UT4. Introducció a Javascript

UT4. Introducció a Javascript

4.7 Introducció al DOM
- 4.7.1. Esdeveniments
- 4.7.2. Objecte Window
- 4.7.3. Objecte document
- 4.7.4. Objecte Navigator
- 4.7.5. Accés al DOM
- 4.7.6. Formularis HTML i el DOM
- 4.7.7. Maneig d'errors

4.7 Introducció al DOM

4.7.1. Esdeveniments

Els esdeveniments de Javascript permeten la interacció entre les aplicacions Javascript i els usuaris.

Per exemple:

- Cada vegada que es prem un botó, es produeix un esdeveniment,
- Cada vegada que es prem una tecla també es produeix un esdeveniment.

També hem de saber que, perquè es produeixi un esdeveniment no és obligatori que intervingui l'usuari, ja que per exemple, cada vegada que es carrega una pàgina, també es produeix un esdeveniment.

La versió 3 del DOM inclou l'especificació completa dels esdeveniments Javascript.

4.7 Introducció al DOM

Classes d'esdeveniments

Cada element HTML té definida la seva pròpia llista de possibles esdeveniments (events en anglès)

El nom dels esdeveniments es construeix mitjançant el prefix "**on**", seguit del nom en anglès de l'acció associada a l'esdeveniment.

Per exemple, a l'esdeveniment de clicar damunt un element amb el ratolí, s'anomena "**onclick**", o l'esdeveniment associat a l'acció de moure el ratolí s'anomena "onmousemove".

4.7 Introducció al DOM

La següent llista resumeix els esdeveniments més importants definits per Javascript:

Tipus d'esdeveniment	Nom	Descripció
Relacionats amb el ratolí	onclick	Click sobre un element
	ondblclick	Doble click sobre un element
	onmousedown	Se prem un botó del ratolí sobre un element
	onmouseenter	El punter del ratolí entra a l'àrea d'un element
	onmouseleave	El punter del ratolí surt de l'àrea d'un element
	onmousemove	El punter del ratolí s'està movent sobre l'àrea d'un element
	onmouseover	El punter del ratolí se situa damunt de l'àrea d'un element
	onmouseout	El punter del ratolí surt fora de l'àrea de l'element o fora d'un dels seus fills
	onmouseup	Un botó del ratolí s'allibera estant sobre un element
	contextmenu	Se prem el botó dret del ratolí (abans que aparegui el menú contextual)

4.7 Introducció al DOM

Relacionats amb el teclat	onkeydown	L'usuari té polsada una tecla (per elements de formulari i body)
	onkeypress	L'usuari prem una tecla (moment just en que es prem) (per elements de formulari i body)
	onkeyup	L'usuari allibera una tecla que tenia polsada (per elements de formulari i body)
Relacionats amb formularis	onfocus	Un element del formulari pren el focus
	onblur	Un element del formulari perd el focus
	onchange	Un element del formulari canvia
	onselect	L'usuari selecciona el text d'un element input o textarea
	onsubmit	Se prem el botó d'enviament del formulari (abans de l'enviament)
	onreset	Se prem el botó reset del formulari
Relacionats amb finestres o frames	onload	S'ha completat la càrrega de la finestra
	onunload	L'usuari ha tancat la finestra
	onresize	L'usuari ha canviat la grandària de la finestra

4.7 Introducció al DOM

Relacionats amb animacions i transicions	animationend, animationiteration, animationstart, beginEvent, endEvent, repeatEvent, transitionend
Relacionats amb la bateria	chargingchange, chargingtimechange, dischargingtimechange, levelchange
Relacionats amb cridades de telefonia	alerting, busy, callschanged, connected, connecting, dialing, disconnected, disconnecting, error, held, holding, incoming, resuming, statechange
Relacionats amb canvis al DOM	DOMAttrModified, DOMCharacterDataModified, DOMContentLoaded, DOMElementNameChanged, DOMNodeInserted, DOMNodeInsertedIntoDocument, DOMNodeRemoved, DOMNodeRemovedFromDocument, DOMSubtreeModified
Relacionats amb l'arrossegament d'elements (drag and drop)	drag, dragend, dragenter, dragleave, dragover, dragstart, drop
Relacionats amb video i audio	audioprocess, canplay, canplaythrough, durationchange, emptied, ended, ended, loadeddata, loadedmetadata, pause, play, playing, ratechange, seeked, seeking, stalled, suspend, timeupdate, volumechange, waiting, complete
Relacionats amb la connexió a internet	disabled, enabled, offline, online, statuschange, connectionInfoUpdate

4.7 Introducció al DOM

Exemple 1: Si fem click damunt un text, mostrarà una finestra emergent.

```
<html>
  <head>
    ...
  </head>
 <body>
  <p onclick="alert('Has fet un click');"> Fes click damunt d'aquest text. </p>
 </body>
</html>
```

4.7 Introducció al DOM

Exemple 2: Carregam al body una funció que genera una missatge amb finestra emergent.

```
<html>
  <head>
    <meta charset = "UTF-8"/>
  </head>
  <body onload='missatge()'>
    <script>
    function missatge(){
        alert('Aquest missatge surt quan es carrega la pàgina');
        }
    </script>
  </body>
</html>
```

4.7.2. Objecte Window

És l'objecte de la jerarquia més alt. Tots els altres objectes possibles d'una pàgina web (excepte navigator i screen) estan inclosos en ell. Amb ell ens referim a la finestra actual.

ALGUNES PROPIETATS DE L'OBJECTE WINDOW

- **closed**: Ens diu si la finestra està tancada, i en aquest cas el seu valor serà true, o oberta, false.
- **defaultStatus**: És el text que apareix en la barra d'estat del navegador. És una cadena.
- **frames**: Conté en un array tots els frames de la finestra. Estaran en el mateix ordre en el qual es defineixen en el document, sent el primer el d'índex número 0.
- **history**: Array que guarda les pàgines que hem visitat, s'emmagatzema un historial amb elles.
- **length**: Guarda el número de frames que conté la pàgina actual.
- **location**: Localització de la pàgina. És el que se'ns mostra en la barra de direcció.
- **name**: Conté el nom de la finestra, o frame actual.
- **status**: Cadena amb el missatge que té la barra d'estat.

4.7.2. Objecte Window

Exemples:

Retorna la URL de la pàgina actual.

```
window.location.href
```

Retorna la ruta i el nom de l'arxiu de la pàgina actual.

```
window.location.pathname
```

Carrega un nou document.

```
window.location.assign
```

4.7.2. Objecte Window

ALGUNS MÈTODES DE L'OBJECTE WINDOW

- **alert(missatge)**: Mostra el missatge en un quadre emergent.
- **blur()**: Elimina el focus de l'objecte window actual.
- **close()**: Tanca l'objecte window actual.
- **confirm(missatge)**: Mostra un quadre de diàleg amb el missatge 'missatge' i dos botons, un d'acceptar i un altre de cancel·lar. Retorna true si es prem acceptar i false si es prem cancel·lar.
- **focus()**: Captura el focus del ratolí sobre l'objecte window actual.
- **moveBy(x,i)**: Mou l'objecte window actual el nombre de píxels especificats per (x,i).
- **moveTo(x,i)**: Mou l'objecte window actual a les coordenades (x,i).
- **open(URL, nom, característiques)**: Obri una finestra amb el nom que li especifiquem i en ella la URL que posem en el primer paràmetre. Podem indicar les característiques d'aquesta finestra, les possibilitats que tenim són les següents:
- **toolbar**: Si desitgem que la finestra tinga barra d'eines o no. Pot ser yes/no o 1/0.
- **location**: Si desitgem que la finestra tinga camp de localització o no. Pot ser yes/no o 1/0.

4.7.2. Objecte Window

ALGUNS MÈTODES DE L'OBJECTE WINDOW

- **directories**: Si desitgem que la finestra tinga botons de direcció o no. Pot ser yes/no o 1/0.
- **status**: Si desitgem que la finestra tinga barra d'estat o no. Pot ser yes/no o 1/0.
- **menubar**: Si desitgem que la finestra tinga barra de menu o no. Pot ser yes/no o 1/0.
- **scrollbars**: Si desitgem que la finestra tinga barres de desplaçament o no. Pot ser yes/no o 1/0.
- **resizable**: Si desitgem que la finestra pugui ser canviada de grandària o no. Pot ser yes/no o 1/0.
- **width**: Amplada que volem que tinga la finestra en píxels.
- **height**: Alt que volem que tinga la finestra en píxels.
- **left**: Distància en píxels des del costat esquerre de la pantalla a la qual volem col·locar la finestra.
- **Top**: Distància en píxels des del costat superior de la pantalla a la qual volem col·locar la finestra.
- **prompt(missatge, valorInicial)**: Mostra un quadre de diàleg amb el missatge especificat i ens permet introduir un valor. Aquest valor és guardat en una variable de tipus text. El segon paràmetre és opcional i permet donar un valor inicial a aquesta variable.
- **scrollBy(x,i)**: Desplaça la finestra els píxels especificats per (x,i).
- **scrollTo(x,i)**: Desplaça la finestra a les coordenades (x,i).
- **setInterval()**: Avalua una expressió en intervals específics (en milisegons).

4.7.2. Objecte Window

Exemples:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <h2> Exemples amb mètodes de l'objecte window </h2>
    <button onClick="obrir_finestra()"> Fes un click per obrir una finestra amb la url Google </button> <br><br>
    <button onClick="tancar_finestra()">Fes un click per tancar la finestra actual</button> <br><br>
    <button onClick="print()"> Fes click per imprimir el contingut de la web actual </button><br>
    <script>
        function obrir_finestra() {
          var finestra = window.open("http://www.google.es/", "Google", "menubar=yes,resizable=yes,scrollbars=yes");
        }
        function tancar_finestra() {
          var finestra = window.close();
        }
    </script>
</body>
</html>
```

4.7.3. Objecte document

Aquest objecte és el que conté tota la pàgina actual que s'està visualitzant en el moment. Depèn de l'objecte window. Ens permetrà afegir de manera dinàmica contingut a la nostra pàgina.

PROPIETATS

- **alinkColor**: És el color dels enllaços actius.
- **anchors**: Array amb els enllaços del document.
- **bgColor**: Es refereix al color de fons del document.
- **cookie**: A través de ella podem crear, llegir, modificar i eliminar cookies.
- **domain**: Nom del domini del servidor que ha servit el document.
- **fgColor**: Propietat que conté el color del text.
- **forms**: Array amb tots els formularis del document. Aquests al seu torn tenen els seus elements que ja veurem en el seu moment.
- **images**: Array que conté les imatges del document.
- **lastModified**: Conté la data de l'última modificació del document.

4.7.3. Objecte document

Aquest objecte és el que conté tota la pàgina actual que s'està visualitzant en el moment. Depèn de l'objecte window. Ens permetrà afegir de manera dinàmica contingut a la nostra pàgina.

PROPIETATS

- **linkColor**: Emmagatzema el color dels enllaços.
- **links**: Array amb els enllaços externs.
- **location**: Cadena que conté la URL del document.
- **title**: Cadena que conté el títol del document actual.
- **vlinkColor**: Color dels enllaços que ja han sigut visitats.

MÈTODES

- clear(): Netejar finestra del document.
- open(): Obrir escriptura sobre un document.
- close(): Tancar escriptura sobre el document actual.
- write(): Escriure text en el document.
- writeln(): Escriu text en el document afegint un salt de línia al final.

4.7.3. Objecte document

Exemple:

```
<!DOCTYPE html>
<head>
      <meta charset="UTF-8">
      <meta http-equiv="X-UA-Compatible" content="IE=edge">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Exemple de l'objecte document</title>
</head>
<body>
      <h2> Exemple de l'objecte document </h2>
      <button onClick="mostrar_data()"> estat </button><br><br>
      <script>
      function mostrar_data() {
            alert(document.lastModified); //Mostra la data de l'última modificació del document web actual
      }
      </script>
</body>
</html>
```

4.7.4. Objecte navigator

L’objecte Navigator ens permet conèixer dades com el navegador que s'utilitza, la seva versió, el sistema operatiu, etc.

Propietats

- **appCodeName**: Retorna el codi del nom del navegador.
- **appName**: Retorna el nom del navegador.
- **appVersion**: Retorna la versió del navegador.
- **cookieEnable**: Determina si les cookies estan habilitades o no.
- **platform**: Retorna la plataforma sobre la qual s'està executant el navegador.
- **userAgent**: Retorna una información completa sobre l'agent d'usuari (normalment és el navegador).

Mètode

- **javaEnabled()**: Informa si el navegador està habilitat per a soportar l'execució de programes escrits en Java.

4.7.4. Objecte navigator

Exemples:

```
<script>
 document.writeln("El nom del navegador és: " + navigator.appName );
 document.writeln(" Les cookies del teu navegador estan habilitades?: " + navigator.cookieEnabled
);
</script>
```

4.7.5. Accés al DOM

El model d'objectes del document (DOM), és una interfície de programació per als documents HTML i XML.

- És una representació completament orientada a l'objecte de la pàgina web i pot ser modificat amb un llenguatge de script com Javascript.
- Facilita una representació estructurada del document i defineix de quina manera els programes poden accedir, a fi de modificar, tant la seva estructura com l'estil i contingut.
- Dóna una representació del document com un grup de nodes i objectes estructurats que tenen propietats i mètodes. Essencialment, connecta les pàgines web a scripts o llenguatges de programació.
- Va ser dissenyat per ser independent de qualsevol llenguatge de programació particular.
- Totes les propietats, mètodes i esdeveniments disponibles per a la manipulació i la creació de pàgines web està organitzat dins d'objectes.

4.7.5. Accés al DOM

DOM i Javascript

El DOM no és un llenguatge de programació, però sense ell, el llenguatge Javascript no té cap model o noció de les pàgines web HTML ni de les pàgines XML.

Cada element (per exemple, tot el document, el títol, les taules, els títols de les taules, etc.), és part del model d'objecte del document, d'aquesta manera, es pot accedir i manipular-los (mitjançant el DOM) i amb un llenguatge d'escriptura com és Javascript.

Al començament, JavaScript i DOM estaven hermèticament enllaçats, però després es van desenvolupar com a entitats separades.

4.7.5. Accés al DOM

Com s'accedeix a DOM?

No s'ha de fer res especial per començar a utilitzar el DOM.

Els diferents navegadors tenen directrius DOM diferents, i aquestes directrius tenen diversos graus de conformitat a l'actual estàndard DOM, però tots els navegadors web empren el model d'objecte de document per fer accessibles les pàgines web a l'script.

En Javascript, la forma d'accedir al DOM, és a través dels seus nodes, anem a veure els tipus de nodes que hi podem trobar.

4.7.5. Accés al DOM

Tipus de nodes del DOM

L'especificació completa de DOM defineix 12 tipus de nodes, encara que les pàgines HTML, es poden manejar amb només 4 o 5 tipus de nodes:

- **document**: El node document és el node arrel, a partir del qual deriven la resta de nodes.
- **element**: Són els nodes definits per etiquetes html. Per exemple, una etiqueta "div" genera un node. Si dins del "div" tenim 3 etiquetes "p", aquestes etiquetes defineixen nodes fills de l'etiqueta "div".
- **text**: El text dins d'un node element es considera un nou node fill de tipus text.
- **attr**: Els atributs de les etiquetes defineixen nodes, encara que amb Javascript no les veurem com a nodes, si no que es consideren informació associada al node de tipus element.
- **comment**: Els comentaris i altres elements com són les declaracions "doctype" a la capçalera dels documents HTML generen també nodes.

4.7.5. Accés al DOM

Cóm s'accedeix als nodes?

DOM proporciona dos mètodes alternatius per accedir a un node específic:

- a través del node pare (l'arrel)
- o amb accés directe.

És més ràpid accedir de forma directa, per tant veurem les funcions per accedir directament.

És important saber que per accedir, modificar o eliminar algun node, només es pot fer quan l'arbre DOM s'ha construït per complet, és a dir, quan la pàgina s'ha carregat per complet.

4.7.5. Accés al DOM

Mètodes d'accés directe:

- **document.getElementById**(): És el mètode o funció més emprada quan es fan aplicacions web dinàmiques. Permet accedir directament a un node per llegir i/o modificar les seves propietats. Retorna un element HTML que té un atribut que coincideix amb el que s'ha indicat al paràmetre de la funció.

Sintaxi:

```
var variable=document.getElementById("valor_id");
```

4.7.5. Accés al DOM

Exemple: Canviar el color d'un títol h2

```
<html>
   <head>
      <title>Exemple getElementById</title>
   </head>
   <body>
      <h2 id="clic">Text que ha de canviar de color</h2>
      <button onclick="canviarcolor('orange');">Taronja</button>
      <button onclick="canviarcolor('green');">Verd</button>
      <script>
      function canviarcolor(pcolor) {
            var x = document.getElementById('clic'); // Posa dins la variable "x" l'element HTML gràcies al id="clic". I així podrem canviar aquest element.
            x.style.color = pcolor; // Posam aquest element en el color que indica el paràmetre "pcolor". Accedim a la propietat style.color de l'element.
      }
      </script>
      </body>
 </html>
```

4.7.5. Accés al DOM

- **document.getElementsByTagName()***:* Obté, en forma d'array, tots els elements de la pàgina HTML on l'etiqueta sigui igual que el paràmetre que se li passa a la funció.

Sintaxi:

```
var variable = document.getElementsByTagName("nom_etiqueta_element");
```

El valor que retorna és un objecte de tipus "nodelist" (similar a l'array) amb tots els nodes que tenen la etiqueta igual al paràmetre que passa el mètode. Per obtenir el valors d'aquest objecte, es farà igual que amb els arrays:

Exemple: Obtenir els títols h1 de la pàgina

```
var titol = document.getElementsByTagName("h1");  <!-- titol serà un array de tots el <h1> de la pàgina.-->
```

4.7.5. Accés al DOM

Per recorre l'objecte i obtenir totes les etiquetes títol h1 de la pàgina, podriem fer amb un for:

Exemple: Canviar el color del fons dels h1

```
<html>
   <head>
     <meta charset="UTF-8">
     <title>h1 en vermell</title>
   </head>
   <body>
     <h1>Primer títol</h1>
     <h1>Segon títol</h1>
     <script>
          var titol = document.getElementsByTagName("h1");
          for (var i=0; i< titol.length; i++) {
          titol[i].style.backgroundColor = 'red'
          }
     </script>
   </body>
</html>
```

4.7.5. Accés al DOM

- **document.getElementsByClassName():** Retorna tots els elements amb el nom de la classe especificada.

  Sintaxi:

```
var x = document.getElementsByClassName("nom_classe"); //on x és de tipus nodelist (array).
```

4.7.5. Accés al DOM

Exemple: Canviar l'estil d'un títol de tipus <h3> i d'un paràgraf amb la mateixa classe.

```
<html>
   <head>
       <meta charset="UTF-8">
       <title>Exemple</title>
   </head>
   <body>
       <h3 class="canvi">Títol h3 de la pàgina amb la classe anomenada "canvi"</h3>
       <p> Paràgraf de la pàgina sense classe.</p>
       <p class="canvi">Paràgraf de la pàgina amb la classe anomenada "canvi"</p>
       <button type=button onclick="classes()"> clica aquí</button>
       <script>
       function classes() {
              var classe=document.getElementsByClassName('canvi');
              for (i = 0; i < classe.length; i++){
              classe[i].style.color = "orange";
              classe[i].style.fontSize = "25px";
              classe[i].style.backgroundColor = "yellow";
              classe[i].style.borderStyle = "solid";
              }
       }
       </script>
       </body>
 </html>
```

4.7.5. Accés al DOM

- **Mètodes per crear nodes:**
  - **document.createElement():** Crea un node tipus d'element. El paràmetre serà per tant una etiqueta HTML.

Sintaxi:

```
var x = document.createElement("etiqueta_element");
```

Exemple: Afegim un element "p" al body.

```
<html>
   <body>
        <h1>Fes clic al botó i es crearà un botó</h1>
        <button onclick="funciocrearbotó()">Fes click</button>
        <script>
            function funciocrearbotó() {
                var boto = document.createElement("p"); // Cream l'element p.
               boto.InnerHTML = "Text del paràgraf"; // InnerHTML és on es guardarà el text
de l'element.
            }
        </script>
    </body>
</html>
```

4.7.5. Accés al DOM

- **document.createTextNode():** Crea un node de tipus text.

Sintaxi: `var t = document.createTextNode("text");`

Exemple: Afegeix element a una llista

```
<html>
   <body>
         <h1>Fes clic i s'afegirà elements a una llista</h1>
         <button onclick="afegirtext()">Fes click</button>
         <ol id="llista">
         <li>Un element</li>
         <li>Un altre element</li>
         </ol>
         <script>
         function afegirtext() {
               var nodeElement = document.createElement("li"); // Crea un node element <li>
               var nodeText = document.createTextNode("Text d'un altre element"); // Crea un text node amb el seu text
               nodeElement.appendChild(nodeText); // Afegeix el node de text a l'element "li
               document.getElementById("llista").appendChild(nodeElement);  Afegim el node element amb el seu text al pare
         }
         </script>
  </body>
</html>
```

4.7.5. Accés al DOM

Mètodes per a la gestió de nodes

- **Inserir:** Els dos mètodes anteriors, ens serveixen per crear elements i nodes de text, però si no les afegim a l'arbre, aquests elements no representaran bé el contingut de la pàgina web. Per poder afegir aquests elements tenim els següent mètodes:

node.appendChild(): Ens permet afegir un nou node element fill a l'arbre. Com podrem veure a la sintaxi aquest mètode se vincula directament al node que volem afegir el nou node.

Sintaxi:

```
elementpare.appendChild(nouelement);
```

4.7.5. Accés al DOM

- **node.insertBefore():** Ens permet afegir un nou element abans d'una altre.

Sintaxi: `elementpare.insertBefore(nodeElementNou, nodeGerma);`

Exemple: Inserim un paràgraf com a primer, és a dir, abans de l'actual primer paràgraf.

```
<html>
<body>
  <div id="parraf">
        <p>Primer p</p>
        <p>Segon p</p>
  </div>
  <button onclick=afegir()> Click </button>
  <script>
        function afegir() {
        var nodeElement = document.createElement('p'); //cream i guardam a una variable el nou paragraf
        noup.appendChild(document.createTextNode('Nou paragraf')); //Afegim el text
        var nodePare = document.getElementById('parraf'); // On inserirem
        var nodeGerma = nodePare.firstChild; // Guardam el primer paràgraf.
        nodePare.insertBefore(nodeElement, nodeGerma); //inserim el nou element
        }
  </script>
</body>
</html>
```

4.7.5. Accés al DOM

Reemplaçar/eliminar/clonar:

- **node.replaceChild():** Permet reemplaçar un node element existent.

  Sintaxi: `nodepare.replaceChild(nodenou, nodesubstituit)`

  Exemple: Reemplaçam un paràgraf per un altre

```
<html>
<body>
   <section id="s1">
  <p id="p1">Un paràgraf</p>
  <p id="p2">Un altre paràgraf</p>
   </section>
   <script>
  var nodeNou=document.createElement("p");
  var nodeText=document.createTextNode("Nou paràgraf");
  nodeNou.appendChild(nodeText);
  var nodePare=document.getElementById("s1");
  var nodeGerma=document.getElementById("p1");
  nodePare.replaceChild(nodeNou,nodeGerma);
   </script>
</body>
</html>
```

4.7.5. Accés al DOM

- **node.removeChild():** Permet esborrar un node element.

  Exemple: Esborrar elements d'una llista

```
<!DOCTYPE html>
<html>
<body>
   <ul id="llista">
          <li>Ana</li>
          <li>Pep</li>
          <li>Leire</li>
   </ul>
   <p>Clica al botó per eliminar elements a la llista</p>
   <button onclick="eliminar()">Fes click</button>
   <script>
          function eliminar() {
          var list = document.getElementById("llista"); //Guardam a una variable el punt a partir del qual hem d'eliminar
          list.removeChild(list.childNodes[0]); // Esborram el primer element. childNodes és una propietat que veurem a
continuació
          }
   </script>
</body>
</html>
```

- **node.cloneNode(true):** Permet fer una copia exacta d'un node element. Amb el paràmetre "true", el que es fa, és dir-li al mètode que volem clonar el node i tots els seus fills (inclosos els de text).

4.7.5. Accés al DOM

- **node.removeChild():** Permet esborrar un node element.

  Exemple: Esborrar elements d'una llista

```
<!DOCTYPE html>
<html>
<body>
   <ul id="llista">
          <li>Ana</li>
          <li>Pep</li>
          <li>Leire</li>
   </ul>
   <p>Clica al botó per eliminar elements a la llista</p>
   <button onclick="eliminar()">Fes click</button>
   <script>
          function eliminar() {
          var list = document.getElementById("llista"); //Guardam a una variable el punt a partir del qual hem d'eliminar
          list.removeChild(list.childNodes[0]); // Esborram el primer element. childNodes és una propietat que veurem a
continuació
          }
   </script>
</body>
</html>
```

- **node.cloneNode(true):** Permet fer una copia exacta d'un node element. Amb el paràmetre "true", el que es fa, és dir-li al mètode que volem clonar el node i tots els seus fills (inclosos els de text).

4.7.5. Accés al DOM

- **Algunes propietats generals del nodes**

Independentment del tipus de node, hi ha una sèrie de propietats generals que comparteixen tots:

- **node.childNodes**: Ens retorna un array amb tots els nodes fill d'un element. És de només lectura. Els nodes resultants són tant elements, com nodes de tipus text.

  Sintaxi:

  ```
  var variable = document.element.chilNodes;
  ```

  Exemple: Obtenir els nodes de l'element body

  ```
  var c = document.body.childNodes; //obtenim un array "c" amb els fills de "body"
  ```

  Nota: Per accedir a cadascun dels fills haurem de fer-ho com un array ([])

- **node.children:** Retorna els nodes fills de tipus element d'un node pare.

4.7.5. Accés al DOM

- **node.firstChild**: Retorna el primer node fill directe d'un node, i null si no té fills. És de només lectura. A alguns navegador, el contingut de "firstChild", també inclou com a fill possibles espais en blanc o salts de línia que l'element pot contenir, i això ocasiona problemes.

  Exemple: Obtenir el primer fill d'un element span

```
<p id="ex">
  <span> això és un span </span>
</p>
<script>
  var x = document.getElementById('ex');
  var fill = x.firstChild;
</script>
```

- **lastChild**: Ens retorna el darrer node fill directe de l'element (funciona com el firstChild).

4.7.5. Accés al DOM

- **nextSibling: Ens retorna el germà següent al node que s'especifica.**

  Sintaxi:

  var node = document.getElementById('valor_id').nextSibling

- **previousSibling**: Ens retorna el germà anterior al node que s'especifica. Retorna null si el node és el primer fill.
- **hasChildNodes**: Indica si el node té fills o no. Retorna un booleà.
- **parentNode**: Retorna el node pare d'un element.
- **nodeName:** És el nom del node, que es defineix depenent del tipus de node (qualsevol tipus de node). Retorna una cadena literal.

  Exemple:

```
alert( document.body.nodeName ); // BODY
```

- **tagName:** Lo mateix que nodeName, però només per a nodes de tipus element.

4.7.5. Accés al DOM

- **node.innerHTML: E**ns permet recuperar el contingut text d'un element o assignar nou contingut text. Si assignam nou text, destruim el que té. Emmagatzema també la informació d'HTML.

  Sintaxi:

```
var contingut = document.getElementById("valor_id").innerHTML;
```

Exemple: Canviar el text d'un paràgraf

```
<!DOCTYPE html>
<html>
<body>
	<p id="inner" onclick="canviar()">Clica damunt aquest paràgraf per canviar el
text.</p>
	<script>
	function canviar() {
		document.getElementById("inner").innerHTML = "Nou text";
	}
	</script>
</body>
</html>
```

4.7.5. Accés al DOM

Treballant amb atributs

Hi ha quatre mètodes per modificar atributs d'elements:

- **hasAttributte():** Comprova si un element té un atribut concret. Retorna un booleà.

  Sintaxi:

  ```
  element.hasAttribute('nom_atribut');
  ```

4.7.5. Accés al DOM

Exemple: Comprovar que un element botó té un atribut determinat

```
<html>
  <body>
      <p>Clica para saber si l'element botó té l'element onclick.</p>
      <button id="boto" onclick="has()">clica</button>
      <p id="demo"></p>
      <script>
      function has()
      {
      var x = document.getElementById("boto").hasAttribute("onclick");
      if (x==true) {
            document.write("el botó té l'atribut onclik")
      }
      else {
            document.write("El botó no té l'atribut onclick")
      }
      }
      </script>
   /body>
</html>
```

4.7.5. Accés al DOM

- **getAttribute():** Mostra el valor d'un atribut especificat o null.

  Sintaxi:

  ```
  element.getAttribute("nom_atribut")
  ```

4.7.5. Accés al DOM

Exemple: Mostrar valor d'un atribut

```
<!DOCTYPE html>
<html>
<head>
   <style>
        .vermell {
        color: red;
        background-color: rgb(0, 255, 255)
        }
   </style>
</head>
<body>
        <h1 class="vermell">Provam getAttribute</h1>
        <h1> Un mètode del DOM</h1>
        <button onclick="comprovar()">Clica</button>
        <p id="afegirparagraf"></p>
        <p>afegirem mitjançant un paragraf, el valor de l'atribut class que du el primer títol</p>
        <script>
        function comprovar() {
                var x = document.getElementsByTagName("h1")[0].getAttribute("class");  /* guardam dins d'x el valor
                                                                                        * de l'atribut class del 1er element h1*/
                document.getElementById("afegirparagraf").innerHTML = x; //innerHTML ens permet afegir contingut a un element
        }
        </script>
</body>
</html>
```

4.7.5. Accés al DOM

- **setAttribute():** Agrega o actualitza el valor d'un atribut especificat.

  Sintaxi:

```
element.setAttribute("nom", "valor");
```

4.7.5. Accés al DOM

Exemple: Agregar atribut "class"

```
<!DOCTYPE html>
<html>
<head>
   <style>
      .vermell {
      color: red;
      }
</style>
</head>
<body>
   <h1>Aplicarem una classe a aquest títol</h1>
   <button onclick="aplicarclasse()">clica</button>
   <script>
      function aplicarclasse() {
            document.getElementsByTagName("h1")[0].setAttribute("class", "vermell");
      }
   </script>
</body>
</html>
```

4.7.5. Accés al DOM

- **removeAttribute():** Elimina un atribut d'un element.

  Sintaxi:

```
element.removeAttribute("valor_atribut");
```

4.7.5. Accés al DOM

Treballant amb classes

- **className**: És un propietat. Obté o estableix una classe.

  Sintaxi:

  ```
  element.className;
  ```

Exemple: Afegir una classe a h1

```
<html>
   <head>
       <meta charset="UTF-8">
       <title>Afegir classe</title>
       <style>
       .estilnou { color: green; }
       </style>
   </head>
   <body>
       <h1 id="ex">Afegim classe</h1>
       <script>
       document.getElementById("ex").className = "estilnou";
       </script>
   </body>
</html>
```

4.7.5. Accés al DOM

- **classList.add():** És un mètode que agrega, un o més valors de classe.

  Sintaxi:

  ```
  element.classList.add("valor",...);
  ```

4.7.5. Accés al DOM

Exemple: afegir classes amb classList.add()

```
<html>
<head>
   <style>
       .estil {
       width: 300px;
       height: 50px;
       background-color: red;
       }
       .estil2 {color: red; }
   </style>
</head>
<body>
   <button onclick="afegirclasse()">clica</button>
   <main id="nav"> <p>Text del main</p> </main>
   <script>
       function afegirclasse() {
              document.getElementById("nav").classList.add("estil","estil2");
       }
   </script>
</body>
</html>
```

4.7.5. Accés al DOM

- **classList.toggle():** És un mètode que pot tenir un o dos arguments, si n'hi ha un, alterna el valor de la classe, és a dir, si la classe existeix l'elimina i retorna false, i si no existeix l'afegeix i retorna true. Si s'indiquen els dos arguments, si el segon argument s'avalua com a true, s'afegeix la classe i si s'avalua com a false, elimina la classe.

  Sintaxi:

  ```
  element.classList.toggle("classe", "valor_true_o_false");
  ```

- **ClassList.contains():** Aquest mètode comprova si el valor de classe existeix.

  Sintaxi:

  ```
  element.classList.contains('active');
  ```

- **classList.replace():** Mètode que substitueix un valor de classe existent per un nou.

  Sintaxi:

  ```
  element.classListreplace('old','new');
  ```

- **classList.remove():** Mètode per eliminar un valor de classe.

  Sintaxi:

  ```
  element.classList.remove('active');
  ```

4.7.5. Accés al DOM

Alguns atributs són accessibles de forma directa, es tracten com a propietats: href, style, className, src, etc..

Exemples:

```
var enllaç = document.getElementById("enl");
alert(enllaç.href); // mostrarà la URL de l'enllaç amb id="enl")

var imatge = document.getElementById("img");
alert.log(imatge.style.margin);  //mostrarà el valor del marge de la imatge amb id="img"

var p = document.getElementById("par");
alert(p.className); // mostrarà el nom de la classe del paràgraf amb id="par"

var imatge= document.getElementById("img");
alert(img.src); // mostrarà el fitxer d'imatge que l'element imatge amb id="img"
```

4.7.6. Formularis HTML i el DOM

Propietats bàsiques d'accés al formulari i als seus elements

- **Mitjançant l'objecte "document":**

  Quan es carrega una pàgina web, el navegador crea automàticament un array anomenat "***forms***" que conté la referència a tots els formularis de la pàgina. Per a accedir a aquest array, s'utilitza l'objecte "document", per tant "document.forms" és l'array que conté tots els formularis de la pàgina. Com es tracta d'un array, l'accés a cada formulari es realitza amb la mateixa sintaxi dels arrays. La següent instrucció accedeix al primer formulari de la pàgina: document.forms[0].

  També es crea un array anomenat "***elements***" per cadascun dels formularis de la pàgina, que conté la referència a tots els elements (quadres de text, botons, llistes desplegables, etc.) d'aquest formulari. La següent instrucció obté el primer element del primer formulari de la pàgina: `document.forms[0].elements[0] .`

  Aquesta manera d'accedir als array de formularis i elements és ràpida i senzilla, però si es canvia el disseny de la pàgina (per exemple l'ordre dels formularis), ja no funciona correctament.

4.7.6. Formularis HTML i el DOM

- **A través de l'atribut "name" i/o "id":**

Per evitar la dependència al disseny indicada anteriorment, es pot accedir als formularis a través del seu nom (atribut name) o a través del seu atribut id (com amb qualsevol altre element):

```
Per accedir als formularis:

var PrimerForm = document.nomformulari1;
var SegonForm = document.nomformulari2;

HTML:
<form name="nomformulari1" >
  ...
</form>
<form name="nomformulari2" >
  ...
</form>
```

```
Per accedir als elements:

var PrimerElement =
document.nomformulari1.element;

HTML:
<form name="nomformulari1">
  <input type="text" name="element"
/>
</form>
```

4.7.6. Formularis HTML i el DOM

- **Mitjançant les funcions/mètodes del DOM:**

Exemple: Empram la funció document.getElementById() per accedir directament a un formulari i a un dels seus elements.

```
var PrimerForm = document.getElementById("formulari");
 //suposam que "formulari" és l'id del formulari

var PrimerElement = document.getElementById("idelement");
//suposam que "idelement" és l'id del primer element del formulari

HTML:
<form name="nomformulari1" id="formulari" >
  <input type="text" name="element" id="idelement" />
</form>
```

4.7.6. Formularis HTML i el DOM

Exemple: Funció JavaScript amb dades recollides d'un formulari.

```
<!DOCTYPE html>
<head>
	<meta charset="UTF-8">
	<title>Document</title>
</head>
<body>
	<form>
	<fieldset>
		<label for="nom">Introdueix el teu nom:</label>
		<input type="text" id="nom"/>
		<button type="button" onclick="saludar()">Executar</button>
	</fieldset>
	</form>
	<script>
	function saludar(){
		var n=document.getElementById("nom");
		var usuari= n.value;
		alert("Hola " +usuari);
	}
	</script>
</body>
</html>
```

Nota: Dins de "value", és a on trobem la dada introduïda al formulari.

4.7.7. Maneig d'errors

Estructura de control d'errors try/catch

Els errors a Javascript són objectes que es mostren cada vegada que es produeix un error de programació. Aquest objectes contenen molta informació sobre el tipus d'error.

Javascript també permet crear errors personalitzats per proporcionar informació adicional.

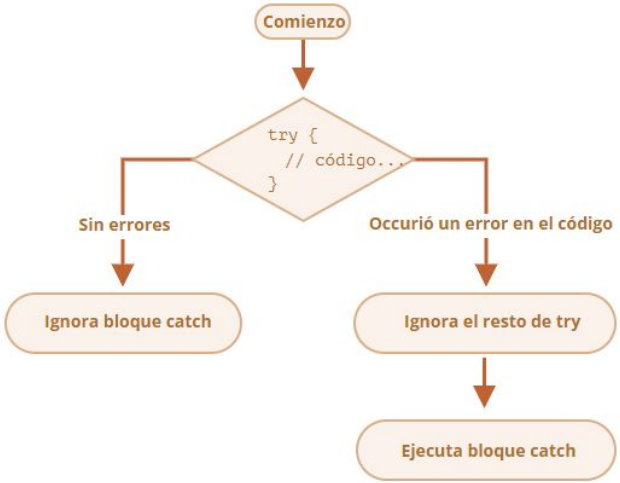
Try … catch és el tipus d'estructura que ens serveix per controlar i comprovar el flux d'un programa davant comportaments inesperats, és adir, per gestionar els errors. El que fa és llançar una acció de protecció davant els possibles errors.

4.7.7. Maneig d'errors

Sintaxis try … catch

Té dos blocs principals:

```
try {
   // codi...
} catch (err) {
   // manipulació d'error
}
```



Funcionament:

- Primer, s'executa el codi a try {...}.
- Si no hi ha hagut errors, s'ignora catch(err), és a dir, l'execució arriba al final de try i continua, ometent catch.
- Si es produeix un error, l'execució de try s'atura i el control va al començament catch(err). La variable o paràmetre "err" contendrà un objecte d'error amb detalls sobre aquest error.

Conclusió: Un error dins del bloc try {...}, no mata l'script, tenim l'oportunitat de manejar-lo a catch.

4.7.7. Maneig d'errors

Exemples:

Sense errors:

```
try {
  alert('Hola');
  alert('Adeu');
} catch (err) {
  alert('Se ignora catch perquè no hi ha errors'); /No se mostra
}
```

Amb errors:

```
try {
  alert('Hola');
  quetal
  alert('Adeu');
} catch (err) {
   alert(err); //es mostra un error intern "quetal no definit"
}
```

4.7.7. Maneig d'errors

- **try...catch només funciona per a errors en temps d'execució**. Per exemple, No funcionarà si el codi és sintàcticament incorrecte, per exemple, si hi ha claus sense tancar. Llavors, try...catch només pot manejar errors que ocorren en un codi vàlid. Aquests errors es denominen “errors de temps d'execució” o, a vegades, “excepcions”.

Atenció:

- Si utilitzam "let" per declarar una variable dins de "try", aquesta variable no estirà disponible per ser utilitzada dins de "catch" o "finally", això és per el que ja sabem, que son declaracions d'àmbit local.

Exemple:

```
try {
 let x = 10;
 throw "La variable x no té àmbit global ";  //Això és una excepció. Aquest concepte es veurà a continuació
}
catch(err) {
 alert(err); // Retorna "Reference a is not defined"
}
```

4.7.7. Maneig d'errors

Objecte Error

Quan es produeix un error, Javascript genera un objecte que conté els detalls sobre aquest error.

L'objecte es passa com a argument a catch:

```
try {
  // ...
} catch(err) { // Objecte error
  // ...
}
```

4.7.7. Maneig d'errors

- **Parts de l'objecte error**: Per aquests errors, l'objecte té dues propietats principals:
  - name: És el nom de l'error. Per exemple, per a una variable indefinida és "ReferenceError".
  - message: És el missatge de text sobre els detalls de l'error.

```
try {
  variable // error, variable no està definit
} catch (err) {
  alert(err.name); // Mostra: ReferenceError
  alert(err.message); // Mostra: variable no està definit
}
```

També es pot mostrar un error amb el nom i el missatge a la mateix vegada

```
try {
  variable // error, variable no està definit
} catch (err) {
  alert(err); // ReferenceError: variable no està definit
              // L'error es converteix en una cadena com "nom: missatge"
}
```

4.7.7. Maneig d'errors

- **Parts de l'objecte error**: Per aquests errors, l'objecte té dues propietats principals:
  - name: És el nom de l'error. Per exemple, per a una variable indefinida és "ReferenceError".
  - message: És el missatge de text sobre els detalls de l'error.

```
try {
  variable // error, variable no està definit
} catch (err) {
  alert(err.name); // Mostra: ReferenceError
  alert(err.message); // Mostra: variable no està definit
}
```

També es pot mostrar un error amb el nom i el missatge a la mateix vegada

```
try {
  variable // error, variable no està definit
} catch (err) {
  alert(err); // ReferenceError: variable no està definit
              // L'error es converteix en una cadena com "nom: missatge"
}
```

4.7.7. Maneig d'errors

Excepcions

- **Excepcions amb throw:** Es poden llançar excepcions mitjançant la sentència **throw**, quan l'objecte l'error no està definit.

Exemple: **arxiu html:**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Exepcions</title>
</head>
<body>
    <form>
        <label>Introdueix un número entre 5 i 10:</label>
        <input type="text" id='t'>
        <input type="button" value="click" onclick="er()">
    </form>
    <script src="err.js"></script>
</body>
</html>
```

arxiu Javascript "err.js"

```
function er() {
  x = document.getElementById("t").value;
  try {
    if (x == "") throw "input buit";
    if (isNaN(x)) throw "No és un número";
    x = Number(x);
    if (x < 5) throw "Massa petit";
    if (x > 10) throw "Massa gran";
  } catch (e) {
    //tractam l'excepció
    alert(e);
  }
}
```

4.7.7. Maneig d'errors

Exemple: Suma de n nombres (màxim 10), sense la part catch que mostraria l'excepció llançada

```
function suma(arguments) { //argument és una array
      try{
            var resultat = 0,
            l = arguments.length;
            if ( l > 10 ) {
                  throw 'Massa arguments!';  // La resta del codi serà ignorat quan es llança l'excepció
            }
            for ( var x = 0; x < l; x++ ) {
                  resultat += arguments[x];
            }
            return resultat;
      }
      catch(err){
            alert(err)
      }
}
```

4.7.7. Maneig d'errors

Try...catch.. finally

També podem tenir try...catch...finally. En aquest cas el bloc finally s'executarà tant si el codi s'executa correctament com si es produeix un error.

Sintaxi:

```
try {
  // sentències
 }
 catch(error) {
  // sentències d'errors
}
 finally() {
  // sentències que s'executaran sempre
 }
```

4.7.7. Maneig d'errors

Exemple:

```
try {
  throw 'Hi ha un error al codi'; }
catch (err) {
  alert(err.message);
}
finally {
  alert('s'ha executat el bloc de finally'); ç
}
```

La sortida serà: Una finestra emergent amb l'error: Hi ha un error al codi Una finestra emergent amb el codi de finally: s'ha executat el bloc de finally

4.7.7. Maneig d'errors

- Si tenim una instrucció "return" dins d'un "finally" i també dins d'un "try", només s'executarà la instrucció del "finally"

Exemple:

```
function exemple() {
try {
   return 100;
} finally {
   console.log("S'executa la part de finally");
   return 5000;
}
}
console.log(exemple()); // La consola retorna 5000
```