’agit de la notation angosaxonne et vous obtenez une chaine du type « Fri May 26 2018 16 :17 :09 GTM+0200 ».

La première solution pour adapter le formatage de cette date consiste à utiliser les multiples méthodes qui lui sont associées.

getFullYear() : renvoie l’année

getMonth() : renvoie le numéro du mois (de 0 à 11)

getDate() : renvoie le numéro du jour (de 1 à 31)

getHours() : renvoie l’heure courante (de 0 à 23)

getMinutes() : renvoie la minute actuelle (de 0 à 59)

getSeconds() : renvoie les secondes (de 0 à 59)

getMilliseconds : renvoie les millisecondes (de 0 à 999)

getDay() : renvoie le jour de la semaine (de 0 à 6, la semaine commence le dimanche)

alert(*"*nous sommes le *"+ date.getDate() + "/"+eval(date.getMonth()+1)+ "/"+date.getFullYear()+" "+date.getHours()+"h"+date.getMinutes()) ;*

ce qui donne : nous sommes le 17/8/2018 11h40

Une autre solution consiste à recourir aux paramètres locaux de la machine de l’utilisateur.

A l’aide de la méthode toLocaleDateString(), vous personnalisez leur affichage et adoptez une notation française.

var options = {weekday : *"long", year : "numeric", month : "long", day : "numeric", hour : "numeric", minute : "numeric};*

var heure1 = new Date().toLocaleDateString(*"fr-FR");*

var heure2 = new Date().toLocaleDateString(*"fr-FR", options);*

alert(heure1);

alert(heure2);

Pour heure1, nous utilisons la méthode toLocaleDateString() avec le code ISO pour la notation française. Le résultat est 17/08/2018.

Pour heure2, grâce à l’objet options, nous complétons l’affichage en indiquant les éléments que nous souhaitons voir apparaitre ainsi que leur forme syntaxique (long : pour un affichage en toutes lettres, numerique : pour une valeur numerique). Le résultat est vendredi 17 aout 2018 à 11 :40

Vous avez la possibilité de créer un objet Date en lui associant une date spécifique.

var date2 = new Date(*"2018/08/17");*

var date3 = new date();

date3.setFullYear(*"2018");*

*date3.setMonth(“8”);*

*date3.setdate(“17”);*

*alert(date2.toLocaleDateString(“fr-Fr”,options));*

*alert(date3.toLocaleDateString(“fr-Fr”,options));*

**Retarder ou répéter une fonction**

Retarder l’exécution d’une fonction ou au contraire de la rappeler à intervalles réguliers, c’est le rôle de deux méthodes supplémentaires, setTimeout() et setInterval(), qui ne s’appliquent pas à l’objet Date mais plutôt à la fenêtre complète de votre application.

Les deux méthodes s’utilisent de la même manière. En guise d’arguments, vous leur passez respectivement la fonction à exécuter, le délai ou l’intervalle exprimé en millisecondes et éventuellement des paramètres complémentaires.

Exécution toutes les secondes (cad toutes les 1000 millisecondes) d’une fonction avec l’heure courante. La méthode utilisée est setTimeout(). De plus nous utilisons la méthode toLocaleDateString() de l’objet Date, qui renvoie l’heure, les minutes et les secondes dans la syntaxe locale de l’utilisateur.

<span id=*“horloge”></span>*

*<script>*

*function chrono(){*

*var d = new date();*

*var t = d.tlLocaleDateString();*

*document.getElementById(“horloge”).innerHTML=t:*

*}*

*setInterval(function(){chrono()},1000);*

*</script>*

Création d’un effet d’animation classique (affichage d’un diaporama automatique).

Nous préparons une série d’images dont le nom est construit autour de la même numérotation

Lorsque l’on atteint la dernière image de la série, on boucle et on réinitialise à nouveau ce compteur.

<img id=*“diaporama” src=”fleur01.jpg”/>*

*<script>*

*var image= document.getElementById(“diaporama”);*

var i=1 ;

function diapo(i){

if (i==5){

i=1;

}

image.src=”fleur0”+i+”.jpg”;

i=i+1;

setTimeout(diapo,2000,i);

}

diapo(i);

</script>

Les classes et les objets

Face aux projets les plus ambitieux, vous avez intérêt à structurer votre code JavaScript. Même si ce n’est pas un langage orienté objet à proprement parler, c’est un langage intimement fondé sur les objets : vous pouvez ainsi reproduire le principe de **classes**, en leur associant des propriétés et des méthodes, pour développer un code plus facilement réutilisable et moins redondant.

JavaScript n’est pas un langage orienté objet, mais plutôt un langage orienté prototype.

Petit rappel sur les principaux éléments de la programmation orientée objet.

Imaginez un véhicule terrestre :

Elle présente une série de propriétés (nombre de roues, couleur, nombre de portes,…). attributs

Elle est également capable de faire des actions (avancer, reculer, tourner, …). methodes

Elle peut interagir avec d’autres objets : entre le conducteur et le véhicule terrestre, une relation évidente les lie. association

Elle sert de base à d’autres objets qui en dérivent naturellement : une voiture ou un camion reprend les propriétés et les méthodes d’un véhicule terrestre, et peut les compléter avec des caractéristiques qui lui sont propres comme pour le camion le volume de la remorque. héritage

On crée la classe Voiture, qui sera instanciée sous forme d’objets individuels, présentant chacun des propriétés spécifiques et ayant accès à des méthodes communes.

La classe Voiture ou Camion hérite de la classe VehiculeTerrestre et empruntent ses caractéristiques fondamentales, en les complétant.

JavaScript étant un langage de programmation orienté prototype, on n’instancie pas des classes mais directement des objets définis sous forme de prototypes, en s’articulant autour de la pure création de fonctions.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

function Voiture(marque) {

this.marque=marque;

}

Voiture.prototype.avancer = function(){

alert("la voiture "+this.marque + " avance.");

}

var voiture1 = **new** Voiture("Peugeot");

voiture1.avancer();

</script>

</body>

</html>

Plus récemment, les équipes responsables de l’élaboration et de la certification du langage, ont proposé l’apparition de la syntaxe class, avec la sortie d’ECMAScript2015, qui agit en réalité comme un alias

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

**class** Voiture {

**constructor**(marque){

this.marque=marque;

}

avancer() {

alert("la voiture "+this.marque + " avance.");

}

}

var voiture1 = new Voiture("Peugeot");

voiture1.avancer();

</script>

</body>

</html>

En coulisse, class Voiture va en réalité construire la fonction voiture() à partir du constructeur défini à la ligne suivante. Les autres fonctions définies dans le cadre de la classe vont compléter le prototype.

* Utilisez le mot-clé class suivi du nom unique à utiliser. Le corps de la classe, c’est-à-dire la définition de son constructeur et de ses différentes méthodes, est déclaré entre accolades.
* La méthode constructor() est indispensable et elle doit être unique à chaque classe : on ne peut pas définir deux constructeurs puisque cette méthode est implicitement appelée lors de l’instanciation de la classe.

Voyons maintenant l’héritage. Nous commençons par créer une nouvelle classe **Quadrilatere**, qui va nous servir de base à de multiples sous-classes.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

class Quadrilatere {

constructor(hauteur, largeur){

this.nom="Quadrilatere";

this.hauteur=hauteur;

this.largeur=largeur;

}

typeFigure() {

console.log("je suis un "+this.nom + " .");

}

}

var quadri = new Quadrilatere(300,400);

quadri.typeFigure();

console.log("La largeurde ce quadrilatere est de "+quadri.largeur);

</script>

</body>

</html>

La notion d’héritage et de sous-classe est importante. Une sous-classe hérite des propriétés et des méthodes de la classe générique ou super-classe. Depuis ECMAScript 2015, il est possible d’adopter ce type de syntaxe à l’aide du mot-clé extends.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

class Quadrilatere {

constructor(hauteur, largeur){

this.nom="Quadrilatere";

this.hauteur=hauteur;

this.largeur=largeur;

}

typeFigure() {

console.log("je suis un "+this.nom + " .");

}

}

**class Carre extends Quadrilatere {**

**constructor(longueur) {**

**super(longueur, longueur);**

**this.nom="Carre";**

**}**

**get aire() {**

**return this.hauteur \* this.largeur;**

**}**

**set aire(valeur) {**

**this.aire=valeur;**

**}**

**}**

var quadri = new Quadrilatere(300,400);

quadri.typeFigure();

console.log("La largeurde ce quadrilatere est de "+quadri.largeur);

**var carre = new Carre(5);**

**carre.typeFigure();**

**console.log("L'aire de ce carre est de "+carre.aire);**

</script>

</body>

</html>

Ce mot-clé extends indique que la classe Carre hérite de la classe Quadrilatere.

Dans la mesure où un carré présente quatre côtés de même longueur, il n’est plus nécessaire de dissocier la hauteur et la largeur : on appelle toutefois la méthode super() pour pointer vers le constructeur de la classe parente et ainsi faire coïncider les valeurs. En clair, nous aurons ici : this.hauteur= this.largeur= this.longueur. après avoir appelé la méthode super(), on peut utiliser toutes les propriétés.

On poursuit la déclaration en créant deux méthodes (accesseurs), un getter et un setter.

La seule originalité tient à l’utilisation des propriétés hauteur et largeur alors que nous ne les avons pas définies explicitement dans cette sous-classe. C’est bien la preuve du concept d’héritage, la sous-classe héritant de la classe parente.

Que se passe-t-il si l’on se rend compte, pour des besoins spécifiques à une sous-classe, qu’une méthode de la classe parente ne convient pas ?

C’est là qu’intervient la réécriture des méthodes de sous-classe.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

class Quadrilatere {

constructor(hauteur, largeur){

this.nom="Quadrilatere";

this.hauteur=hauteur;

this.largeur=largeur;

}

typeFigure() {

console.log("je suis un "+this.nom + " .");

}

}

class Carre extends Quadrilatere {

constructor(longueur) {

super(longueur, longueur);

this.nom="Carre";

}

get aire() {

return this.hauteur \* this.largeur;

}

set aire(valeur) {

this.aire=valeur;

}

}

**class Rectangle extends Quadrilatere {**

**constructor(hauteur, largeur) {**

**super(hauteur,largeur);**

**this.nom = "restangle";**

**}**

**typeFigure() {**

**console.log("Quant à moi, je suis un "+this.nom + " .");**

**}**

**}**

var quadri = new Quadrilatere(300,400);

quadri.typeFigure();

console.log("La largeur de ce quadrilatere est de "+quadri.largeur);

var carre = new Carre(5);

carre.typeFigure();

console.log("L'aire de ce carre est de "+carre.aire);

**var rect = new Rectangle(50, 60);**

**rect.typeFigure();**

</script>

</body>

</html>

On profite là encore des propriétés standards de la classe parente, en particulier nom. Ici, on réécrit très simplement la méthode parente en réutilisant le même nom mais en changeant la valeur retournée.

Sachez qu’il est possible d’associer à une classe une méthode générale, qui ne dépend pas de ses différentes instances : on parle de méthode statique et elle a généralement le rôle d’utilitaire au sein de la classe.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

class Quadrilatere {

constructor(hauteur, largeur){

this.nom="Quadrilatere";

this.hauteur=hauteur;

this.largeur=largeur;

}

typeFigure() {

console.log("je suis un "+this.nom + " .");

}

**static calcul(hauteur,largeur) {**

**return largeur \* hauteur;**

**}**

}

class Carre extends Quadrilatere {

constructor(longueur) {

super(longueur, longueur);

this.nom="Carre";

}

get aire() {

return this.hauteur \* this.largeur;

}

set aire(valeur) {

this.aire=valeur;

}

}

class Rectangle extends Quadrilatere {

constructor(hauteur, largeur) {

super(hauteur,largeur);

this.nom = "restangle";

}

typeFigure() {

console.log("Quant à moi, je suis un "+this.nom + " .");

}

}

var quadri = new Quadrilatere(300,400);

quadri.typeFigure();

console.log("La largeurde ce quadrilatere est de "+quadri.largeur);

var carre = new Carre(5);

carre.typeFigure();

console.log("L'aire de ce carre est de "+carre.aire);

var rect = new Rectangle(50, 60);

rect.typeFigure();

**var rectangle = new Rectangle(12,46);**

**console.log("l'aire de ce rectangle est de "+Quadrilatere.calcul(rectangle.hauteur,rectangle.largeur));**

</script>

</body>

</html>

JavaScript ne repose en réalité que sur la manipulation d’objets au sens large : String, Number, Array ou Date ne sont que des objets.

Il parait donc possible de créer des sous-classes de tous ces objets, afin de les adapter à des besoins spécifiques.

Vous disposez de nombreuses méthodes pour récupérer la date courante et vous définissez plus globalement un objet de type **Date** avec le mot-clé new, à la manière de l’instanciation d’une classe.

<!DOCTYPE html>

<html>

<head><title>essai</title></head>

<body>

<script>

var date = new Date();

var nomJour;

console.log("nous sommes le " + date.getDate() + "/" + eval(date.getMonth()+1) + "/" + date.getFullYear() + " "

+ date.getHours() + "h" + date.getMinutes());

/\* voici la déclaration d'une sous-classe qui hérite de cet objet et qui dispose d'une méthode personnalisée \*/

class MaDate extends Date {

constructor() {

super();

}

getDateFormatee() {

var mois = ['Jan','Fev','Mar','Avr','Mai','Juin','Juil','Aout','Sept','Oct','Nov','Dec'];

var jour = dateProjet.getDay();

switch (jour){

case 0:

nomJour = "Dimanche";

break;

case 1:

nomJour = "Lundi";

break;

case 2:

nomJour = "Mardi";

break;

case 3:

nomJour = "Mercredi";

break;

case 4:

nomJour = "Jeudi";

break;

case 5:

nomJour = "Vendredi";

break;

case 6:

nomJour = "Samedi";

break;

}

return nomJour+ ' '+this.getDate() + ' '+ mois[this.getMonth()]+' '+ this.getFullYear();

}

}

var dateProjet = new MaDate();

console.log(dateProjet.getTime());

console.log(dateProjet.getDateFormatee());

</script>

</body>

</html>

A la découverte de jQuery

Apparue en janvier 2006, jQuery est une bibliothèque JavaScript qui figure aujourd’hui parmi les outils incontournables du web. Elle regroupe de multiples méthodes, éléments et propriétés qui viennent compléter ou remplacer habilement les fonctions natives du JavaScript. Ses atouts sont nombreux et jQuery vient en particulier moderniser les traitements du langage : vous devez impérativement la maitriser pour accélérer vos développements.

La naissance de comportements modernes ont vite rendu complexe la syntaxe de JavaScript. Les appels Ajax en constituent un singulier exemple, avec d’innombrables paramètres à injecter alors qu’il s’agit d’une tâche de plus en plus routinière.

JavaScript jouant un rôle majeur dans la montée en puissance des applications web, une solution robuste et fiable est imaginée, pour unifier et simplifier le code, tout en assurant un comptabilité maximale avec l’ensemble des navigateurs En Janvier 2006, la bibliothèque jQuery voit le jour.

JQuery est avant tout une bibliothèque JavaScript et le langage doit être pris en charge par le navigateur pour que jQuery soit opérationnel. Pour l’intégrer à vos projets, deux solutions :

* Télécharger les fichiers nécessaires sur le site officiel ([www.jquery.com](http://www.jquery.com))
* Appeler directement les fichiers nécessaires depuis le CDN officiel.

jQuery UI petite sœur de jQuery se spécialise dans le éléments d’interface.

jQuery Mobile pour le développement de site web responsive, dont l’affichage s’adapte à merveille aux petits écrans des smartphones et des tablettes.

<script src=<https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js>></script>

Proposition de recettes prêtes à l’emploi, pour gagner du temps.

**Sélecteurs d’éléments**

En JavaScript natif, vous sélectionnez un élément de la page et vous changez son contenu avec l’appel suivant.

<p id=*“intro”>Mon paragraphe</p>*

*<span class=”texte”>Voici mon texte</span>*

*<script>*

*document.getElementById(“intro”).innerHtml=”Bienvenue dans JavaScript”;*

*</script>*

En chargeant la bibliothèque jQuery, vous pouvez vous contenter de l’appel suivant :

$(*“p“).html(“Bienvenue dans jQuery“) ;*

$() est un alias de la fonction jQuery(), au cœur de la bibliothèque.

jQuery(*“p“).html(“Bienvenue dans jQuery“) ;*

Le paramètre passé à la fonction $() correspond à un sélecteur. $(*“p“) sélectionne tous les éléments <p> de la page.*

*Pour pointer vers un élément précisément, on emploie l’id.*

$(*“#intro“).html(“Bienvenue dans jQuery“) ;*

Comme en CSS, vous pouvez séparer les sélecteur par une virgule, afin d’appliquer le même traitement à plusieurs éléments à la fois

Pour pointer vers une classe spécifique.

$(*“.texte“).css(“font-size“, “16px“) ;*

L’appel de la fonction css() permet d’appliquer à la volée un autre style CSS. Le premier paramètre correspond à la règle concernée, le deuxième paramètre à la valeur à appliquer.

Utiliser les méthodes addClass() et removeClass() afin d’ajouter ou de retirer des classes.

<ol id*=“liste“>*

*<li>Maitriser les bases du JavaScript</li>*

*<li>Créer vos propres variables</li>*

*<li>Gérer les boucles conditionnelles</li>*

*</ol>*

*<style>*

*.vert {*

*background-color : #7ee97e ;*

*}*

*.jaune {*

*background-color : #e9e37e ;*

*}*

*.rouge {*

*background-color : #fd7059 ;*

*}*

</style>

<script>

$(document).ready(function(){ // vous avez l’assurance d’exécuter du code JavaScript dès que le document est prêt dans le navigateur du visiteur

$(*“#liste“).addClass(“vert“) ; // pointe vers une liste complète*

*$(“#liste > li“).addClass(“jaune“) ; //pointe vers ses enfants*

*$(“#liste  li:last“).hover(function(){ //pointe vers le dernier element enfant*

*$(this).addClass(“rouge“);*

*},function(){*

*$(this).removeClass(“rouge“) ;*

*}) ;*

*$(“#liste“).find(“li“).each(function(i){ // recherche tous les éléments <li> compris dans le bloc identifié par id=”liste”, puis leur applique individuellement la function définie entre accolades*

*i=i+1;*

$(this).append(*“—dossier--“); //ajoute du texte à la suite. prepend() ajoute un élément avant le sélecteur concerné.*

*});*

*});*

*</script>*

La méthode addClass nous permet d’ajouter une classe aux éléments considérés et ainsi de mettre en forme tous les éléments de manière dynamique

Dans cet exemple, un clic sur le bouton <Se connecter> laisse apparaitre une boite de dialogue et un message d’avertissement enchainés à l’aide de la méthode fadeIn().

<html>

<head>

<title>essai</title>

<script src=https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js></script>

</head>

<body>

<style>

#alerte {

background-color:#7ee97e;

font-weight:strong;

display: none;

}

#zone{

background-color:#e9e37e;

border-radius:5px;

padding:5px;

display:none;

width:340px;

}

</style>

<div id="zone">

<h3>Connectez-vous</h3>

Identifiant <input type="text"><br><br>

Mot de passe <input type="text">

</div>

<span id="alerte">Mot de passe oublié ? </span>

<button id="login">Se connecter</button>

<script>

$(document).ready(function(){

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

$("#zone").fadeIn(200, function(){

$("#alerte").fadeIn(600);

});

return false;

});

});

</script>

</body>

</html>

Lorsque l’on clique sur le bouton, les deux éléments suivants, masqués par défaut à l’aide de la propriété CSS display :none ;, apparaissent progressivement.

La méthode fadeIn() accepte en guise de paramètre une valeur correspondant à la durée de l’animation en millisecondes.

Divers avec la bibliotheque jQuery UI (<http://jqueryui.com/>)

**Menu accordéon**

<!doctype html>

<html lang="fr">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>jQuery UI Menu Accordeon - Fonctionnalité par défaut</title>

<link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/smoothness/jquery-ui.css"/>

<link rel="stylesheet" href="/resources/demos/style.css">

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>

<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<script>

$( function() {

$( "#accordion" ).accordion();

} );

</script>

</head>

<body>

<div id="accordion">

<h3>Section 1</h3>

<div>

<p>

Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer

ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit

amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut

odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

</p>

</div>

<h3>Section 2</h3>

<div>

<p>

Sed non urna. Donec et ante. Phasellus eu ligula. Vestibulum sit amet

purus. Vivamus hendrerit, dolor at aliquet laoreet, mauris turpis porttitor

velit, faucibus interdum tellus libero ac justo. Vivamus non quam. In

suscipit faucibus urna.

</p>

</div>

<h3>Section 3</h3>

<div>

<p>

Nam enim risus, molestie et, porta ac, aliquam ac, risus. Quisque lobortis.

Phasellus pellentesque purus in massa. Aenean in pede. Phasellus ac libero

ac tellus pellentesque semper. Sed ac felis. Sed commodo, magna quis

lacinia ornare, quam ante aliquam nisi, eu iaculis leo purus venenatis dui.

</p>

<ul>

<li>List item one</li>

<li>List item two</li>

<li>List item three</li>

</ul>

</div>

<h3>Section 4</h3>

<div>

<p>

Cras dictum. Pellentesque habitant morbi tristique senectus et netus

et malesuada fames ac turpis egestas. Vestibulum ante ipsum primis in

faucibus orci luctus et ultrices posuere cubilia Curae; Aenean lacinia

mauris vel est.

</p>

<p>

Suspendisse eu nisl. Nullam ut libero. Integer dignissim consequat lectus.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per

inceptos himenaeos.

</p>

</div>

</div>

</body>

</html>

**Onglets**

<!doctype html>

<html lang="fr">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>jQuery UI Menu Accordeon - Fonctionnalité par défaut</title>

<link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/smoothness/jquery-ui.css"/>

<link rel="stylesheet" href="/resources/demos/style.css">

<script src="https://code.jquery.com/jquery-1.12.4.js"></script>

<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>

<script>

$( function() {

$( "#onglets" ).tabs();

} );

</script>

</head>

<body>

<div id="onglets">

<ul>

<li><a href="#onglet-1">Section 1</a></li>

<li><a href="#onglet-2">Section 2</a></li>

<li><a href="#onglet-3">Section 3</a></li>

<li><a href="#onglet-4">Section 4</a></li>

</ul>

<div id="onglet-1">

<h3>Section 1</h3>

<div>

<p>

Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer

ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit

amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut

odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

</p>

</div>

</div>

<div id="onglet-2">

<h3>Section 2</h3>

<div>

<p>

Sed non urna. Donec et ante. Phasellus eu ligula. Vestibulum sit amet

purus. Vivamus hendrerit, dolor at aliquet laoreet, mauris turpis porttitor

velit, faucibus interdum tellus libero ac justo. Vivamus non quam. In

suscipit faucibus urna.

</p>

</div>

</div>

<div id="onglet-3">

<h3>Section 3</h3>

<div>

<p>

Nam enim risus, molestie et, porta ac, aliquam ac, risus. Quisque lobortis.

Phasellus pellentesque purus in massa. Aenean in pede. Phasellus ac libero

ac tellus pellentesque semper. Sed ac felis. Sed commodo, magna quis

lacinia ornare, quam ante aliquam nisi, eu iaculis leo purus venenatis dui.

</p>

<ul>

<li>List item one</li>

<li>List item two</li>

<li>List item three</li>

</ul>

</div>

</div>

<div id="onglet-4">

<h3>Section 4</h3>

<div>

<p>

Cras dictum. Pellentesque habitant morbi tristique senectus et netus

et malesuada fames ac turpis egestas. Vestibulum ante ipsum primis in

faucibus orci luctus et ultrices posuere cubilia Curae; Aenean lacinia

mauris vel est.

</p>

<p>

Suspendisse eu nisl. Nullam ut libero. Integer dignissim consequat lectus.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per

inceptos himenaeos.

</p>

</div>

</div>

</div>

</body>

</html>

Ajax pour plus d’intéraction

Offrez à vos utilisateurs des interactions plus dynamiques, avec des mises à jour de votre contenu qui s’effectuent en réaction à leurs différents clics : c’est le principe d’Ajax, un ensemble de technologies visant à paralléliser les traitements en évitant ainsi de réactualiser la page.

Signifiant Asynchronous JavaScript and XML, il s’agit d’un ensemble de technologies qui vont envoyer des requêtes http au serveur web avant d’utiliser JavaScript pour évaluer et afficher les données transmises en retour.

Le dialogue passe par différentes étapes incontournables :

* On crée tout d’abord une instance de l’objet XMLHttpRequest
* On prépare ensuite la requête à l’aide de la méthode open() de cet objet. Celle-ci accepte jusqu’à cinq paramètres, dont seuls les deux premiers sont obligatoires. Il s’agit du type de requête (GET ou POST principalement) puis de l’URL à laquelle on la soumet. Parmi les autres paramètres, on retrouve une valeur booléenne autorisant l’exécution d’une requête synchrone ainsi que d’éventuels identifiants et mots de passe si des mesures de sécurité ont été appliquées coté serveur.
* On envoie ensuite la requête avec la méthode send().
* Pour traiter la réponse du serveur, on utilise l’évènement readystatechange de l’objet XMLHttpRequest, qui se déclenche pour différents états de la requête.

Ainsi, les valeurs comprises entre 0 et 4 vont respectivement indiquer que :

0 -> la requête n’a encore été envoyée

1 -> la requête a été définie

2 -> la requête a été envoyée

3 -> la requête est en cours de traitement

4 -> la réponse a été reçue

* A travers une boucle conditionnelle, on associe ainsi un traitement à ce dernier cas de figure et on récupère les résultats à l’aide de la propriété reponseText de l’objetXMLHttpRequest.

Traitement d’une requête de type GET

Notre page web comporte un bouton et lorsque l’utilisateur clique dessus, la requête va charger le contenu d’un fichier texte. Son contenu va alors apparaitre dynamiquement dans une zone prévue à cet effet, sans la moindre réactualisation de la page. Seule la zone concernée sera mise à jour.

***On prépare un fichier ajax.txt avec une simple ligne de texte.***

*<!DOCTYPE html>*

*<html>*

*<head><title>essai</title></head>*

*<body>*

*<button id="envoi">Lire le fichier</button>*

*<div id="resultat"></div>*

*<script>*

*var element = document.getElementById("envoi");*

*element.addEventListener("click",function(){*

*var ajax = new XMLHttpRequest();*

*ajax.open("GET","ajax.txt");*

*ajax.addEventListener("readystatechange",function(){*

*if (ajax.readyState === XMLHttpRequest.DONE && ajax.status === 200){*

*document.getElementById("resultat").innerHTML = ajax.responseText;*

*}*

*});*

*ajax.send(null);*

*});*

*</script>*

*</body>*

*</html>*

On crée la variable element qui pointe vers le bouton. On lui associe l’évènement click. Lorsque l’utilisateur clique sur le bouton, on crée l’objet XMLHttpRequest ajax. On appelle ensuite la méthode open() avec les deux seuls paramètres obligatoires : le type de la méthode **GET** et l’URL à atteindre **ajax.txt**, situé dans le même dossier que le script.

On prépare en parallèle le gestionnaire de l’évènement *readystatechange* qui se déclenche tout au long du cycle de vie de la requête. Si la propriété *readyState* de l’objet XMLHttpRequest présente la valeur **DONE** (réponse reçue) avec une propriété *status* définie à 200 (tout s’est bien déroulé), on met à jour le contenu de la zone **resultat** avec le corps de la réponse. On affiche le contenu de la propriété *responseText*.

Dans l’acronyme AJAX, on mentionne le format XML. C’est le format privilégié pour récupérer et traiter les informations agencées de manière plus complexes.

**On prépare le fichier ajax.xml suivant :**

*<langages>*

*<langage>*

*<nom>JavaScript</nom>*

*<type>coté client</type>*

*</langage>*

*</langages>*

Dans l’évènement click on traite une réponse au format d’un arbre XML, c’est le rôle de la propriété responseXML.

*<!DOCTYPE html>*

*<html lang="fr">*

*<head>*

*<meta charset="UTF-8">*

*<title>essai</title></head>*

*<body>*

*<button id="envoi">Lire le fichier</button>*

*<div id="resultat"></div>*

*<script>*

*var element = document.getElementById("envoi");*

*element.addEventListener("click",function(){*

*var ajax = new XMLHttpRequest();*

*ajax.open("GET","ajax.xml");*

*ajax.addEventListener("readystatechange",function(){*

*if (ajax.readyState === XMLHttpRequest.DONE && ajax.status === 200){*

*var xml = ajax.responseXML;*

*var langage = xml.getElementsByTagName("langage")[0];*

*var nom, type;*

*for (var i=0 ; i<langage.childNodes.length;i++){*

*if (langage.childNodes[i].nodeName == "nom"){*

*nom= langage.childNodes[i].firstChild.nodeValue;*

*} else if ( langage.childNodes[i].nodeName == "type"){*

*type= langage.childNodes[i].firstChild.nodeValue;*

*}*

*}*

*document.getElementById("resultat").innerHTML = "Langage : " + nom + " Exécution : " + type;*

*}*

*});*

*ajax.send(null);*

*});*

*</script>*

*</body>*

*</html>*

Le format JSON (JavaScript Object Notification) constitue une troisième voie extrêmement populaire. **Créons le fichier ajax.json suivant :**

*{*

*"langage":"JavaScript",*

*"type":"coté client"*

*}*

Nous avons besoin de la propriété responseText pour traiter cette structure. Nous allons utiliser la méthode native JSON.parse() afin de sérialiser les données et d’y accéder de manière intuitive, à travers la notation point.

*<!DOCTYPE html>*

*<html lang="fr">*

*<head>*

*<meta charset="UTF-8">*

*<title>essai</title></head>*

*<body>*

*<button id="envoi">Lire le fichier</button>*

*<div id="resultat"></div>*

*<script>*

*var element = document.getElementById("envoi");*

*element.addEventListener("click",function(){*

*var ajax = new XMLHttpRequest();*

*ajax.open("GET","ajax.json");*

*ajax.addEventListener("readystatechange",function(){*

*if (ajax.readyState === XMLHttpRequest.DONE && ajax.status === 200){*

*var json = ajax.responseText;*

*var obj = JSON.parse(json);*

*var langage = obj.langage;*

*var type = obj.type;*

*document.getElementById("resultat").innerHTML = "Langage : " + langage + " Exécution : " + type;*

*}*

*});*

*ajax.send(null);*

*});*

*</script>*

*</body>*

*</html>*

Traitement d’une requête de type POST

On utilise la saisie de l’utilisateur pour valider un champ. On vient récupérer le paramètre qu’on lui envoie et qui renvoie une chaine mise en forme.

**On prépare le fichier ajax.php suivant :**

… pb d’exécution du code