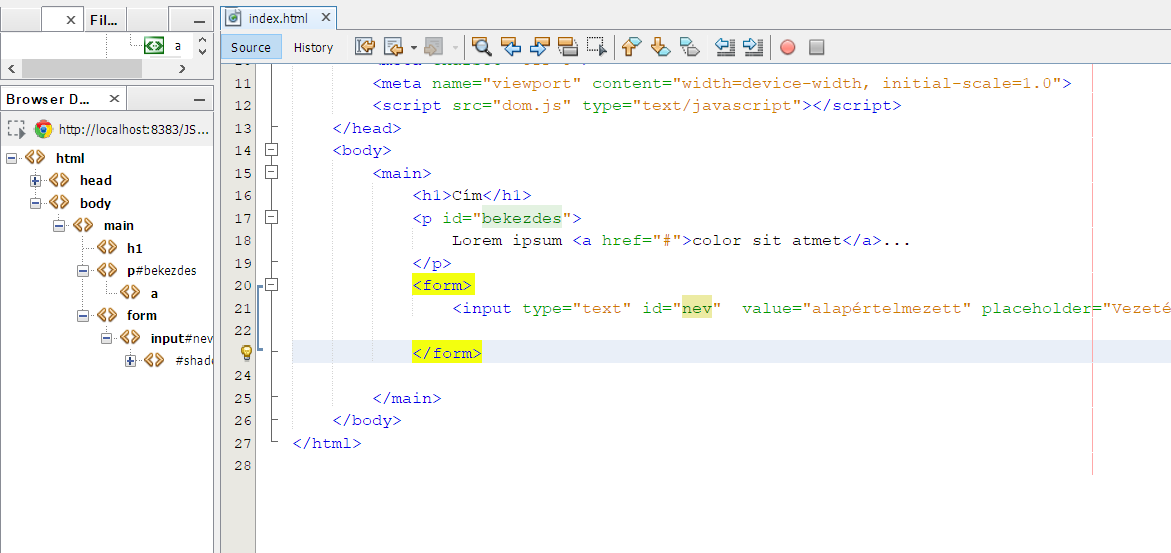
# Hogy érjük el a HTML dokumentum egyes elemeit?

## A Dokumentum Objektum Modell - DOM

A HTML elemekhez való hozzáférés JavaScriptben egy szabványos interfészen, a Dokumentum Objektum Modellen (röviden DOM) keresztül történik.

A HTML dokumentum egy fastruktúra, ahol HTML elemek egymás mellett vagy egymásba ágyazva jelennek meg.

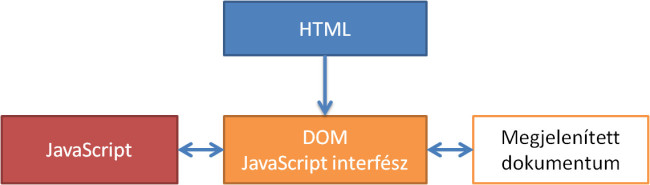


A szöveges HTML állomány betöltésekor a böngésző minden egyes HTML elemnek (még a szövegeseknek is) létrehoz egy JavaScript objektumot, és ezeket az objektumokat ugyanolyan fastruktúrába rendezi, mint ahogy az az eredeti HTML elemeknél volt. A folyamat eredményeképpen kialakul egy objektumhierarchia, ez a DOM.

[Live DOM viewer-ben](http://software.hixie.ch/utilities/js/live-dom-viewer/) nézve a DOM szerkezet:



A böngésző az oldal megjelenítését a DOM alapján végzi el. A DOM és a kirajzolt oldal között nagyon szoros kapcsolat van. Ha változik a DOM, akkor az az oldalon azonnal érvényesül.



DOM minden egyes HTML elemről rengeteg információt tárol egy-egy JavaScript objektum képében, és nagyon sokféle műveletet enged elvégezni velük. Ezekkel gyakorlatilag az oldal tetszőleges megváltoztatása lehetséges. A főbb műveletcsoportok:

* a faszerkezet bejárása,
* új elemek létrehozása,
* meglévők módosítása és törlése.

**Összegezve**: **Dokumentum Objektum Modell (DOM)** nem más, mint az oldal szerkezetéért felelős HTML elemek JavaScript objektum reprezentánsai, amik ugyanolyan fastruktúrába szervezve a megjelenítés alapját képezik, és amin keresztül a dokumentum programozhatóvá válik. **Magyarán szólva a DOM nem más, mint az oldal szerkezetének programozási felülete.** Rajta keresztül az egész dokumentum a legkisebb részletig tetszőlegesen bejárható és manipulálható.

## JavaScript fájl csatolása és futtatása

A JavaScript fájlra való hivatkozás helye a HTML-ben a head tagek között van, vagy közvetlen a </body> előtt. A javaScript fájlt script tag-gel csatolhatjuk az oldalhoz.

Mi az órákon a helyét a head tagek között határozzuk meg, így fogjuk használni:

A képen szöveg látható

Automatikusan generált leírás

Ezután a Live Serverrel indítva az index.html-t, a csatolt js fájl is lefut a böngészőben.

## Hivatkozás az elemekre DOM-művelettel

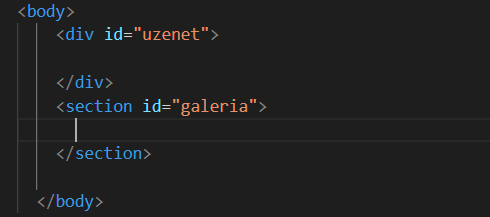
### Elem azonosítójának kiválasztása: document.getElementById

A megfelelő elemet egyedi azonosítóval látjuk el az id attribútumán keresztül, és a létrejövő objektumhierarchiában megkeressük az adott azonosítójú elemet. A keresést az egész DOM gyökérelemén, a document objektumon kell kezdeményeznünk a getElementById() paranccsal, aminek paraméterének a keresendő elem azonosítóját kell megadni, a rá való hivatkozást pedig a függvény visszatérési értéke adja meg:

Általános forma:

###### var elem = document.getElementById('id');

Példa:





A HTML elemeknek megfelelő DOM objektumok a böngésző által létrehozott JavaScript objektumok. Mint minden JavaScript objektumnak, ezeknek is vannak adattagjai és metódusai, amelyeket a DOM szabvány határoz meg.

### Egy elem belső tartalmának megváltoztatása: - innerHTML

Ha statikus HTML-t kellene készítenünk, akkor hogyan oldanánk meg a feladatot?

Az eredményt mondjuk a <p> tag-ek közé szeretnénk írni. Adjunk a p tagnek egy azonosítót:

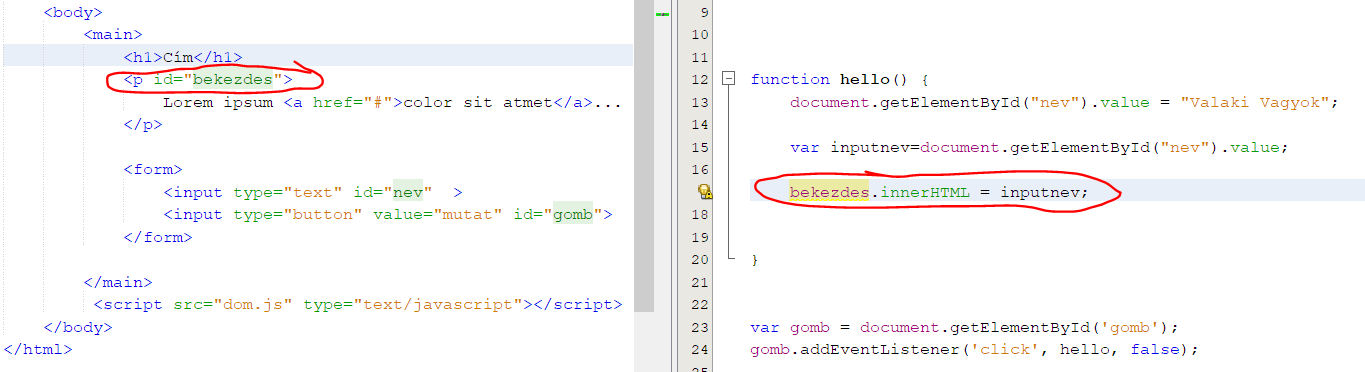
Írás:

###### elem.innerHTML = 'Tetszőleges HTML szöveg';

Olvasás

###### elem.innerHTML;

Amennyiben nem alert-tel, hanem a HTML oldalon belül, mondjuk a <p> tagbe szeretném kiíratni:



Az elemekre hivatkozhatunk még a class nevükkel. Ebben az esetben az elérés:

var tomb=document.getElementsByClassName('osztalynev')

Mivel class név többször is szerepelhet egy oldalon, ezért ennek visszatérési értéke egy tömb.

Az első 'osztalynev' elemre így hivatkozhatunk: document.getElementsByClassName('osztalynev')[0], vagy tomb[0].

Legáltalánosabban a HTML elemeket a

var tomb=document.querySelectorAll('selectorok') módon érhetjük el. Itt is tömbbe kerülnek az elemek és indexeléssel érhetjük el az egyes előfordulásaikat.

pl: var kepekTomb= document.querySelectorAll(’div img’) ősszegyűjti az összes olyan képet, amely egy div-ben van és beteszi őket a kepekTomb-be.

A 3. képet így érjük el: kepekTomb[2]

## A Dokumentum betöltődése függvény

Amikor a JS kód lefut, akkor a HTML oldalunk még nem töltődött be, tehát a böngészőnek fogalma sincs arról, hogy milyen ID-jú elemek léteznek az oldalon. Ezért hibát fogunk kapni a böngészőben a fejlesztői eszközök consol oldalán.

Ehhez az alábbi módon kell meghívnunk az utasításainkat:

A képen szöveg látható

Automatikusan generált leírás

Rövidítőfüggvény az

document.getElementById("uzenet")

elemhez:

A képen szöveg látható

Automatikusan generált leírás

Így tehát használhatjuk a documentum.getElementById helyett az ID függvényt.



## Feladat:

Adott az alábbi tömb, 3 képobjektummal. Az objektumok egy-egy kép tulajdonságait írják le: cím, elérési út, leírás:

A képen szöveg látható

Automatikusan generált leírás

Tegyük fel, hogy egy div elembe ki szeretnénk íratni a 3 kép adatai, illetve a hozzájuk tartozó képet is meg szeretnénk jeleníteni a HTML oldalunkon.

A HTML kód így nézne ki:

A képen szöveg látható

Automatikusan generált leírás.

Ebből a section tag az adott ID-vel már a HTML-ben van fixen.

Egy kép HTML kódja:

A képen szöveg látható

Automatikusan generált leírás

Ezt kell dinamikusan előállítani JavaScript segítségével.

A megjlenít függvényünk:

A képen szöveg látható

Automatikusan generált leírás

## Események és eseménykezelők – avagy hogyan reagálunk egy eseményre?

### Események típusai

Egy weboldalon nagyon sokféle esemény következhet be. Ezek egy részét a felhasználói cselekvések váltják ki, másokat maga a böngésző indukálja. A teljesség igénye nélkül nézzük meg, milyen események lehetnek.

### Egéresemények

* click: kattintás
* dblclick: duplakattintás
* mousedown: egér gombjának lenyomása
* mouseup: egér gombjának felengedése
* mouseover: az egérkurzor belépése egy elem fölé
* mousemove: az egérkurzor mozog egy elem fölött
* mouseout: az egérkurzor elhagyja az elemet

### Billentyűzetesemények

* keydown: billentyű lenyomása
* keypress: billentyű megnyomása (pl. hosszan nyomva tartásnál többször meghívódik)
* keyup: billentyű felengedése

### Oldal események

* load (window, frame, object, image): az adott objektum betöltődött
* unload (window, frame): az adott objektum bezárul (lecserélődik)
* abort (image, object): a betöltés megszakadt
* error (object, image, frame): hiba a betöltés során
* resize (document): a dokumentum mérete megváltozik
* scroll (document): a dokumentumban görgetnek

### Űrlap és űrlapelem események

* submit (form): az űrlap elküldésre kerül
* reset (form): az űrlap alaphelyzetbe áll
* select (input, textarea): szöveg kijelölése
* change (input, select, textarea): az űrlapmező értékének megváltozása
* focus (input, select, textarea, label, button): az űrlapelem megkapta fókuszt (nem csak űrlapelemeknél használható, hanem szinte az összes elemnél, pl. hivatkozásoknál)
* blur (input, select, textarea, label, button): a fókusz elhagyása (ld. az előző megjegyzést)

### Érintés események

* touchstart
* touchend
* touchmove
* touchcancel

## Eseménykezelők regisztrálása

### Inline módszer

Ezt már láttuk a múlt alkalommal:

###### <elem ontípus="JavaScript kód">

Illetve a script kód helyett egy függvényt is megadhatunk:

###### <input type="button" onclick="hello()">

### Programkódból rendelhetünk egy eseménykezelő függvényt egy eseményhez.

Az eseménykezelő függvényt a kiválasztott elem megfelelő adattagjához kell rendelni:

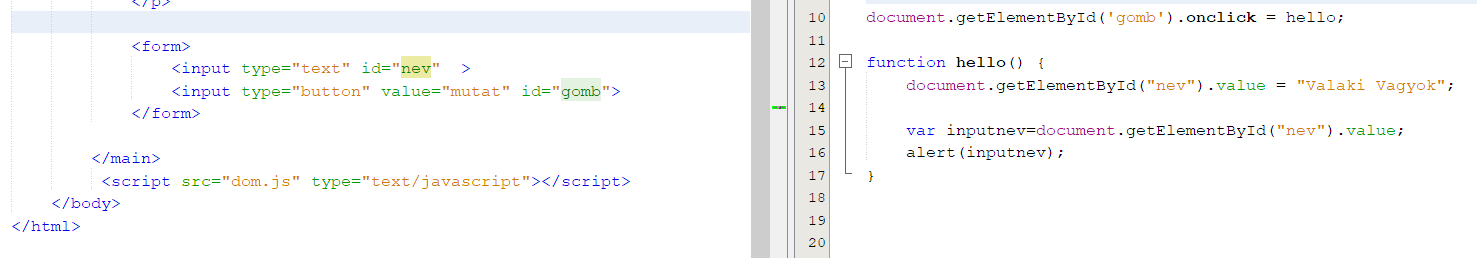
###### elem.ontípus = függvény;

Eseménykezelő függvény törlése úgy történik, hogy a megfelelő adattaghoz null értéket rendelünk:

//Eseménykezelő törlése

###### elem.ontípus = null;

Példa:



### Szabványos módszer

A szabvány két függvényt vezetett be minden DOM objektumhoz: az addEventListener függvénnyel feliratkozhatunk egy elem adott típusú eseményére, a removeEventListener függvény pedig megszünteti a hozzárendelést. Mindkét függvénynek három paramétere van:

* az esemény típusa, szövegként megadva;
* az eseménykezelő függvény referenciája; végül
* az elkapás irányát jelző logikai érték (ld. később), tipikusan hamisra állítva.

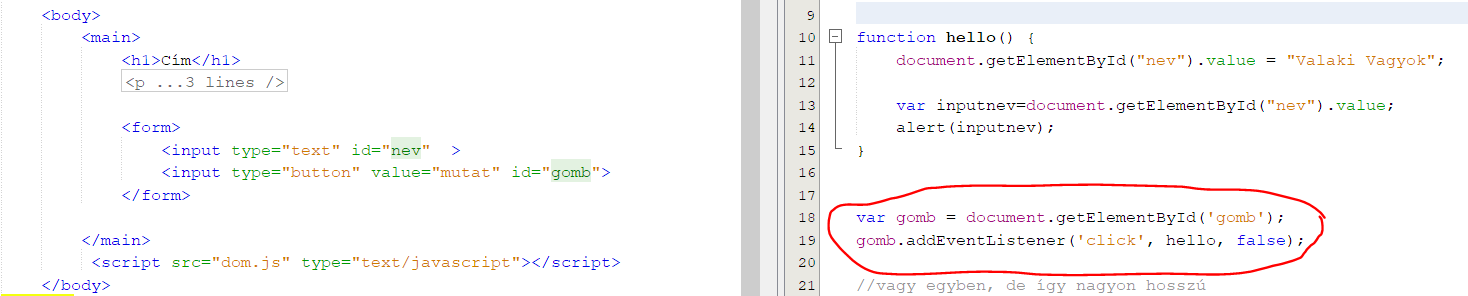
//Feliratkozás

###### elem.addEventListener('típus', függvény, false);

//Leiratkozás

###### elem.removeEventListener('típus', függvény, false);

Példa:



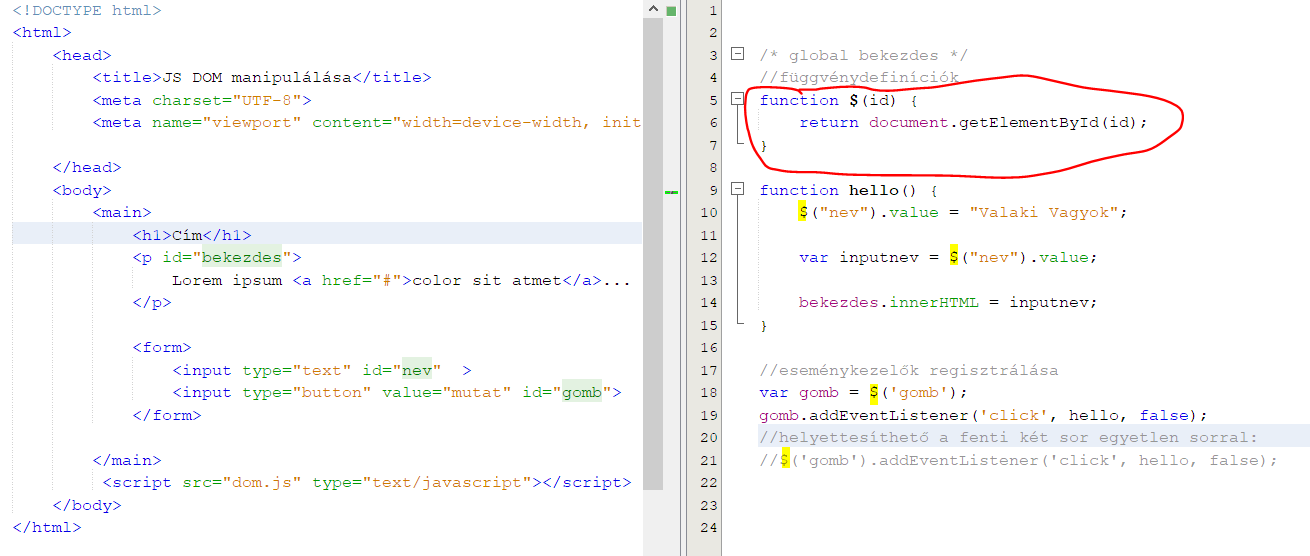
Vagy egyben, de ez így nagyon hosszú:



## A $() függvény

Kliensoldali szkriptekben gyakran kell különböző dokumentumbeli elemekre hivatkoznunk, így viszonylag gyakran fordul elő a document.getElementById() függvény használata. Azon túl, hogy a hosszúsága miatt olvashatatlanná teszi a kódot, még könnyen el is gépelhető. A gyorsabb fejlesztés, a jobban olvasható kód és a hibamentesség érdekében bevezetünk egy $() nevű függvényt, amely ugyanazt adja vissza, mint a document.getElementById() függvény, gyakorlatilag azt csomagolja be egy rövidebb nevű formába.

Példa:



## Szerkezet és viselkedés szétválasztása

* HTML: az oldal szemantikai *szerkezet*éért
* CSS: az oldal *megjelenés*éért
* JavaScript: az oldal *viselkedés*éért

### Hol csatoljuk a js fájlt? A css-t a head-ben szoktuk!

Miért van az oldal alján?

Ha a dom.js állományt a head részben töltjük be, le is fut azonnal, **mielőtt az oldal tovább töltődne.** A függvénydefiníciókkal nincs is baj, de **az eseménykezelő regisztrálásakor olyan elemre hivatkozunk (gomb), ami még be sem töltődött**, hiszen az <input type="button"> sor a forráskódban később szerepel.

***Ezért az body lezárása előtt csatoljuk a script fájlt.***

Elegánsabb megoldás azonban a script elemet a **fejlécben tartani**, akkor az eseménykezelők regisztrálását csak akkor szabad elvégezni, ha már az oldal teljesen betöltődött.

Az oldal betöltődését egy esemény jelzi, a window objektum load eseménye, amire felírathatunk egy olyan függvényt, mely az oldal betöltődése után a kezdeti beállításokat elvégzi. Nevezzük ezt init függvénynek. Ezzel a JavaScript kód a következőképpen alakul:



Forrás: <http://webprogramozas.inf.elte.hu/tananyag/wf2/lecke5_lap1.html>