**Commandes en JavaScript – Web**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Exemple | Utilisation | |
| Commandes |  |  |  |  | |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Objets |  |  |  |  |  |
| document. | Objet "document" faisant référence au nœud <html> racine de la page Web. |  | document.body; |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Propriétés |  |  |  |  |  |
| .nodeType | Propriétés de tous les objets DOM dont la valeur dépend du type du nœud auquel elle appartient  Nœud de type élément  Nœud de type textuel | ELEMENT\_NODE  TEXT\_NODE |  | if (document.body.nodeType === Node.ELEMENT\_NODE) {  console.log("Body est un noeud de type élément");  } else {  console.log("Body est un noeud de type textuel");  } | détermination du type de noeud |
| .nodeName | renvoi le nom du nœud (de l'élément) | monElt.nodeName.toLowerCase(); |  |  | Remarque : il est conseillé de le couplé à .toLower/UpperCase() |
| .hasChildNodes() | vérifie la présence d'éléments enfants   * renvoie true si enfant(s) * renvoie false si aucun enfant | monElt.hasChildNodes(); |  |  |  |
| .parentNode | Renvoi l'objet : nœud parent | monElt.parentNode; |  |  |  |
| .childNodes | Contient la liste ordonnée de tous les enfants directs d'un nœud.  ATTENTION : après l'ouverture d'une balise un espace ou un retour à la ligne est considéré comme du texte => premier nœud fils de type texte…  ATTENTION : avec un tableau  accès aux enfants seulement après la commende .childNodes[1]  pour lignes : 1er elt (i = 0) = ligne, 2e = text, 3e ligne, …  pour colonnes : normal (cad 1er elt (j = 0) = text, 2e = td, 3e = text, …) |  | <table>  <tr>  <td></td>  <td></td>  </tr>  <tr>  <td></td>  <td></td>  </tr>  </table> | nœud.childNodes.length  nœud.childNodes[i]  for (var i = 0; i < document.body.childNodes.length; i++) {  console.log(document.body.childNodes[i]);  }  var tableElt = document.getQuerySelector("table");  tableElt.childNodes[1].childNodes[i];  tableElt.childNodes[1].childNodes[i].childNodes[j]; | (comme un tableau)  nombre de ses enfants directs  accès à un de ses objets enfants  parcourir la totalité de ses enfants les uns après les autres  ATTENTION avec un tableau :  accède à la ie ligne du tableau en commençant par 0  (0 = 1re ligne, 1 = texte, 2 = 2e ligne, …)  accède à la je colonne de la ie ligne en commençant par 1  (0 = texte, 1 = 1re td, 2 = texte, 3 = 2e td, …) |
| .firstChild  .lastChild | renvoi le 1er / dernier enfant de l'élément (souvent un nœud de type texte) | monElt.firstChild; |  |  |  |
| .firstElementChild  .lastElementChild | renvoi le 1er / dernier enfant de l'élément qui est de type élément (pas type texte) |  |  |  |  |
| .nextSibling  .previousSibling | renvoi l'élément suivant / précédent de l'élément (frère) |  |  |  |  |
| .nextElementSibling  .previousElementSibling | idem en ne ciblant que les nœuds de type éléments (pas texte) |  |  |  |  |
|  |  |  |  |  |  |
| .innerHTML | Donne le contenu HTML de tous les descendants de l'objet | object.innerHTML; |  |  |  |
| .textContent | Donne le contenu textuel de tous les descendants de l'objet  (comme .innerHTML sans les balises) |  |  |  |  |
|  |  |  |  |  |  |
| Cas particuliers |  |  |  |  |  |
| .disabled | donné en attribut à une balise HTML rendra cet élément inaccessible à l'utilisateur bien que toujours affiché (en transparence) | monElt.disabled; |  | monElt.disabled = true;  monElt.disabled = false; | donne l'attribut disabled à l'élément  supprime l'attribut disabled à l'élément |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Méthodes |  |  |  |  |  |
| .getElementsByTagName( ) | Renvoi sous forme de liste tous les éléments enfants (direct et indirect) de l'objet ayant pour balise celle placée en paramètre | var listeElts = object.getElementsByTagName("balise"); | var titresElts= document.getElementsByTagName("h2");  console.log(titreElts[0]);  console.log(titresElts.length); |  |  |
| .getElementsByClassName( ) | Renvoi sous forme de liste tous les éléments enfants (direct et indirect) de l'objet ayant la classe donnée en paramètre |  |  |  |  |
| .getElementById | Renvoi l'élément enfant (direct ou indirect) de l'objet ayant l'ID donné en paramètre |  |  |  |  |
| .querySelectorAll( ) | Idem en utilisant les sélecteurs comme en CSS pour sélectionner les éléments |  | document.querySelectorAll("#antique .existe"); |  |  |
| .querySelector( ) | Idem, mais ne renvoi qu'un seul élément : le premier de la page qui correspond à la demande. |  |  |  |  |
| information |  |  |  |  |  |
| .getAttribute( ) | Renvoie la valeur de l'attribut donné en paramètre |  | console.log(document.querySelector("a").getAttribute("href")); |  |  |
| .id  .href  .value | la valeurs de ses attributs peuvent s'obtenir directement (sans passer par la prop. précédente) |  |  |  |  |
| .classList | Donne sous forme d'un tableau la valeur de toutes les classes de l'objet |  |  |  |  |
| .htmlFor | donne la valeur de l'attribut for utilisé dans les <Label> |  |  |  | Remarque : on n'utilise pas "for" car ce mot est réservé en JS |
| .hasAttribute( ) | Vérifie la présence de l'attribut donné en paramètre dans l'objet. |  |  | if (document.querySelector("a").hasAttribute("target")) {  console.log("Le premier lien possède l'attribut target");  } else {  console.log("Le premier lien ne possède pas l'attribut target");  } |  |
| .contains( ) | permet de tester la présence d'une valeur dans un tableau |  |  | if (document.getElementById("antiques").classList.contains("merveille")) {  console.log("L'élément identifié par antiques possède la classe merveille");  } else {  console.log("L'élément identifié par antiques ne possède pas la classe merveille");  } | associé à la propriété .classList , cette méthode permet de déterminer si l'objet contient une classe donnée. |
| getComputedStyle( ) | renvoi le style CSS de l'élément placé en paramètre  ATTENTION : il s'agit d'une fonction et non pas une méthode ! | getComputedStyle(eltSelected); | var stylePara = getComputedStyle(document.getElementByTagName("p");  console.log(stylePara.fontStyle);  console.log(stylePara.color); |  |  |
|  |  |  |  |  |  |
| Modification du contenu HTML |  |  |  |  |  |
| .innerHTML += | Rajoute un élément au contenu HTML d'un objet |  | document.getElementById("IdDeObjetAModifier").innerHTML += '<p>Texte rajouté</p>' | document.getElementById("langages").innerHTML += '<li id="français">Français</li>'; | ATTENTION : ici on utilise l'apostrophe et non le " qui sera réservé pour du texte nécessitant des guillemets en HTML |
| .innerHTML = | Remplace le code HTML de l'élément par un autre (ou le supprime si = " " ) |  |  |  |  |
| .textContent + = | Idem, mais modifie uniquement le contenu textuel |  |  |  |  |
| .setAttribute( , ) | Crée ou modifie la valeur d'un attribut | .setAttribute("nomAttribut", "valeurAttribut"); |  |  |  |
| .id= " "  .href= " "  .value= " " | Idem pour les attributs possédent leur propriété : .id, .href, .value | .id = "valeurID"; |  |  |  |
| .classList.add( ) | ajouter une classe à un élément | .classList.add("valeurNouvelleClasse"); |  |  |  |
| .classList.remove( ) | supprimer une classe de l'élément |  |  |  |  |
| .htmlFor | modifier la valeur de l'attribut for utilisé dans les <Label> |  |  |  | Remarque : on n'utilise pas "for" car ce mot est réservé en JS |
| ajout d'élément |  |  |  |  |  |
| .createElement( ) | créer un nouvel élément dans la page avec la balise HTML donné en paramètre | var newElt = document.createElement("balise"); |  | newElt.textContent = "Texte du nouvel élément";  newElt.id = "valeurID";  newElt.setAttribute("nomAttr", "valeurAttr");  newElt.classList.add("valeurClasse"); | une fois le nouvel élément créé, on lui donne des valeurs grace aux propriétés/méthodes vu au dessus. |
| .createTextNode( ) | créer un nouvel élément de type textuel (qui doit donc être placé à l'intérieur d'un élément de type bloc) | var newText = document.createTextNode("Texte à ajouter"); |  | eltParentSelected.appendChild(document.createTextNode("Texte à ajouter")); | ajoute un nœud de type textuel à la suite d'un élément existant en une seule ligne de code. |
| .cloneNode( ) | clone = fait la copie de l'élément (sans l'insérer dans la page)  avec ses descendants  sans ses descendants | monElt.cloneNode(true);  monElt.cloneNode(false); |  |  | ATTENTION : le clonage ne copie pas les gestionnaires d'événements pouvant être liées à l'élément à copier |
| .appendChild( ) | ajoute un nouvel élément (placé en paramètre) comme dernier enfant à l'élément sur lequel est appliquée la méthode | eltParentSelectionné.appendChild(newElt); |  |  |  |
| .insertBefore( , ) | ajoute un nouvel élément (paramètre 1) juste avant l'élément frère placé en paramètre 2 (et comme enfant de l'élément sur lequel est appliqué la méthode) | eltParentSelected.insertBefore(newElt, eltFrereSuivantSelected); | document.getElementById("langages").insertBefore(rubyElt, document.getElementById("php"); |  |  |
| .insertAdjacentHTML( , ) | ajoute une chaine de caractère (param. 2) à un endroit spécifique (param. 1) par rapport à l'élément sur lequel elle est appliquée  avant l'elt lui-même  dans l'elt, avant son 1er enfant  dans l'elt, après sont dernier enfant  après l'elt lui-même | eltSelected.insertAdjacentHTML("position", '<balise attributs… >Texte à ajouter</balise>');  "beforebegin",  "afterbegin",  "beforeend",  "afterend", | eltSelected.insertAdjacentHTML("afterend", '<p id="valeurID">Texte à ajouter<p>'); |  |  |
| .replaceChild( , ) | remplace un élément enfant (param. 2) par un nouvel élément (param. 1) quand appliqué sur l'élément parent | eltParentSelected.replaceChild(newElt, eltEnfantSelectedARemplacer); |  |  |  |
| .removeChild( ) | fait disparaitre le nœud enfant donné en paramètre (ainsi que ses descendants) | eltParentSelected.removeChild(eltEnfantSelectedASuppr); |  |  | Remarque : retire l'élément de la page, mais ne le détruit pas. Il reste manipulable et ré-insérable. |
| .style | modifie le style d'un élément  ATTENTION :  pour les propriétés de style composés :   * pas de tiret * 1re lettre 2e mot en Maj.   (norme camelCase) | eltSelected.style.propStyle = "valeurProp"; | var pElt = document.querySelector("p");  pElt.style.color = "red";  pElt.style.margin = "50px";  pElt.style.fontFamily = "Arial";  pElt.style.backgroundColor = "black"; |  |  |
|  |  |  |  |  |  |
| Programmation évènementielle |  |  |  |  |  |
| .addEventListener( , ) | ajoute un gestionnaire d'évènement à un élément de la page pour y réagir suite à l'action de l'utilisateur  (en mode Bouillonnement par défaut)  en mode Capture | monElt.addEventListener("nomEvent", maFonction);  monElt.addEventListener("nomEvent", maFonction, true); |  | monElt.addEventListener("nomEvent", function( ) {  instructions…  }); | cas avec une fonction anonyme  en mode Bouillonnement :  eltCible > div > body > document  en mode Capture :  document > body > div > eltCible |
| .removeEventListener( , ) | supprime le gestionnaire d'évènement  (exactement même syntaxe que précédent) | monElt.removeEventListener("nomEvent", maFonction); |  |  | cette méthode ne peut pas supprimer un gestionnaire avec une fonction anonyme |
| .stopPropagation( ) | arrête la propagation d'un évènement aux éléments parents  (appliquée sur l'objet Event dans la fonction du gestionnaire d'évènement sans paramètre) | function maFonction {  instructions  e.stopPropagation();  } |  |  |  |
| .preventDefault( ) | arrête le comportement par défaut de certains évènements (ex. : clic sur lien, clic droit de souris, …)  (même syntaxe que précédent) | function maFonction {  instructions  e.preventDefault();  } |  |  |  |
| event | un objet Event est automatiquement créé à chaque action de l'utilisateur sur un élément de la page |  |  |  |  |
| .type | propriété associée à tout objet Event et dont la valeur indique le type d'évènement |  |  |  |  |
| .target | propriété associée à tout objet Event et dont la valeur indique l'élément de la page ciblé par l'évènement |  |  |  | un gestionnaire d'événement placé sur un élément parent va être répercuté sur les enfants.  Or si on souhaite avoir en référence seulement l'élément parent, il faudra utilisé .currentTarget plutôt que .target |
| .currentTarget | propriété associé au premier objet ayant déclenché l'évènement jusqu'à ce que celui-ci soit terminé |  | *voir exemple-web2 : exemple 1* |  |
| Evt clavier |  |  |  |  |  |
| "keypress" | appuis sur une touche alphanumérique du clavier  prop. stockant la touche en Unicode | .charCode |  |  |  |
| String.fromCharCode( ) | méthode traduisant une valeur Unicode en son caractère associé | String.fromCharCode(valeur); | String.fromCharCode(84, 101, 115, 116); => affiche "Test" | String.fromCharCode(event.charCode); | obtenir le caractère de la touche pressée par l'utilisateur |
| "keydown" | pression sur n'importe qu'elle touche du clavier  prop. stockant la touche codé | .keyCode |  |  | pas codé en Unicode |
| "keyup" | relachement de n'importe qu'elle touche du clavier  prop. stockant la touche en codé | .keyCode |  |  | pas codé en Unicode |
| Evt souris |  |  |  |  |  |
| "click" | appuis sur l'un des boutons de la souris  bouton utilisé pour le clic  coordonnées du clic par rapport  au coin sup. gauche de la page Web  coin sup. g. de la portion de page visible | .button  .pageX .pageY  .clientX .clientY |  | *Demande confirmation avant de suivre un lien :*  document.getElementById("monLien").addEventListener("click", function (e) {  if (!confirm("Etes-vous sûr de vouloir suivre ce lien ?"))  e.preventDefault();  }); | valeurs possible pour .button :  0 => gauche  1 => milieu  2 => droit |
| "dblclick" | double clic |  |  |  |  |
| "mousedown" | pression sur un bouton de la souris |  |  |  | mêmes propriétés que "click" |
| "mouseup" | relâchement d'un bouton de la souris |  |  |  | mêmes propriétés que "click" |
| "mouseover" | la souris entre sur l'élément |  |  |  |  |
| "mousemove" | déplacement du curseur sur l'élément |  |  |  |  |
| "mouseout" | la souris sort de l'élément |  |  |  |  |
| .relatedTarget | permet de connaitre l'élément lié à la fin d'un événement de type "mousemove" ou "mouseout" |  | *voir exemple-web2 : exemple 2* |  |  |
| Evt fenêtre |  |  |  |  |  |
| "load" | détecte la fin du chargement de la page | window.addEventListener("load", function () {  instruction  } |  |  |  |
| "beforeunload" | detecte la fermeture de la page | window.addEventListener("beforeunload", function () {  instruction  } |  | window.addEventListener("beforeunload", function () {  var message = "On est bien ici !";  e.returnValue = message;  return message;  }); | Affiche un message usuel de demande de confirmation avant de fermer la fenêtre |
|  |  |  |  |  |  |
| Formulaires |  |  |  |  |  |
| form.elements | contient la liste de tous les éléments (nœuds) du formulaire |  |  |  |  |
| "submit" | événement créé à la suite de l'appui sur le bouton "submit" du questionnaire  (il permet de vérifier les données avant leur transmission vers la page de traitement) | form.addeventListener("submit", function () {  instructions  }); |  |  | ATTENTION : l'évènement "submit" doit être utilisé sur l'élément formulaire et non pas sur le bouton submit lui-même !!! |
| "reset" | créé lors de la réinitialisation du formulaire |  |  |  |  |
| .submit()  .reset() | méthodes permettant l'envoi des données du formulaire / son réinitialisation via JS |  |  |  | ATTENTION : l'utilisation de ses méthodes ne déclenche pas les événements "submit" et "reset" |
|  |  |  |  |  |  |
| zone de texte |  |  |  |  |  |
| .value | acceder au contenu d'un champ de texte |  |  |  |  |
| "focus" | evénement crée lors de la selection d'une zone de texte pour la saisie |  | <p>  <label for="pseudo">Pseudo</label>  <input type="text" name="pseudo" id="pseudo" required />  <span id="aidePseudo"></span>  </p> | pseudoElt.addEventListener("focus", function () { document.getElementById("aidePseudo").textContent = "Entrez votre pseudo";  }); | affiche un message contextuel pour donner une indication sur la saisie  (code HTML associé est donné en exemple case de gauche pour la balise <span>) |
| "blur" | évènement crée lors de la dé-selection de la zone de texte (clic sur une autre zone de texte ou ailleur sur la page) |  |  | pseudoElt.addEventListener("blur", function (e) { document.getElementById("aidePseudo").textContent = "";  }); | supprime le message contextuel lorsque la zone de texte en question n'est plus sélectionnée |
| .focus( ) | permet de présélectionner un élément | monElt.focus( ); |  |  |  |
| .blur( ) | permet de pré-désélectionner un élément | monElt.blur( ); |  |  |  |
| Elt d'options |  |  |  |  |  |
| "change" | évènement associé au changement intervenu sur une case à cocher ou un élément de bouton radio |  |  |  |  |
| .currentTarget.checked | propriété de l'évènement change  pouvant prendre 2 valeurs  case venant d'être cochée  case venant d'être dé-cochée | True  False |  |  |  |
| .target.value | pour un groupe de bouton radio,  cette propriété associé à change  permet d'obtenir la valeur sélectionnée |  |  | var aboElts = document.getElementsByName("abonnement");  for (var i = 0; i < aboElts.length; i++) {  aboElts[i].addEventListener("change", function (e) {  console.log("Formule d'abonnement choisie : " + e.target.value);  });  } | attribue un gestionnaire d'évènement à chaque bouton radio pour indiquer lequel a été selectionné en dernier |
| .target.value | Idem pour une liste déroulante, |  |  | document.getElementById("nationalite").addEventListener("change", function (e) {  console.log("Code de nationalité : " + e.target.value);  }); | idem mais avec un code plus simple dans le cas d'une liste déroulante (puisque ces éléments ne sont pas indépendants !) |
| Validation | en cours de frappe |  |  |  |  |
| "input" | évènement créé à chaque modification de la valeur d'un champ de type <input > |  |  | document.getElementById("mdp").addEventListener("input", function (e) {  var mdp = e.target.value;  var longueurMdp = "faible";  var couleurMsg = "red";  if (mdp.length >= 8) {  longueurMdp = "suffisante";  couleurMsg = "green";  } else if (mdp.length >= 4) {  longueurMdp = "moyenne";  couleurMsg = "orange";  }  var aideMdpElt = document.getElementById("aideMdp");  aideMdpElt.textContent = "Longueur : " + longueurMdp;  aideMdpElt.style.color = couleurMsg;  }); | contrôle la longueur d'un mot de passe pendant sa saisie |
|  | après la saisie |  |  |  |  |
| "blur" | utiliser l'évènement blur pour controler la valeur à la fin de la saisie |  |  |  |  |
| .indexOf( ) | méthode permettant de chercher une valeur (chaine de caractère donnée en paramètre) dans une chaine de caractère |  | if (e.target.value.indexOf("@") === -1) {  instruction  } |  | si valeur trouvée dans la chaine => 0  si valeur absente de la chaine => -1 |
|  | Exrpession régulière |  |  |  |  |
| / / | création d'une expression régulière, cad un motif destiner à être comparé avec des chaines de caractères | var regex = /valeurConstante/;  regex.test("chaineATester"); | var regexMail = /@/;  regexMail.test("alex@sfr.fr"); |  | Remarque :  renvoie true si motif retrouvé  renvoie false si motif absent |
|  |  |  |  |  |  |
| Upload |  |  |  |  |  |
| files | tableau de fichiers soumis par le client via <input type="file"> | document.getElementById("loadFile").files[0] |  |  |  |
| Blob  File | objets de références stockant les données des fichiers soumis sous forme binaire  nom du fichier  date de dernière modif | var monFichier = files[2];  monFichier.name  monFichier.lastModifiedDate |  |  |  |
| FileReader | objet possédents plusieurs prop et méthodes de traitement des fichiers  (nécessite une instanciation)  données converties en base64 et accessibles via une URL  données stockées dans une chaine | var reader = new FileReader();  reader.readAsDataURL(monFichier);  reader.readAsText(monFichier); |  | reader.readAsText(file, "UTF-8");  "load" | 2e méthode prend en param. facultatif l'encodage à utiliser  ces 2 méthodes sont asynchrones et sont ponctués par des événements, dont "load" déclenché à la fin de la lecture si sans erreur |
| .result | prop permettant de récupérer le résultat des méthodes précédentes | reader.result; |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| Animation |  |  |  |  |  |
| setInterval( , ) | fonction permettant d'appeler une fonction (param1) à intervalle de temps régulier (param2) donné en ms  param3,4,… sont facultatif et seront les param.1,2,… de maFonction | setInterval(maFonction, 1000);  setInterval(maFonction, 1000, param3, param4); |  | function afficherTemps() {  var time = Number(timeElt.textContent);  timeElt.textContent = time + 1;  }  …  var nbClicBouton = 0;  var inervalId = null;  boutonElt.addEventListener("click", function () {  nbClicBouton++;  if (nbClicBouton % 2 === 1) {  boutonElt.textContent = "Arrêter";  intervalId = setInterval(afficherTemps, 1000);  } else {  clearInterval(intervalId);  boutonElt.textContent = "Démarer";  }  }); |  |
| clearInterval( ) | fonction permettant d'interrompre la répétions induite par setInterval( , ) | var intervalId = setInterval(maFonction, 1000);  if (…) {  clearInterval(intervalId);  } |  | si setInterval( , ) peut être appelée plusieurs fois (comme ici avec l'arrêt et le redémarrage du chrono au clic), il faut créer la variable intervalId avant et non pas au moment d'appel de setInterval( , ) afin qu'elle ne soit pas considérée comme une variable anonyme valable uniquement à l'intérieur de la fonction |
| setTimeout( , ) | fonction permettant d'appeler une fonction (param1) une seule fois après un certain délai (param2) donné en ms  si maFonction à besoin de param., on les précises en param3,4,… (facultatifs) | setTimeout(maFonction, 1000);  setTimeout(maFonction, 1000, param3, param4); |  |  |  |
| clearTimeout( ) | fonction permettant d'interrompre la fonction setTimout( , ) avant qu'elle ne soit exécutée | var timeoutId = setTimeout(maFonction, 1000);  if (…) {  clearTimeout(timeoutId);  } |  |  |  |
| requestAnimationFrame( ) | fonction permettant d'appeler une fonction en paramètre pour créer une animation  (elle optimise l'affichage pour rendre l'animation la plus fluide possible) | function animer () {  instructions;  requestAnimationFrame(animer);  }  requestAnimationFrame(animer); |  |  |  |
| parseFloat( ) | fonction permettant de convertir la valeur d'une propriété (not. CSS) de type XXpx en nombre de type XX |  |  | var xBloc = parseFloat(getComputedStyle(divElt).left);  divElt.style.left = (xBloc + 10) + "px"; | xBloc = valeur de type nombre de la position relative (left) de Div  attribution d'une nouvelle position de Div en l'augmentant de 10 px |
| cancelAnimationFrame( ) | fonction permettant de stopper une animation induite par requestAnimationFrame( ) | Idem à clearInterval( ) |  |  |  |
|  |  |  |  |  |  |
| Redirection |  |  |  |  |  |
| window.location | redirection vers une autre page avec modification de l'URL dans le navigateur | monElt.addEventListener('click', function() {  window.location = 'http://monurl.com';  }); |  |  |  |
|  |  |  |  |  |  |
| iFrame |  |  |  |  |  |
| .contentDocument | permet d'accéder au contenue d'une iframe | var frame = document.getElementbyId("maFrame").contentDocument |  |  | les éléments de la frame sont alors accessibles avec les prop. habituelles du DOM |
| "load" | événement créé lorsque la frame est complètement chargée |  |  |  |  |
| window.top | pointe vers la fenêtre mère  utilisé dans le script d'une frame pour accéder à une fonction du script de la page principale | window.top.window.maFonction(); |  |  |  |
|  |  |  |  |  |  |
| AJAX |  |  |  |  |  |
| XMLHttpRequest | objet permettant de faire une demande au serveur et de stocker sa réponse | var req = new XMLHttpRequest(); |  |  |  |
| .open( , , ) | méthode associé à l'objet précédent permettant de configurer la requête  param1 : méthode http (GET, POST, PUT, …)  param2 : URL cible  *param3 : false = synchrone*  *true = asynchrone (défaut)*  *param4 : nom utilisateur*  *param5 : mdp* | XMLHttpRequest.open("GET", "http://localhost/…", false);  XMLHttpRequest.open("GET", "http:// …", false, "user", "mdp"); |  |  | les 3 derniers param sont facultatif :  par défaut = asynchrone  param 4 et 5 en cas d'identification nécessaire sur le serveur |
| .send( ) | méthode associé à l'objet précédent permettant l'envoi de la requête au serveur  si méthode HTML = GET  = POST ou PUT | XMLHttpRequest.send(null);  XMLHttpRequest.send(contenu); |  |  |  |
| .reponseText | prop contenant sous forme textuelle la réponse renvoyé par le serveur | XMLHttpRequest.reponseText; |  |  |  |
| .status | prop contenant le Code/Statut HTML de la réponse (2xx et 3xx = succès, 4xx et 5xx = échec) |  |  | req.addEventListener("load", function () {  if (req.status >= 200 && req.status < 400) {  console.log(req.responseText);  } else {  console.error(req.status + " " + req.statusText);  }  }); | gère les erreurs de traitement de la requête (ressource non trouvée sur le serveur, pb interne au serveur, …) |
| .statusText | prop contenant le message associé à cette réponse (OK, Not Found, …) |  |  |
| .upload | prop. donnant accés à plusieurs événements d'écrivant l'état de transmission des données (utilisé not quand données de taille importante (image, …) |  |  |  |  |
| "progress" | évènement accessible depuis .uptoad et contenant 2 propriétés :  nb d'octets transférés  nb total d'octet à transférer | xhr.upload.addEventListener("progress", function(e) {  e.loaded;  e.total; |  |  | à utiliser par exemple avec la balise <progress> |
| "error" | évènement créé sur l'objet requête lorsque celle-ci n'a pas réussi à atteindre le serveur (pb de réseau, nom du serveur incorect, …) |  |  | req.addEventListener("error", function () {  console.error("Erreur réseau");  }); | gère lorsque les erreurs lorsque la requête n'a pas réussi à atteindre le serveur |
|  | fonction AJAX générique  AJAX GET : fonction servant à créer et soumettre une requête  Param. 1 : URL  Param. 2 : fonction pour traiter la réponse | function ajaxGet(url, callback) {  var req = new XMLHttpRequest();  req.open("GET", url);  req.addEventListener("load", function () {  if (req.status >= 200 && req.status < 400) {  callback(req.responseText);  } else {  console.error(req.status + " " + req.statusText + " : " + url);  }  });  req.addEventListener("error", function () {  console.error("Erreur réseau avec l'URL : " + url);  });  req.send(null);  } |  | Code Source page HTML :  <script src="../js/ajax.js></script>  <script src="…  </body>  2 possibilités pour la 2e fonction :   * fonction classique   ajaxGet("http://localhost/…", maFonction);   * fonction anonyme   ajaxGet("http://localhost/…", function (reponse) {  instructions;  }); | il est possible d'isoler cette fonction dans un fichier .js pour qu'elle serve à plusieurs scripts sans avoir à réécrire son code dans chacun d'eux.  La seule condition est de faire appel à ce script avant tous les autres dans le code source de la page HTML |
| JSON.parse( ) | permet de passer  chaine de caractère 🡪 format JSON | JSON.parse(maChaine); |  |  |  |
| JSON.stringify() | permet de passer  format JSON 🡪 chaine de caractère (JSON)  objet JS 🡪 chaine de caractère (JSON) | JSON.sringify(monObjetJson); |  |  |  |
| FormData | objet (constructeur) permettant la mise en forme des données à envoyer vers un serveur (données de formulaire, …) | var monFormulaire = new FormData(); | var identite = new FormData();  //Ajout d'info dans l'objet  identite.append("login", "Bob");  identite.append("password", "azerty"); | var form = document.querySelector("form");  var data = new FormData(form); | récupère toutes les données d'un formulaire en une seule étape ! |
| .append( , ) | méthode associée à l'objet FormData permettant d'y ajouter des couple nom-valeur  Param. 1 : nom  Param. 2 : valeur | monFormulaire.append("nom", "valeur"); |  |  |
|  | fonction AJAX générique  AJAX POST : fonction servant à créer et à envoyer des données vers un serveur  Param. 1 : URL  Param. 2 : données à envoyer (objet FormData)  Param. 3 : fonction pour traiter la réponse  Param. 4 : sert à traiter l'envoi de donnée au format JSON   * true si donnée au format JSON * rien sinon | function ajaxPost(url, data, callback, isJson) {  var req = new XMLHttpRequest();  req.open("POST", url);  req.addEventListener("load", function () {  if (req.status >= 200 && req.status < 400) {  callback(req.responseText);  } else {  console.error(req.status + " " + req.statusText + " : " + url);  }  });  req.addEventListener("error", function () {  console.error("Erreur réseau avec l'URL : " + url);  });  if (isJson) {  req.setRequestHeader("Content-Type", "application/json");  data = JSON.stringify(data);  }  req.send(data);  } |  | idem à AJAX GET  3e possibilité pour la fonction :   * pas de fonction pour la réponse   ajaxPost("http://localhost/…", data, function () {}); | Idem à AJAX GET |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |