Mobil- és webes szoftverek

- JavaScript alapok
- DOM manipuláció
- Események kezelése

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Gincsai Gábor
gincsai@aut.bme.hu

Java és JavaScript

- Brendan Eich, a Netscape mérnöke alkotta meg.
  - Mocha, majd LiveScript néven 1995-ben.
  - 1995 decemberben a Netscape és a Sun licenc megállapodás, azóta JavaScriptnek hívják.
  - A „JavaScript” ma az Oracle bejegyzett védjegye.
- Az eredeti cél egy egyszerű, nem csak hivatásos fejlesztők által használható szkript nyelv megalkotása volt, amely a Java nyelvet kiegészítve lehetővé teszi interaktív weboldalak készítését.

JavaScript változatai

- JScript (1996 augusztus)
  - Microsoft által a jogi problémák elkerülésére más néven kiadott dialektus.
  - Az Internet Explorer 9-ben található JScript 9.0 a JavaScript 1.8.1 és az ECMA-262 5. változatával kompatibilis.
- ECMAScript (1 változat 1996 június)
  - ECMA által az ECMA-262 szám alatt szabványosított változat
  - Mind a JavaScript, mind pedig a JScript a szabványhoz képest további funkciókat biztosít.
- Jelenleg ECMAScript 2017 a lefrissebb verzió.

„ECMAScript was always an unwanted trade name that sounds like a skin disease.”

JavaScript egyéb változatai

- ActionScript
  - Macromedia (azóta Adobe) által a Flash platform programozására kialakított dialektus.
- TypeScript
  - Microsoft által készített bővítmény
  - Típusossággal és valódi osztály-alapú objektum orientáltsággal egészíti ki a JavaScriptet.
  - Funkcionalitásában és szintaktikájában az ECMAScript-re épít.
  - A JS és TS fejlődése nagy hatással vagy egymásra mindkét irányban.

JavaScript, CSS és HTML

- HTML (HyperText Markup Language): tartalom
- CSS (Cascading Style Sheets): megjelenítés
- JavaScript: interaktivitás

```
<!DOCTYPE html>
<html>
   <head>
      <link type="text/css" rel="stylesheet" href="mystyle.css" />
      <script type="text/javascript" src="myscript.js"></script>
   </head>
   <body>
      ...
      <script type="text/javascript" src="myscript.js"></script>
   </body>
</html>
```

Java és a JavaScript mindenben eltér

Java	JavaScript
statikusan típusos	dinamikusan típusos
erősen típusos	gyengén típusos
bájtkódból töltődik be	forráskódból töltődik be
osztály-alapú objektumok	prototípus-alapú objektumok

- Az eredeti JavaScript specifikációban az összes Java kulcsszót lefoglalták.

HTML és a JavaScript

- A `<script>` elemet mindig explicit záró címkével használjuk.
- A `<body>`-ba is tehető JS hivatkozás
  - Teljesítmény okokból az oldal végére a `</body>` elé tesszük.
- A `<script>` tag segítségével a HTML fájlba is kerülhet JS kód.
  - kód karbantarthatósága,
  - a funkciók szétválasztása ,
  - teljesítmény okok miatt ez nem célszerű.
- Az elemekhez a HTML-ben is rendelhetünk eseménykezelőket
  - pl: onclick="..."
  - kód átláthatósága miatt ez nem célszerű.

A nyelv elemei

Kommentek	`// egy soros; /* több soros */`
Aritmetikai operátorok	`+, -, /, *, %, ++, --`
Értékadás operátorok	`=, +=, -=, *=, /=, %=`
Bitenkénti operátorok	`&, |, ^, ~, <<, >>, >>>`
Logikai operátorok	`||, &&`
Összehasonlító operátorok	`==, ===, !=, !==, <, >, <=, >=`
Feltétel vizsgálat	`if..else, switch..case (break, default), instanceof, typeof, .. ? .. : ..`
Ciklusok	`for, for..in, while, do..while, break, continue`
Hibakezelés	`try..catch..finally, throw`
Objektumok kezelése	`New, delete`
Függvények	`function, return`

Nyelv által definiált típusok

- Egyszerű adattípusok
  - `String` – objectbe burkolható
  - `Number` – objectbe burkolható
  - `Boolean` – objectbe burkolható
  - `Null`
  - `Undefined`
  - `Symbol` – (ES6 óta)
- Összetett adattípus
  - `Object`

Egyszerű értékadás

```
var szoveg = 'Gipsz Jakab’;
szoveg2 = "Gipsz Jakab";
var szam = 10;
var logikai = true;
```

- Figyeljük meg, hogy ha a var kulcsszót elhagyjuk, akkor is létrejön a változó, azonban ebben az esetben a globális névtérben a window objecten jön létre.

strict mód

- A var kulcsszó véletlen elhagyásából eredő problémákat a strict mód bekapcsolásával lehet elkerülni, mert ilyenkor a var elhagyása hibát eredményez.

```
"use strict";
var ev = 2016;
honap = "január"; // Ez hibát okoz
```

Gyenge típusosság

- A JavaScript nyelv gyengén típusos
  - `var` kulcsszóval hozzuk létre
  - Egy változó amiben sztring érték van tartalmazhat később számot.
  - Összeadás operátor működése változik a változóban tárolt érték típusától
    - Stringnél konkatenáció
    - Numbernél összeadás
    - És mi lesz ha stringet és numbert adunk össze?

Gyenge típusosság

```
var a = 3;
var b = 2;
alert(a + b); // 5 (Number)

a = 'Gipsz ';
b = 'Jakab';
alert(a + b); // Gipsz Jakab (String)
```

Implicit típus konverzió

- A JavaScript gyengén típusos
  - Ha összeadunk két változót, előfordulhat, hogy az egyikben string a másikban pedig number szerepel.
  - Implicit típus konverzió van.

```
var ev = 2016;
var honap = 'január';
alert( ev + honap ); // '2016január'

alert( ev + honap + 1 ); // '2016január1'
alert( ev + 1 + honap ); // '2017január'
```

== és === közötti különbség

- Az == csak az értéket hasonlítja össze tehát a 1 mint szám és az "1" mint sztring egyenlő lesz.
- Az === egyenlőség viszont a típust infót is ellenőrzi tehát egy szám nem lehet egyelő egy sztringgel.

```
var ev = 2016;
if( ev == '2016' ) { // Ez igaz
      alert( "2016 == '2016' igaz" )
}
if( ev === '2016' ) { // Ez nem igaz
      alert( "2016 === '2016' igaz" )
} else {
      alert( "2016 === '2016' hamis" )
}
```

Ha nem adunk meg értéket undefined

- Ha egy változók értékadás nélkül hozunk létre, akkor
  - az értéke **undefined** lesz
  - a típusa is **undefined** lesz

```
var nap;
alert(nap); // undefined
if( nap === undefined ) { // true
      alert( "nap === undefined igaz.");
}
if( typeof nap === 'undefined' ) { // true
      alert( "typeof nap === 'undefined' igaz")
}
```

Mi a null típusa?

- Az `undefined` típusa is `undefined`
- A `null` esetében más a helyzet, mert a null egy `object`.

```
var n = null;
alert( typeof n ); // 'object'
```

Konstansok

- EcmaScript 6-tól van konstans is a JavaSciprtben.
- Konstans létrehozásakor az értéket is meg kell adni.
- Később nem módosítható az értéke.

```
const PI; // SyntaxError: missing = in const declaration
const PI = 3.14;
alert( PI ); // 3.14
PI = 500; // SyntaxError: invalid assignment to const PI
```

Változók láthatósága

- A `var`-ral létrehozott változók az egész függvényen belül látható (function scoping).
- ES6-től változót `let`-tel is létre lehet hozni.
- A `let`-tel létrehozott változók csak a blokkon belül láthatók (block scoping).

```
for( var i = 0; i < 2; i++ ){
        let loc = i;
}
console.log(loc); // ReferenceError: loc is not defined
```

Logikai típusok

- JavaScriptben minden bool-lá alakítható!

truthy	falsey
true	false
'0'	0
123 vagy -123	Nan
'valami'	'', (üres string)
[]	null
{}	undefined

Minden bool-lá alakítható

- A dupla tagadással mindenből készíthetünk bool-t.

```
var b = !!'valami';
alert(typeof b);    // boolean
alert(b);           // true

alert(!!'0');       // true
alert(!!0);         // false
```

== operátor még egyszer

- Mi lesz ha ==-vel összehasonlítjuk a sztring 0-át a szám 0-val?
  - A '0' sztringként thruthy.
  - A 0 szám pedig falsy.
- Mi lesz az eredmény?

```
if( '0' == 0 ) {    // true
      alert("Igaz")
}
```

- Az == csak az értéket nézni és az mindkét esetben 0-a, nem alakítja bool-lá előtte.

Primitív típus burkolása objektumba

- Az egyszerű típusokat objektumba lehet burkolni
  - Numbert a `new Number( 12 )`-vel.
  - A bool értéket pedig `new Boolean( true )`-val.
- Ilyen esetben ha bool-lá alakítjuk dupla tagadással igaz vagy hamis értéket kapunk? Miért?

```
var n = new Number(0);
alert(!!n);  // true

var b = new Boolean(false);
alet(!!b);   // true ?!
```

Változók létezésének vizsgálata

- Mivel minden változó az értékétől függetlenül bool-lá alakítható és az undefined érték falsy, ezért azt, hogy egy változónak van-e értéke az alábbi kóddal tudjuk vizsgálni.
- Mi történik akkor ha a változó értéke false?

```
var valtozo;
if( valtozo ) {      // bool-lá alakítja
      alert("A valtozó létezik");
} else {
      alert("A változó nem létezik")
}
```

Alapértelmezett érték megadása

- Mivel az undefined falsy, ezért ha egy változónak csak akkor szeretnénk értéket adni, ha még nincs neki, azt az alábbi kódrészlettel megtehetjük.

```
var afakulcs;

var szokasosafakulcs = 27;

var adokulcs = afakulcs || szokasosafakulcs;

alert(adokulcs);    // 27
```

- A megoldás lényege, hogy az `afakulcs` false lesz ezért a vagy után megadott rész is kiértékelődik és azt az értéket kapja meg az `adokulcs` változó.

Feltételes kódfuttatás

- Az alábbi kódrészlettel megoldható, hogy az `alert` csak akkor hívódjon meg, ha az `x` változó értéke truthy.

```
var x = ''; // próbáljuk ki 'a' -val is.
x && alert('fut');
```

Tömbök kezelése JavaScriptben

- Tömböket az alábbi két módon lehet létrehozni.
  - A két megoldás ekvivalens egymással.

```
var napok = [ 'hétfő', 'kedd', 'szerda' ];
var evszakok = new Array('tavasz','nyár','ősz','tél');

alert( typeof napok ); // 'object'
alert( typeof evszakok ); // 'object'

for( var i = 0; i < napok.length; i++ ) {
      alert( napok[ i ] ); // hétfő, kedd, szerda
}
```

Elem hozzáadása és elvétele

- A tömbhöz új elemet a push()-al tudnk adni
- Tömbből az utolsó elemet a pop()-al tudjuk kivenni.

```
napok.push( 'csütörtök' );
napok.push( 'péntek' );
var utolso = napok.pop();
alert(utolso); // péntek

for( var i in napok ) {
      alert( napok[ i ] );
      // hétfő,kedd,szerda,csütörtök, de péntek nem!
}
```

splice(index, howMany [, el1, ... elN])

- Index: a tömb mósdosítása melyik elemnél kezdődik
  - ha negatív a tömb végéről számol
- howMany: az eredeti tömbből hány elemet szeretnénk törölni
- el1...elN: opcionális, a tömbbe beszúrandó új elemek.

```
var szinek = [ 'piros', 'sárga', 'kék' ];
var toroltek = szinek.splice( 1, 2, 'fehér', 'zöld' );
alert("Törölt színek:");
for( var t in toroltek ) {
   alert( toroltek[ t ] ); // A törölt elemek: sárga,kék
}
alert("Ami maradt:");
for( var sz in szinek ) {
   alert( szinek[ sz ] ); // A megváltozott:piros,fehér,zöld
}
```

new Array(3) vs [3] különbség 1.

- Az előző példában láthattuk, hogy a [] vagy new Array()-el létrehozott tömbök ugyanúgy viselkednek.
- Egy apró eltérés van
  - A new Array(3) esetében egy 3 elemből álló tömböt hozunk létre úgy, hogy minden eleme undefined lesz.
  - A [3] pedig egy egy elemű tömböt hoz létre aminek a 0. eleme a 3 lesz.

```
var a = [3];
var b = new Array(3);

alert( a.length );          // 1
alert(b.length );           // 3
```

new Array() vs [] különbség 2

- Ezen felül még egy a gyakorlatban nem előforduló eset lehet.
- A `new Array()` meghívja az `Array` kontruktorát
- A `[]` pedig csak létrehoz egy üres tömböt.
- A probléma akkor lép fel, ha létrehozunk egy `Array` nevű függvényt ☺

new Array() vs [] különbség 2 példa

```
function Array() {
      this.is = 'SPARTA’;
}

var a = new Array();
var b = [];

alert(a.is); // => 'SPARTA'
alert(b.is); // => undefined
a.push('A’); // => TypeError: a.push is not a function
b.push('A’); // => 1 (OK)
```

Függvények

- A paramétereknek típusa
- A visszatérési érték típusát sem lehet megadni
- Ha a paraméterek száma a híváskor és deklarációkor eltért nem okoz gondot.
  - Minden paraméter opcionális, ha valamit nem adunk meg undefined lesz az értéke.
  - Ha több paraméterrel hívjuk meg figyelmen kívül hagyja.
  - Függvény híváskor a paraméterekre névvel nem lehet hivatkozni, ezért csak a paraméter lista végéről tudunk elhagyni elemeket.
- Nincs overload
  - Ha van két azonos nevű függvény, a később deklarált nyer.

Függvények

```
function kiir(szoveg) {
      alert(szoveg);
}
kiir('Ébresztő!');

function osszead( a, b ) {
      return a + b;
}
var osszeg = osszead( 2, 3 );
kiir(osszeg); // => 5
```

Paraméter alapértelmezett értékkel

- ES6-ban már lehetőség van a függvény létrehozásakor megadni a paramétereknek alapértelmezett értéket. Így ha az alábbi példában elhagyjuk a b paramétert, akkor az nem undefined lesz, hanem 4.

```
function osszeadAlap(a, b = 4) {
      return a + b;
}
alert(osszeadAlap( 2 ) )    // => 6
```

A függvény is egy teljes értékű típus

- Meglepő, de JavaScripben a function egy teljes értékű típus.

```
function kiir(szoveg) {
     alert(szoveg);
}
alert( typeof( kiir ) ); // => 'function'
```

Függvényt paraméterül is átadhatunk

- Mivel teljes értékű típus a function ezért egy függvénynek lehet függvény a bemenő paramétere
  - A paraméterként megkapott függvényt meg is tudjuk hívni.
  - Figyelni kell, hogy tényleg függvény-e a paraméter.

```
function mind( tomb, fv ) {
        for( var i in tomb ) {
                fv( tomb[ i ] );
        }
}
mind( napok, kiir );
```

Weboldal tartalmának manipulálása JavaScriptből

DOM manipuláció

JavaScript API-k

- Browser API-k:
  - DOM API: HTML és CSS manipulálására.
    - HTML tagek dinamikus létrehozása, törlése, módosítása
    - HTML elemek stílusának módosítása futási időben
  - Szerver kommunikációt lehetővé tevő API
    - XmlHttpRequest amin keresztül kéréseket indíthatunk a szerver felé. Tipikusan REST API-n keresztül.
  - Client Storage API
    - Kliens oldali adattárolást tesz lehetővé. Pl.: web storage, IndexedDB
- Külső API-k:
  - Google Maps API
  - Twitter API

Legfontosabb böngészőbeli objektumok

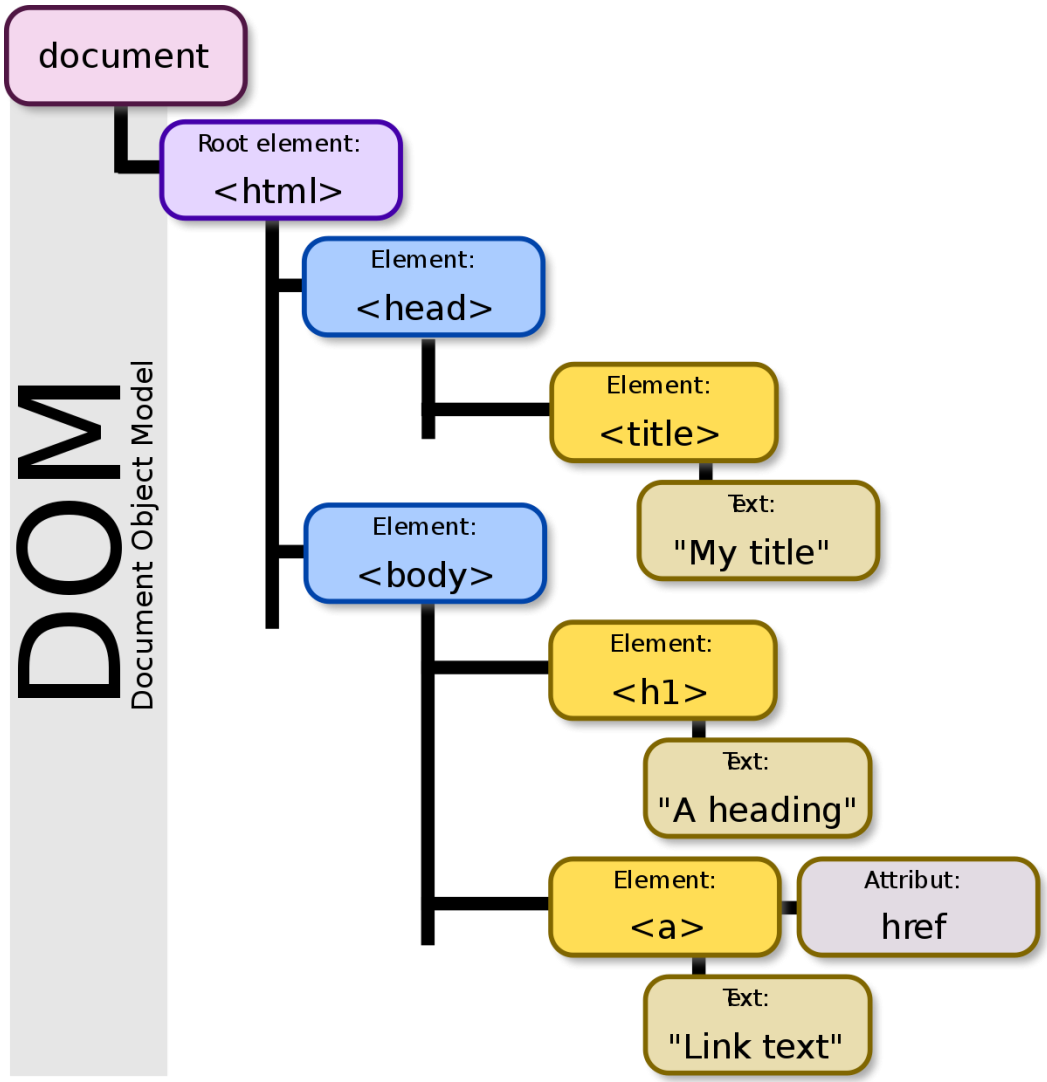
A legfontosabb objektumok

- Window objektum
  - A böngésző tabfüle, amibe a weboldal betöltődik
    - Itt érhetjük el a console-t.
    - Itt jönnek létre a globális objektumok.
    - Az oldal URL-jét (localtion) is itt találjuk és módosíthatjuk.
    - Vagy a history-t, ami az előre / hátra lapozáshoz kell.
- Navigator objektum
  - A böngésző állapotát tárolja
    - Lekérdezhető a felhasználó által preferált nyelv, geolokáció…
- Document objektum
  - Maga a DOM, ami a Windows objektumba betöltődik.
    - Ezen keresztül tudjuk a HTML tageket módosítani.

DOM – Document Object Model

- A DOM egy olyan modell, mely leírja egy HTML oldal felépítését egy fa struktúrában, melynek gyökér eleme a „Document” objektum, ami alatt az oldalon lévő elemek találhatók hierarchikusan.
- A DOM-ból is több verzió létezik, amit szinteknek neveznek. Az egyes verziókón látható a DOM fejlődése is

A DOM fa



Forrás: https://en.wikipedia.org/wiki/Document_Object_Model

Elemek lekérdezése a DOM-ból

- Elem lekérdezése tag alapján
  - `document.getElementsByTagName("img")`
- Adott CSS osztállyal rendelező elem:
  - `document.getElementByClassName()`
- Adott id-jú elem:
  - `document.getElementById(...)`
- Tetszőleges elem lekérdezésére selectorral
  - **`document.querySelector('a’);`**
  - **`document.querySelectorAll('a');`**

Elemek létrehozása dinamikusan

- A `createElement()` segítségével új HTML elemeket hozhatunk létre.

```
var para = document.createElement('p');
```

- Állítsuk be a szükséges attribútumokat, vagy a tartalmát

```
para.textContent = 'Új paragrafus';
```

- Keressünk ki egy a HTML-ben már létező elemet és ahhoz fűzzük hozzá az újat, hogy megjelenjen.

```
var sect = document.querySelector('section');
sect.appendChild(para);
```

Egy példa elem létrehozására

```
// Első section lekérdezése
var sect = document.querySelector('section');

// <p> létrehozása
var para = document.createElement('p');
para.textContent = 'Új paragrafus';

// <p> hozzáfűzése a section végéhez
sect.appendChild(para);

// Sima szöveg létrehozása tag nélkül
var text = document.createTextNode(' — további szöveg ');
// Az első <p> tag megkeresése
var linkPara = document.querySelector('p');
// <p>-hez hozzáfűzzük a szöveget.
linkPara.appendChild(text);
```

Oldal elemeinek módosítása

- A lekérdezett adatokat módosítani is lehet.

```
var para = document.querySelector('p');
para.style.color = 'white';
para.style.backgroundColor = 'black';
para.style.padding = '10px';
para.style.width = '250px';
para.style.textAlign = 'center’;
```

```
<p style="color: white; background-color: black;
padding: 10px; width: 250px; text-align: center;">
Új paragrafus</p>
```

Tulajdonságok CSS-ben és JS-ben

- Ügyeljünk arra, hogy a tulajdonságokat másképpen írjuk CSS-ben és JS-ben.
  - míg CSS-ben csupa kisbetűvel és kötőjellen
  - addig JS-ben camelCase-el írjuk
- `CSS: background-color`
- `JS: backgroundColor`

Oldal elemeinek módosítása II

- Arra is van lehetőség, hogy a HTML tagekre egy-egy új attribútumot aggassunk futási időben.
- Az alábbi kódrészlet a class attribútumot állítja be highlightra.

```
para.setAttribute('class', 'highlight');
```

DOM bejárása

```
function walkTheDOM(node, func) {
  // Aktuális elemen meghívjuk a megkapott függvényt.
  func(node);
  // Vesszük az első gyerek elemét
  node = node.firstChild;
  while (node) {
    // Rekurzívan tovább hívjuk. (mélységi bejárás.)
    walkTheDOM(node, func);
    // Ha már nincs gyerek, akkor nézzük a következő testvért.
    node = node.nextSibling;
  }
}
```

Események kezelése

Eseménykezelők

- Ahhoz, hogy a JavaScript segítségével dinamikus weboldalakat tudjuk létrehozni a felhasználói interakciókat kliens oldali kezelni kell.
- Ehhez számos eseményre tudunk feliratkozni
  - Elemre kattintás esemény: **click**
  - Billentyű lenyomása: **keypress**
  - Elem fókuszba kerülése: **focus**
  - **...**

Felhasználói interakció kezelése

- Ahhoz, hogy a JavaScript segítségével dinamikus weboldalakat tudjuk létrehozni a felhasználói interakciókat kliens oldali kezelni kell
- Ehhez számos eseményre tudunk feliratkozni
  - Elemre kattintás esemény: **click**
  - Billentyű lenyomása: **keypress**
  - Elem fókuszba kerülése: **focus**
  - Egeret egy elem fölé visszük: **mouseover**
  - Beviteli mező tartalma megváltozott: **change**
  - ...

Inline eseménykezelő

- A HTML-ből közvetlenül inline fel tudunk iratkozni az adott elem egy-egy eseményére.

```
<button onclick="alert('click')">Gomb</button>
```

- Ez a megoldás viszonylag egyszerű, hiszen amint létrejön a gomb azonnal fel is iratkozunk a kattintás eseményre
- Átláthatóság szempontjából viszont nem jó, mert a HTML-ből állítjuk a gomb viselkedését.

Oldal betöltődése esemény

- Eseménykezelőt csak akkor tudunk egy elemhez regisztrálni, ha az az elem már létezik.
- Szükséges egy olyan globális esemény, ami azt jelzi, hogy a HTML elemek már betöltődtek
  - onLoad esemény pont erre való
  - A window-n találjuk meg, hiszen az adott oldalhoz tartozik.

```
window.onload = function(){
    alert('Az oldal betöltődött')
}
```

Eseménykezelő regisztrálása JS-ből

- Az onLoad lefutása után szabad csak beregisztálni eseménykezelőt, mert előtt nem biztos, hogy létezik a HTML elem.
- Keressük meg az a HTML elemet.

```
var btn = document.querySelector('myBtn');
```

- Adjuk meg, hogy melyik eseménykezelőre, melyik függvényt szeretnénk regisztrálni.
  - Az esemény neve elé kell tenni az **on** prefixet

```
btn.onclick = function() { … }
```

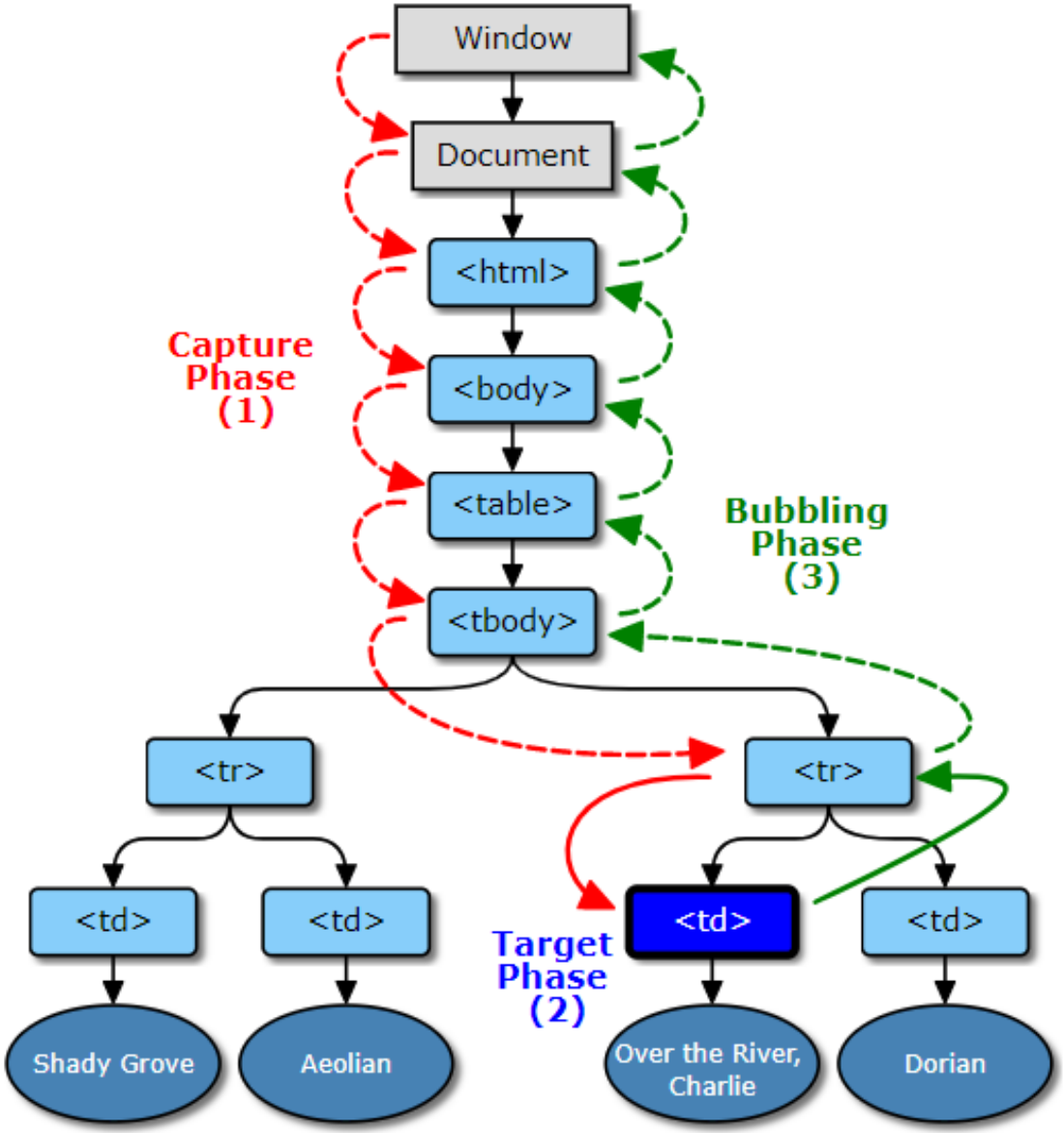
```
btn.onclick = kiir;
```

Több eseménykezelő regisztrálása

- Az előző példával egy elem egy eseményéhez csak egy eseménykezelőt tudunk megadni.
  - Csak 1db onclick lehet, ha újat adunk meg felülírjuk a korábbi.
- Ha több eseménykezelőt szeretnénk regisztrálni, akkor a megoldás az `addEventListener()`.

```
var btn = document.querySelector('myBtn’);
btn.addEventListener("click", modifyText);
```

- Az eseménykezelők a regisztráció sorrendjében egymás után futnak le.



Graphical representation of an event dispatched in a DOM tree using the DOM event flow

Forrás: https://www.w3.org/TR/DOM-Level-3-Events/#event-flow

Mit jelent a bubble?

- Egy a fában több elemnél is feliratkozunk például a kattintás eseményre, akkor
  - Először fentről lefelé megkeresi a böngésző, hogy melyik elemre kattintottak (ahova beregisztráltuk az eseménykezelőt)
  - Meghívja az ott beregisztrált eseménykezelőt
  - Majd ha lefutott, akkor a gyökér elemig felgyűrűzik (bubble up) az esemény.
  - Ezt később is tudjuk kezelni.
- A `stopPropagation()`-el leállíthatjuk a felgyűrűzést.

addEventListener 3. paramétere

- Ha az addEventListener 3. paramétere false, akkor az esemény kezelés úgy történik, ahogy az előbb láttuk. Ez a gyakori eset.

```
btn.addEventListener("click", modifyText, false);
```

- Ha viszont true-t adunk át, akkor pont fordul a kocka, mert a capture fázisban megtalált sorrendben fognak lefutni az eseménykezelők.

```
btn.addEventListener("click", modifyText, true);
```

DEMO

Eseménykezelés

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék