# Bevezetés a JavaScript használatába

## 2. rész

Hatwágner F. Miklós

Széchenyi István Egyetem, Győr

2015. november 24.

# Eseménykezelés

Az eseménykezelés megvalósításának lehetséges módjai:

- soron belüli eseménykezelővel
- klasszikus módszerrel
- W3C eseménykezeléssel
- IE módszerével

# Soron belüli eseménykezelés (DOM Level 0)

## HTML elemen belül megadott eseménykezelő

```
<a href="http://uni.sze.hu"
    onClick="return confirm('Biztos?')">Egyetem</a>
```

Előny:

- könnyen, gyorsan elkészíthető

Hátrányok:

- a JS kód szétszórva helyezkedik el az oldalon → ez nehezíti a hibakeresést és a továbbfejlesztést
- nem valósítható meg vele a *fokozatos fejlődés* elve
- a JS mindig, mindenhol használható lesz?

Használata nem ajánlott, de régi dokumentumokban sok helyen fellelhető ez a módszer.

# Eseménykezelés klasszikus módszerrel (DOM Level 0)

## Klasszikus megadás – Fahrenheit-Celsius átváltó (részletek)

```
var urlap = document.getElementById("valtoUrlap");
urlap.onsubmit = atvalt;
...
window.onload = init;
```

## További lehetőségek

```
if(typeof window.onload == "function") {
  window.onload = function() {
    // Névtelen függvénnyel is megadható
  }
}
...
window.onload = null; // Függvény eltávolítása
```

# Eseménykezelés klasszikus módszerrel (DOM Level 0)

Előnyök:

- egyszerűen, könnyen megvalósítható
- gyakorlatilag minden böngészőben működik

Hátrányok:

- új eseménykezelő hozzáadása felülírja a régit

**Működik, de nem ajánlott!**

```
window.init = function() {
  első_eseménykezelő();
  második_eseménykezelő();
}
```

# W3C eseménykezelési módszer (DOM Level 2)

Előnyök:

- eseménykezelő fv. hozzárendelése JS kódból történik (vö. soron belüli megadás)
- több eseménykezelő is hozzárendelhető ugyanazon eseményhez (vö. klasszikus módszer)
- rugalmas, logikus, viszonylag egyszerű

Hátrányok:

- kicsit terjengősebb a kód
- IE9 előtti változatok nem támogatják

**Használat**

```
objektum.addEventListener(
    esemény_típusa, kezelő_függvény, fázis);
objektum.removeEventListener(
    esemény_típusa, kezelő_függvény, fázis);
```

# W3C eseménykezelési módszer (DOM Level 2)

## Minták

```
window.addEventListener(
  "load", első_eseménykezelő, false);
window.addEventListener(
  "load", második_eseménykezelő, false);
...
window.removeEventListener(
  "load", első_eseménykezelő, false);
```

Megjegyzések:

- a *fázis* értéke általában false (rákerüljön a vezérlés az eseménykezelőre az elfogási fázisban? Ld. később)
- csak pontosan egyező paraméterezés mellett lesz sikeres az eltávolítás
- nem létező eseménykezelő eltávolítása nem okoz hibát

# IE eseménykezelés

Problémák:

- IE9 előtti verziók nem támogatják a W3C módszerét
- de hasonló működésű függvények léteznek: attachEvent(), detachEvent().
- az esemény neve itt onnal kezdődik (pl. *onload* a *load* helyett)

Létre kell hoznunk egy általános eseménykezelőt beállító függvényt/metódust!

## Eseménykezelő hordozható beállítása

```
function addEsemeny(obj, tipus, fv) {
  if (obj && obj.addEventListener) { // W3C
    obj.addEventListener(tipus, fv, false);
  } else if (obj && obj.attachEvent) { // <IE9
    obj.attachEvent("on" + tipus, fv);
  }
}
```

# Eseménytípusok

A továbbiakban csak a leggyakrabban előforduló eseményekkel foglalkozunk. További részletek:Mozilla, W3Schools, . . .

Események típusai forrásuk szerint származhatnak:

- grafikus kurzort vezérlő eszköztől
- billentyűzettől
- böngésző programtól
- űrlapoktól

Nem minden böngésző támogatja valamennyi eseményt.
Kompatibilitási táblázat

# Grafikus kurzort vezérlő eszköztől származó események

Forrása lehet: egér, touchpad, trackball, tablet érintésérzékeny kijelzője, stb.

A(z egér) gombokkal kiváltható események:

| | |
|---|---|
| mousedown | Gombot lenyomták |
| mouseup | Gombot felengedték |
| click | Kattintottak, azaz a gombot lenyomták majd felengedték, és közben nem mozgatták át más elem fölé a kurzort; a *click* előtt a fenti két esemény is létrejön |
| drag | Húzd és ejtsd; a HTML5 előtt teljesen kézzel kellett megvalósítani |
| dblclick | Dupla kattintás; előzőleg két *click* is keletkezik, emiatt a *click* eseményre általában nem regisztrálnak |
| contextmenu | Környezeti/helyi menü előhívása |

# Grafikus kurzort vezérlő eszköztől származó események

A(z egér) mozgatásával kiváltható események:

| | |
|---|---|
| mouseover | Az egér valamilyen elem fölé került |
| mouseout | Az egér elhagyta az elem területét |
| mousemove | Az egeret mozgatják (erőforrásigényes) |

# Grafikus kurzort vezérlő eszköztől származó események

utilities.js

```
// http://www.larryullman.com/downloads/modern_javascript_scripts.zip
// Script 8.1 - utilities.js
// This script defines an object that has some utilitarian functions.
var U = {
    // For getting the document element by ID:
    $: function(id) {
        'use strict';
        if (typeof id == 'string') {
            return document.getElementById(id);
        }
    }, // End of $() function.
    // Function for setting the text of an element:
    setText: function(id, message) {
        'use strict';
        if ( (typeof id == 'string')
        && (typeof message == 'string') ) {
            // Get a reference to the element:
            var output = this.$(id);
            if (!output) return false;
            // Set the text
            if (output.textContent !== undefined) {
                output.textContent = message;
            } else {
                output.innerText = message;
            }
            return true;
        } // End of main IF.
    }, // End of setText() function.
```

# Grafikus kurzort vezérlő eszköztől származó események

utilities.js (folytatás)

```
    // Function for creating event listeners:
    addEvent: function(obj, type, fn) {
        'use strict';
        if (obj && obj.addEventListener) { // W3C
            obj.addEventListener(type, fn, false);
        } else if (obj && obj.attachEvent) { // Older IE
            obj.attachEvent('on' + type, fn);
        }
    }, // End of addEvent() function.
    // Function for removing event listeners:
    removeEvent: function(obj, type, fn) {
        'use strict';
        if (obj && obj.removeEventListener) { // W3C
            obj.removeEventListener(type, fn, false);
        } else if (obj && obj.detachEvent) { // Older IE
            obj.detachEvent('on' + type, fn);
        }
    } // End of removeEvent() function.
}; // End of U declaration.
```

# Grafikus kurzort vezérlő eszköztől származó események

**eger.html**

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>Egéresemények tesztelése</title>
  </head>
  <body>
    <div id="uzenet">Mozgasd ide az egeret!</div>
    <script src="utilities.js"></script>
    <script src="eger.js"></script>
  </body>
</html>
```

**eger.js**

```
function folotte() {
  'use strict';
  U.setText("uzenet", "Viszed innen!");
}
function mashol() {
  'use strict';
  U.setText("uzenet", "Mozgasd ide az egeret!");
}
window.onload = function() {
  'use strict';
  U.addEvent(U.$("uzenet"), "mouseover", folotte);
  U.addEvent(U.$("uzenet"), "mouseout", mashol);
};
```

# Billentyűzettől és böngészőtől származó események

Billentyűzettől származó események:

| | |
|---|---|
| keydown | Billentyű lenyomása |
| keyup | Billentyű felengedése |
| keypress | Egy karakter begépelése |

Váltóbillentyűk lenyomását (Shift, Alt, ...) általában nem jelzik eseménnyel a böngészők.

Böngésző események:

| | |
|---|---|
| load, unload | Valamilyen erőforrás betöltését/megsemmisítését jelzik |
| resize | Ablak átméretezése |
| scroll | Görgetés |
| copy, cut, paste | Vágólap kezelés, kevés böngésző támogatja |
| focus, blur | Fókusz megszerzése, elvesztése |

# Űrlaptól származó események

| | |
|---|---|
| reset | Az űrlapot alaphelyzetbe állítják; ritkán használt |
| change | Megváltozik az űrlapmező tartalma; szövegbeviteli mezőknél csak az elem elhagyásakor keletkezik, jelölőknél, rádiógomboknál, legördülő menüknél azonnal |
| select | Szöveges bevitelre alkalmas vezérlőknél a szöveg kijelölését jelzi |
| click | Jelölőnégyzetek, rádiógombok ezt is kezelik |
| focus, blur | Beviteli mező megkapja/elveszti a beviteli fókuszt |

Megjegyzés:
Fel kell készülni különféle eszközök támogatására. Pl. nincs minden gépen egér, ezért a *mouseover* mellett a *focus* eseményre is ugyanazokat a tevékenységeket kell elvégezni. Hasonlóan: *mouseout/blur*, űrlapoknál a gomb *click*-je helyett/mellett inkább a *submit*-et szerencsésebb kezelni (események párosítása).

# Eseményobjektum fogadása

Sokszor szükséges lenne az esemény körülményeinek ismerete is (pl. mire kattintottak?). Ha az `addEventListener()` támogatott, akkor az eseménykezelő első és egyetlen paramétere az esemény objektum lesz, amit automatikusan megkap. Sajnos az IE8-ig csak az `attachEvent()`, `detachEvent()` használható, és az eseményt a `window.event` objektum jellemzi.

## Megoldások hordozható eseménykezelő készítéséhez

```
function eseményKezelő(e) {
  // Első megoldás
  if (typeof e == "undefined") e = window.event;
  // Második megoldás
  e = e || window.event;
  // Harmadik megoldás
  if (!e) e = window.event;
  // Tényleges eseménykezelés, e használatba vétele
}
```

Probléma egyedül a soron belüli eseménykezelésnél léphet fel (nekünk kell paramétereznünk), de ilyet nem használunk, ugye?

# Eseményobjektum tulajdonságai

Az eseményobjektumból kiolvashatóak az esemény létrejöttének körülményei, de sajnos az objektum felépítése böngészőfüggő.

target/srcElement (W3C/<IE9) az eseményt létrehozó elem referenciája (`forrás.nodeName` megadja az elem nevét, pl. DIV, INPUT)

type az esemény típusa (pl. click, keydown)

Ha billentyű leütése váltotta ki az eseményt:

keyCode a billentyűt azonosító UNICODE kód, IE: ha `keypress` esemény keletkezett, akkor ebben a karakter UNICODE kódját tárolja

which a begépelt karakter kódja, IE: nem támogatott

Kódból karaktert előállítani a `String.fromCharCode()` metódussal lehet.

**Karakterkód hordozható kinyerése keypress esetén**

```
var karakterKód = e.which || e.keyCode;
```

# Eseményobjektum tulajdonságai

Ha billentyű leütése váltotta ki az eseményt:

| | |
|---|---|
| shiftKey | Logikai érték, az esemény közben a *Shift*-et nyomva tartották-e |
| ctrlKey | Logikai érték, az esemény közben a *Ctrl*-t nyomva tartották-e |
| altKey | Logikai érték, az esemény közben a *Alt*-ot (Mac: *Option*) nyomva tartották-e |

Ha egér váltotta ki az eseményt:

| | |
|---|---|
| which, button | A lenyomott/felengedett gomb azonosítója. Csak az a biztos, hogy a jobb gomb száma 2, és a *click* esetén eltérően működnek a böngészők. |
| relatedTarget | W3C; *mouseover* esetén az előző elemet, *mouseout* esetén az új elemet adja meg, ami fölé a kurzor kerül |
| fromElement, toElement | ugyanez IE-ben |

# Eseményobjektum tulajdonságai

## Hordozható megoldás

```
function mouseOverKezelo(e) {
        if (!e) var e = window.event;
        var elem = e.relatedTarget || e.fromElement;
}

function mouseOutKezelo(e) {
        if (!e) var e = window.event;
        var elem = e.relatedTarget || e.toElement;
}
```

# Eseménykezelés teszt

esemeny.html

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>Események vizsgálata</title>
  </head>
  <body>
    <form action="#" id="urlap">
      <fieldset>
        <legend>Válassza ki a tesztelendő eseményeket!</legend>
        <div><input type="checkbox" id="mouseover">
          <label for="mouseover">mouseover</label></div>
        <div><input type="checkbox" id="mouseout">
          <label for="mouseout">mouseout</label></div>
        <div><input type="checkbox" id="click">
          <label for="click">click</label></div>
        <div><input type="checkbox" id="keypress">
          <label for="keypress">keypress</label></div>
        <div><input type="checkbox" id="blur">
          <label for="blur">blur</label></div>
        <div><textarea id="naplo" rows="10" cols="40"></textarea></div>
        <div><input type="submit" id="gomb" value="Érvényesít"></div>
      </fieldset>
    </form>
    <script src="utilities.js"></script>
    <script src="esemeny.js"></script>
  </body>
</html>
```

# Eseménykezelés teszt

esemeny.js

```
function naplozas(e) {
  'use strict';
  if(typeof e == "undefined") {
    e = window.event;
  }
  var cel = e.target || e.srcElement;
  var uzenet = cel.nodeName + ": " + e.type + "\n";
  U.$("naplo").value += uzenet;
}

function setEsemenykezelok(e) {
  'use strict';
  var esemenyek = ["mouseover", "mouseout", "click", "keypress", "blur"];
  for(var i=esemenyek.length-1; i>=0; i--) {
    var jelolo = U.$(esemenyek[i]);
    if(jelolo.checked) {
      U.addEvent(document, esemenyek[i], naplozas);
    } else {
      U.removeEvent(document, esemenyek[i], naplozas);
    }
  }
  U.$("naplo").value = "";
  return false;
}

window.onload = function() {
  'use strict';
  U.$("urlap").onsubmit = setEsemenykezelok;
};
```

# Böngésző alapértelmezett viselkedésének megváltoztatása

Általában nem cél megváltoztatni a böngésző viselkedését, de néha szükséges: pl. hibás adatokat tartalmazó űrlapot nem küldünk el. Ezt tettük eddig is, az eseménykezelő végén lévő `return false;` segítségével.
Gond: csak akkor működik, ha klasszikus módszerrel állítottuk be az eseménykezelőt!
Mit lehet tenni?

- `e.preventDefault()`, ha a böngésző támogatja
- `e.returnValue = false`, <IE9 esetén

## Hordozható megoldás

```
if (typeof e == "undefined") e = window.event;
if (e.preventDefault) {
  e.preventDefault();
} else {
  e.returnValue = false;
}
return false; // Biztos, ami tuti
```

# Böngésző alapértelmezett viselkedésének megváltoztatása

Különbségek a `false` visszatérési érték alkalmazásához képest:

- a `return` azonnal megszakítja a fv. futását, az új módszerek nem
- `false` esetén a többi regisztrált eseménykezelő hívása ugyanazzal az esemény objektummal már nem történik meg
- a `false` megszakítja az esemény *bubbling* fázisát

# Böngésző alapértelmezett viselkedésének megváltoztatása

teglatest.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Téglatest</title>
  </head>
  <body>
    <form action="#" id="urlap">
      <fieldset>
        <legend>Adja meg a méreteket!</legend>
        <div><label for="szelesseg">Szélesség</label>
          <input type="text" id="szelesseg"></div>
        <div><label for="magassag">Magasság</label>
          <input type="text" id="magassag"></div>
        <div><label for="melyseg">Mélység</label>
          <input type="text" id="melyseg"></div>
        <div><input type="submit" id="szamol" value="Számol"></div>
      </fieldset>
    </form>
    <p id="eredmeny"></p>
    <script src="utilities.js"></script>
    <script src="teglatest.js"></script>
  </body>
</html>
```

# Böngésző alapértelmezett viselkedésének megváltoztatása

teglatest.js

```
function szamol(e) {
  'use strict';
  if(typeof e == "undefined") e = window.event;
  var szelesseg = U.$("szelesseg").value;
  var magassag = U.$("magassag").value;
  var melyseg = U.$("melyseg").value;
  if(isNaN(szelesseg) || isNaN(magassag) || isNaN(melyseg)) {
    U.setText("eredmeny", "Adjon meg helyes adatokat!");
  } else {
    var felszin = (szelesseg*magassag + magassag*melyseg +
      melyseg*szelesseg) * 2;
    var terfogat = szelesseg * magassag * melyseg;
    U.setText("eredmeny", "Felszín: " + felszin.toFixed(2) +
      ", térfogat: " + terfogat.toFixed(2));
  }
  if (e.preventDefault) {
    e.preventDefault();
  } else {
    e.returnValue = false;
  }
}
window.onload = function() {
  'use strict';
  U.addEvent(U.$("szelesseg"), "change", szamol);
  U.addEvent(U.$("magassag"), "change", szamol);
  U.addEvent(U.$("melyseg"), "change", szamol);
  U.addEvent(U.$("urlap"), "submit", szamol);
};
```

# Eseménykezelési fázisok

Capturing:
[window] → [document] → [body] → [div] → [p] → [a]
Bubbling:
[window] ← [document] ← [body] ← [div] ← [p] ← [a]
Az esemény akkor is elkapható a *bubbling* miatt, ha az esemény az *a*-n keletkezik, de a kezelő pl. a *div*-hez van bejegyezve. A `target/srcElement` a valódi eseményforrást fogja jelölni.

## Bubbling kikapcsolása

```
if (e.stopPropogation) { // W3C
  e.stopPropogation();
} else { // <IE9
  e.cancelBubble = true;
}
```

A fv. végi `return false;` is megakadályozza a bubbling-ot, de egyéb mellékhatásai is vannak.

# Eseménykezelési fázisok

A böngészők alapértelmezés szerint a bubbling fázisban juttatják el az eseményt a kezelőhöz.
W3C: az `addEventListener()` utolsó paraméterének `true` értéke esetén a capture fázisban kerülnek az események a kezelőhöz.
Nincs minden eseménynek bubble fázisa: focus, blur, change, scroll, submit.

Események delegálása: az eseménykezelőt a szülő elemhez kötik, és minden gyermek elemeknél keletkezett eseményt itt dolgoznak fel
$\rightarrow$ gyors

További információ

# Beépített dialógusablakok

Típusok:

| | |
|---|---|
| `alert(szöveg)` | Szöveg megjelenítés + OK gomb |
| `confirm(szöveg)` | Szöveg megjelenítés + OK/Cancel gombok<br>Logikai értékkel tér vissza (OK = `true`, Cancel = `false`). |
| `prompt(szöveg, alapérték)` | Szöveg megjelenítés + egysoros szövegbeviteli mező + OK/Cancel gombok<br>Visszatérési érték: `String`, ha az OK-ra kattintottak, `null`, ha a Cancel-re |

Tulajdonságaik:

- Nem formázhatóak HTML/CSS-sel, nem bővíthetőek további vezérlőkkel (csak a sortörés engedélyezett `\n`-nel)
- Ahány böngésző, annyiféle megjelenés
- modálisak

# Beépített dialógusablakok

## Dialógusablak teszt

```
function dialogus() {
  var ital = prompt("Mi a kedvenc italod?", "sör");
  if(ital === null) {
    alert("Mi van veled, fáj a fejed?");
  } else {
    var ivas = confirm("Akkor meghívsz egy "+ital+"re?");
    if(ivas) {
      alert("Haver vagy.");
    } else {
      alert("OK, semmi baj.\nLegközelebb is ráérsz.");
    }
  }
}
```

# A window objektum

A window objektum a legmagasabb szintű objektum a JS-ben, így minden objektumon kívül definiált objektum, függvény és változó közvetlenül ebbe kerül.

## A window objektum elemei

```
function fv() {
  console.log("Működik!");
}
fv();
window.fv();

var valt = 3;
console.log(valt);
console.log(window.valt);

alert("Hahó!");
window.alert("Hahó!")
```

# A window objektum

Megjegyzések:

- Nem szerencsés „teleszemetelni” a window objektumot rengeteg adattaggal → saját „csomagoló” objektumok létrehozása
- Sok eddig látott objektum is a window része: Math, Date, ...
- Adattagjain keresztül lekérdezhető (néha még be is állítható) a böngészőablak néhány eleme. Ma már ritkán használják ezeket a lehetőségeket.

## A window érdekes adattagjai

```
if(window.menubar.visible) { ... // Menüsor látható
if(window.toolbar.visible) { ... // Gombsor látható
window.status = "Módosítjuk a böngésző állapotsorát";
  // Ha sikerül...
console.log(window.navigator.userAgent);
  // Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:16.0)
  // Gecko/20100101 Firefox/16.0
```

# A böngészőablak mérete és helye

`screenX` / `screenLeft` Bal felső sarok X koordinátája (*/IE)

`screenY` / `screenTop` Bal felső sarok Y koordinátája (*/IE)

`innerHeight` / `document.body.clientHeight` Kliens terület magassága (*/<IE9)

`innerWidth` / `document.body.clientWidth` Kliens terület szélessége (*/<IE9)

`outerHeight` Böngészőablak magassága

`outerWidth` Böngészőablak szélessége

`moveTo(x, y)` Böngészőablak mozgatása adott helyre

`moveBy(x, y)` Böngészőablak elmozgatása adott képpontnyival

# A képernyő tulajdonságai

`window.screen. ...`

| | |
|---|---|
| `height` | Képernyő magassága (px) |
| `width` | Képernyő szélessége (px) |
| `availHeight` | Felhasználható magasság (px) |
| `availWidth` | Felhasználható szélesség (px) |
| `colorDepth` | Színmélység (bit) |
| `pixelDepth` | Színmélység (bit) |

# Új böngészőablak létrehozása

```
var ablak = window.open(url, név, tulajdonságok);
```

| | |
|---|---|
| `url` | Ezt fogja tartalmazni az új ablak |
| `név` | Az ablak neve, `window.name` |
| `tulajdonságok` | Új ablak tulajdonságai később nem módosíthatóak. Formátum: tulajdonságnév=érték,... Az „érték” csak egész szám, `yes` vagy `no` lehet. |

Fontosabb tulajdonságok:

| | |
|---|---|
| `height` | Kliens terület magassága |
| `width` | Kliens terület szélessége |
| `outerHeight` | Az ablak magassága |
| `outerWidth` | Az ablak szélessége |
| `top` | Új ablak tetejének szülőhöz képesti helye |
| `left` | Új ablak bal oldalának szülőhöz képesti helye |
| `location` | A böngésző eszköztár láthatósága |
| `menubar` | A menüsor láthatósága |
| `scrollbars` | Gördítősávok láthatósága |
| `status` | Állapotsor láthatósága |
| `toolbar` | Eszköztár láthatósága |
| `resizable` | Átméretezhetőség |

Visszatérési érték: az új ablak objektum, vagy `null`, ha nem jött létre.

# Új böngészőablak létrehozása

Bezárás:

`window.close()`

Csak a kódból megnyitott ablakon működik.

## Új ablak nyitása és bezárása

```
var sze = open("http://uni.sze.hu", "sze",
  "height=200,width=200,location=no," +
  "resizable=yes,scrollbars=yes");
alert("Kattintson, ha bezárhatjuk.");
if((sze !== null) && (!sze.closed)) {
    sze.close();
}
```

Másik ablakra fókuszálás:

`window.focus()`

# Az új ablak elérése

Az új ablak csak JS segítségével elérhető, viszont

- nincs mindig bekapcsolva a JS,
- egyszerű böngészők, internetes keresők sem futtatnak JS-et, ...

Megoldás: az ablak megnyitását tegyük lehetővé hivatkozással is, majd definiáljunk eseménykezelőt a hivatkozás `click` eseményére!
JS ki/be: URL: about:config / „Óvatos leszek, megígérem!” / javascript.enabled → false (Firefox 25)

ablak.html

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>Felugró ablakok</title>
  </head>
  <body>
    <a href="felugro.html" id="felugro" target="felugro_ablak">
      Kattintson ide új ablak megnyitásához!</a>
    <script src="ablak.js"></script>
  </body>
</html>
```

# Az új ablak elérése

**felugro.html**

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>Új ablak</title>
  </head>
  <body>
    <p>Hurrá, sikerült!</p>
  </body>
</html>
```

**ablak.js**

```
function kattintas() {
  'use strict';
  var felugro = open("felugro.html", "felugro_ablak",
    "height=200,width=200,top=100,left=100");
  if ((felugro !== null) && !felugro.closed) {
    felugro.focus();
    return false;
  }
}
window.onload = function() {
  'use strict';
  document.getElementById("felugro").onclick = kattintas;
};
```

# Ablakok közötti kommunikáció

A szülő és gyermek ablakok kölcsönösen elérik egymás globális `window` objektumait, és ezen keresztül kommunikálhatnak egymással.

Szülő ismeri a gyereket: `var gyerek = open("gyerek.html");`
Gyerek ismeri a szülőt: `var szulo = window.opener;`

Ezután hívhatják egymás metódusait, használhatják egymás változóit, stb.

```
// Szülőben:
gyerek.window.feldolgoz(
  document.getElementById("urlapmezo").value);
// Gyerekben:
szulo.window.document.getElementById(
  "bekezdes").textContent = eredmeny;
```

# Ablakok közötti kommunikáció

Csak az azonos helyről származó ablakok kommunikálhatnak!

Pl. `http://uni.sze.hu`

Eltérőek a helyek, ha:

- a protokoll nem egyezik, pl. `https://uni.sze.hu`
- más aldomainen belülről származnak az oldalak, pl. `http://it.sze.hu`
- más portot használnak, stb.

# IFrame és ablak közti kommunikáció

Hasonlóan zajlik a kommunikáció a felek között, mint az ablakok esetében.

Szülő ismeri a keretet:
`var keret = document.getElementById("keretId");` vagy
`var keret = window.frames[keretIndex];` vagy
`var keret = window.frames["keretNév"];`
(ez tehát a `name` attribútum, nem az `id`!) majd
`var ablak = keret.contentWindow || keret.contentDocument;`

Gyerek ismeri a szülőt: `var szulo = window.parent;`
Van szülője? `if(window.parent != window.self) { ...`

A HTML5-ben új lehetőségek jelentek meg.

# Ugrás az előzmények között

A vissza/előre lépésre ad lehetőséget (ha megvalósítható) a `window.history`.

| | |
|---|---|
| `go()` | ami adott számú oldallal vissza/előre lépteti a böngésző előzményeket (az aktuális oldal indexe 0) |
| `back()` | előző oldalra lép, azaz `go(-1)` |
| `forward()` | következő oldalra lép, azaz `go(1)` |

A HTML5 új lehetőségeket biztosít az előzmények kezeléséhez (AJAX alkalmazások használatának megkönnyítése érdekében).

Teljesen nem valósítható meg a *fokozatos fejlődés* elve.

# Átirányítás

Szerencsésebb a webszerverrel megoldani (nem kell letölteni egy oldalt a másik oldal letöltéséhez), de néha szükség lehet a böngésző átirányítására.

Lehetőségek:

- `window.location = "http://uni.sze.hu";`
- `window.location.href = "http://uni.sze.hu";`
  Mindkét esetben módosulnak az előzmények. Ha ezt el akarjuk kerülni:
- `window.location.replace("http://uni.sze.hu");`
- Kezdő oldal: `window.home();`
- Lekérdező karakterlánc: `window.location.search`
- Horgony: `window.location.hash`

# Nyomtatás

Oldal kinyomtatása: `window.print();`
Nem minden HW támogatja (okostelefon?)

**nyomtat.html (részlet)**

```
<body>
  <a href="#" id="nyomtat">Nyomtass, az ördögit!</a>
  <script src="nyomtat.js"></script>
</body>
```

**nyomtat.js**

```
window.onload = function() {
  'use strict';
  if(typeof window.print == "function") {
    document.getElementById("nyomtat").onclick = function() {
      window.print();
    }
  }
};
```

# A document objektum néhány tulajdonsága

A `document` objektum reprezentálja a böngészőbe töltött weboldalt. Néhány metódusa:

`getElementById()` lekér a DOM-ból egy `id`-vel azonosított objektumot

`write()` hozzáír a dokumentum forrásához egy sort

`writeln()` mint a `write()`, de sort is emel

Utóbbiak használata nem javasolt, mert

- XHTML dokumentumokkal nem használható
- véletlenül felülírhatjuk a már betöltött dokumentumot
- módosítja a DOM-ot, bár ha ez a szándékunk, arra vannak jobb megoldások

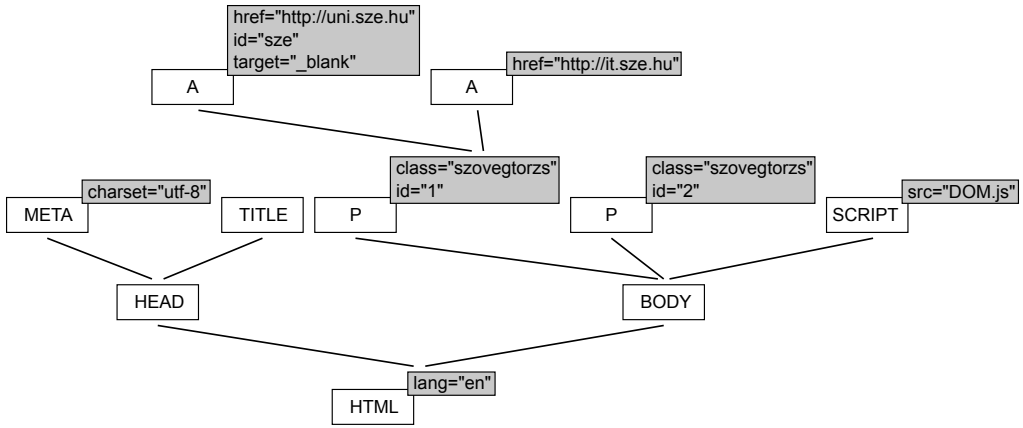
Adattagok:

`title` a böngésző fejlécében megjelenő cím

`compatMode` a böngésző kompatibilitási módjának állapota (DOCTYPE rendben?), értékek: *BackCompat*, *CSS1Compat*.

# Document Object Model, DOM

DOM: W3C 1998-ban szabványosította. XML/XHTML/HTML adatok elérése, feldolgozása.




## DOM.html

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>DOM példa</title>
  </head>
  <body>
    <p class="szovegtorzs" id="1">Kattintson ide az
      <a href="http://uni.sze.hu" id="sze" target="_blank">egyetemi oldal</a>
      megnyitásához! Vagy nézegesse inkább a
      <a href="http://it.sze.hu">tanszéki</a> oldalakat!</p>
    <p class="szovegtorzs" id="2">További kellemes időtöltést!</p>
    <script src="DOM.js"></script>
  </body>
</html>
```

# DOM objektumok elérése

Fa struktúra: csúcsok (node) és élek (az ábrán a text csúcsok nincsenek feltüntetve)
Gyökér: HTML
Minden csúcs egy objektum, adattagjaik:

| | |
|---|---|
| `parentNode` | szülő csúcs |
| `childNodes` | gyerekek tömbje |
| `children` | csak a HTML elemek tömbje |
| `firstChild` | első gyerek |
| `lastChild` | utolsó gyerek |
| `nextSibling` | következő testvér |
| `previousSibling` | előző testvér |
| `nodeName` | az aktuális csúcs neve (pl. P, A, #text, #document) |
| `nodeValue` | csúcs értéke* |
| `nodeType` | csúcs típusa |

# DOM objektumok elérése

`nodeType` lehetséges értékei:

- 1 HTML elem
- 2 text
- 8 megjegyzések
- 9 document
- 10 a HTML gyökérelem*

`document.documentElement`: a HTML gyökérelemet adja meg
Tömbök az egyes elemtípusok gyorsabb eléréséhez: `document.`

- `forms` űrlapok
- `images` képek
- `links` hivatkozások

Indexelhetők számmal, vagy a `name` tulajdonság értékével
`document.head`, `document.body`

# DOM objektumok elérése

**Példák**

```
document.body.parentNode;
  // <html lang="hu">
document.body.childNodes;
  // [<TextNode textContent="\n  ">, p#1.szovegtorzs, ...
document.body.children;
  // [p#1.szovegtorzs, p#2.szovegtorzs, script DOM.js]
document.body.firstChild;
  // <TextNode textContent="\n ">
document.body.nextSibling;
  // null
document.body.previousSibling;
  // <head>
document.body.nodeName;
  // "BODY"
document.body.nodeType;
  // 1
document.documentElement;
  // <html lang="hu">
document.links;
  // [a#sze, a]
document.links[0];
  // <a id="sze" target="_blank" href="http://uni.sze.hu">
```

# DOM objektumok elérése

Elemek gyorsabb elérése:

- `document.getElementById()` adott `id`-jű elem objektumát adja, vagy `null`-t
- `getElementsByTagName()` bármilyen DOM objektumból hívható, a DOM fa ezen ágán lévő adott HTML elemek objektumaiból álló tömböt ad vissza
- `getElementsByClassName()` ilyen osztályú elemek tömbjével tér vissza (nem támogatott <IE9 esetén)

**Példák**

```
document.getElementById("1");
  // <p id="1" class="szovegtorzs">
document.body.getElementsByTagName("a");
  // [a#sze, a]
document.body.getElementsByClassName("szovegtorzs");
  // [p#1.szovegtorzs, p#2.szovegtorzs]
```

# DOM objektumok elérése

CSS szelektorok használata is lehetséges (IE8+, IE8: CSS2.1)
Bármely DOM objektumon hívható metódusok további objektumok eléréséhez:

- `querySelector()` az első illeszkedő objektumot adja
- `querySelectorAll()` minden illeszkedő objektumot visszaad

**Példák**

```
document.body.querySelector("p.szovegtorzs");
  // <p id="1" class="szovegtorzs">
document.body.querySelectorAll("p.szovegtorzs");
  // [p#1.szovegtorzs, p#2.szovegtorzs]
document.body.querySelector(".szovegtorzs");
  // <p id="1" class="szovegtorzs">
document.body.querySelector("p a");
  // <a id="sze" target="_blank" href="http://uni.sze.hu">
document.body.querySelector("p > a");
  // <a id="sze" target="_blank" href="http://uni.sze.hu">
```

# DOM objektumok módosítása

Az objektumok attribútumainak értékei módosíthatóak, pl.

**Adattagok módosítása**

```
document.getElementById("sze").href = "http://rs1.sze.hu";
document.getElementById("1").id = "3";
```

De a `class` és `for` szavak foglaltak a JS-ben, ezért

**class, for**

```
document.getElementById("1").className = "alap";
document.getElementById("cimke").htmlFor = "ujelem"; //*
```

# DOM objektumok módosítása

Szöveges tartalom módosítása:

- `innerText` FireFox: nem megy, de ált. a többiben igen
- `textContent` W3C szabvány, IE: nem megy

**textContent**

```
document.getElementById("1").textContent;
// "Kattintson ide az egyetemi oldal megnyitásához!
// Vagy nézegesse inkább a tanszéki oldalakat!"
```

HTML tartalom módosítása:

- `innerHTML` gyak. minden böngészőben működik

**innerHTML**

```
document.getElementById("1").innerHTML;
  // "Kattintson ide az
  // <a href="http://rs1.sze.hu" id="sze" target="_blank">
  // egyetemi oldal</a>megnyitásához! Vagy nézegesse inkább a
  // <a href="http://it.sze.hu">tanszéki</a> oldalakat!"
```

# DOM objektumok létrehozása

A módszer lassabb, mint az `innerHTML` alkalmazása, de néha elkerülhetetlen.

1. létrehozás: `document.createElement(típus)` új DOM objektumot hoz létre, ami nem része a fának
2. adattagok módosítása: `innerHTML`, `textContent`, ...
3. beszúrás a fába:
   `insertBefore(gyerek, testvér)` testvér elé helyezi
   `appendChild(gyerek)` az utolsó testvér lesz
   `replaceChild(gyerek, régi_gyerek)` testvér lecserélése

## HTML elemek készítése

```
var bekE = document.createElement("p");
bekE.className = "szovegtorzs";
bekE.id = "legelso";
bekE.textContent = "Ez lesz a legelső bekezdés!";
document.body.insertBefore(bekE, document.getElementById("1"));

var bekK = document.createElement("p");
bekK.id = "kozepso";
bekK.textContent = "Ez lesz a középső bekezdés!";
document.body.replaceChild(bekK, document.getElementById("2"));

var bekU = document.createElement("p");
bekU.id = "legutolso";
bekU.textContent = "Ez lesz a legutolsó bekezdés!";
document.body.appendChild(bekU);
```

# DOM objektumok létrehozása

További lehetőségek:

- szöveg csúcs létrehozása: `document.createTextNode("szöveg")` nem módosítja a többi csúcsot, mint egy `innerText/textContent` hívás
- csúcs másolása: `objektum.cloneNode()` nagy részfákat érdemesebb lemásolni, majd visszarakni a fába, mint minden részét külön-külön módosítgatni, mert elég 1x frissíteni a képernyőt
- gyerek eltávolítása: `szulo.removeChild(gyerek)`
- szülő lekérdezése: `gyerek.parentNode`

**HTML elemek másolása, törlése**

```
var duma1 = document.createTextNode("új szöveg");
var duma2 = duma1.cloneNode();
var egy = document.getElementById("1");
egy.insertBefore(duma1, document.getElementById("sze"));
egy.appendChild(duma2);

var ketto = document.getElementById("2");
ketto.parentNode.removeChild(ketto);
```

# Stíluslapok használata

A DOM objektumok `style` objektumán keresztül módosítható a formázás, de

- mindig camelCase formázást kell használni
- mindig meg kell adni a mértékegységet
- soron belüli formázást eredményez (mint a `style` attribútum)

**Hivatkozás méretének növelése**

```
document.getElementById("sze").style.fontSize = "16pt";
```

Az alkalmazott stílusok kinyerhetőek:

- IE: `objektum.currentStyle`
- Hordozható: `window.getComputedStyle(objektum)`

Az eredmény csak olvasható.

**Hivatkozás méretének lekérdezése**

```
window.getComputedStyle(document.getElementById("sze")).fontSize;
  // "21.3333px"
```

# Elemek eltüntetése, megjelenítése

Kétféleképpen lehetséges:

1. `visibility` stílussal: nem tördeli újra az oldalt. Lehetséges értékek: `hidden`, `visible`.
2. `display` stílussal: újratördeli az oldalt. Lehetséges értékek pl.: `inline`, `block`, `none`.

## Láthatóság módosítása

```
var stilus = document.getElementById("1").style;
stilus.visibility = "hidden";
stilus.visibility = "visible";
stilus.display = "none";
stilus.display = "block";
```

# Stíluslapok kezelése

stilus.html

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" id="stilus" href="piros.css">
    <title>Stíluslap példa</title>
  </head>
  <body>
    <p>Milyen színű a szöveg?</p>
  </body>
</html>
```

piros.css

```
p {color: red}
```

kek.css

```
p {color: blue}
```

# Stíluslapok kezelése

## Stíluslap cseréje, törlése

```
var stilus = document.getElementById("stilus");
stilus.href = "kek.css"; // Most kék a szöveg
stilus.disabled = true; // Most fekete
```

## Új szabály beszúrása

```
var elem = document.getElementById("stilus");
var stilus = elem.sheet ? elem.sheet : elem.styleSheet;
if (stilus.insertRule) { // legtöbb böngésző
  stilus.insertRule("p {font-size: 20pt}", 0);
} else { // <IE9
  if (stilus.addRule) {
    stilus.addRule("p", "font-size: 20pt", 0);
  }
}
```

Törlés hasonlóan: `stilus.deleteRule(index)` vagy `stilus.removeRule(index)`

# Stíluslapok kezelése

## Stílus készítése

```
var stilus = document.createElement("style");
if(stilus.textContent !== "undefined") {
    stilus.textContent = "p {font-size: 20pt;}";
} else {
    stilus.innerText = "p {font-size: 20pt;}";
}
document.body.appendChild(stilus);
```

# Sütik

A NetScape fejlesztése. HTTP állapotmentes protokoll → állapotok tárolása lehetséges:

- munkamenetekkel a szerver oldalon (session)
- sütikkel kliens oldalon (cookie)

Sütik tulajdonságai:

- böngésző tárolja őket
- visszaküldi annak az oldalnak, ahonnét kapta
- kis méretűek (max. kb. 4kB)
- a felhasználó letilthatja használatukat
- a felhasználó olvashatja, alakíthatja tartalmát → érzékeny adatok tárolására alkalmatlan

Sütik által tárolt adatok:

- név
- érték
- érvényesség lejártának dátuma és ideje
- elérési út, amelyen érvényes (alapért.: aktuális út)
- tartomány (domain)

## Sütik készítése

Egy süti kulcs-érték pár: `var suti = "color=red";`
Süti hozzárendelése az oldalhoz: `document.cookie = suti;`
Újabb hozzárendelés nem törli az eddigi sütiket!

```
document.cookie = "fontSize=20pt";
console.log(document.cookie);
 // color=red; fontSize=20pt
```

A sütik közé néhány böngésző szóközt is rak!
Süti tulajdonságait ;-k választják el egymástól:

```
document.cookie = "color=red;expires="+
    new Date(2012, 10, 30).toUTCString()+
    ";path=/mappa;domain=*.sze.hu";
```

A néven és értéken kívül más tulajdonság nem olvasható a `document.cookie` használatával!

# Sütik használata

A sütik felhasználása a `document.cookie` változó feldolgozásával lehetséges.

## Sütik olvasása

```
function suti() {
  var sutik = document.cookie.split(";");
  var so = {
    nev: [],
    ertek: []
  };
  for(var i=sutik.length-1; i>=0; i--) {
    var akt = sutik[i].split("=");
    so.nev.push(akt[0].trim());
    so.ertek.push(akt[1]);
  }
  return so;
}
console.log(suti());
```

## Sütik törlése

A sütik azonos nevű, de érték nélküli, a múltban lejáró süti létrehozásával törölhetőek.

```
document.cookie =
    "fontSize=;expires=Thu, 01-Jan-1970 00:00:01 GMT";
```

## Süti könyvtár készítése

Praktikus, ha sokat kell dolgozni sütikkel. Ld. ©Larry Ullman, `cookies.js`

# Sütik használata

suti.html

```
<!doctype html>
<html lang="hu">
<head>
    <meta charset="utf-8">
    <link rel="stylesheet" id="szin" href="piros.css">
    <title>Süti példa</title>
</head>
<body>
    <p>Válasszon bekezdésszínt:
      <a href="#" id="piros">pirosat</a>
      vagy <a href="#" id="kek">kéket</a>.</p>
    <script src="cookies.js"></script>
    <script src="suti.js"></script>
</body>
</html>
```

# Sütik használata

suti.js

```
function beallit(e) {
    'use strict';
    if (typeof e == "undefined") e = window.event;
    var elem = e.target || e.srcElement;
    var lejar = new Date();
    lejar.setDate(lejar.getDate() + 7);
    COOKIE.setCookie("szin", elem.id, lejar);
    document.getElementById("szin").href = elem.id + ".css";
    return false;
}

window.onload = function() {
    'use strict';
    document.getElementById("piros").onclick = beallit;
    document.getElementById("kek").onclick = beallit;
    var szin = COOKIE.getCookie("szin");
    if (szin) {
        document.getElementById("szin").href = szin + ".css";
    }
};
```

# Időzítők

Nincs garancia a pontos időzítésre!

Egyszeri esemény: `var t = setTimeout(fv, kesleltetes_ms);`
Ismétlődő esemény: `var t = setInterval(fv, varakozas_ms);`
Időzítő törlése: `clearInterval(t);`

Feladat: pontos idő kijelzése, képernyő rendszeres frissítésével.

# Időzítők

ido.html

```
<!DOCTYPE html>
<html lang="hu">
  <head>
    <meta charset="utf-8">
    <title>Pontos idő kijelzése</title>
  </head>
  <body>
    <p>A pontos idő:<span id="ido"></span></p>
    <script src="ido.js"></script>
  </body>
</html>
```

# Időzítők

ido.js

```
function frissit() {
  'use strict';
  var cel = document.getElementById("ido");
  if(cel.textContent !== "undefined") {
    cel.textContent = new Date().toLocaleTimeString();
  } else {
    cel.innerText = new Date().toLocaleTimeString();
  }
}

window.onload = function() {
  'use strict';
  setInterval(frissit, 1000);
};
```