4. Laboratóriumi gyakorlat

JavaScript a böngészőben gyakorlat

A gyakorlat célja: A Böngészőben futó JavaScript használata. A szükséges állományok letölthetőek a Moodle honlapról `04_js_browser_lab.zip` név alatt.

Az eseménykezelők beállítását két módon lehet megoldani:

1. Az előadáson bemutatott `addEventListener()` függvénnyel.

2. A második megoldás a gyakorlat állományai mellett található `utils_old.js` fájlban található `addHandler()` függvény használata, ami általános régi Microsoft böngészőkre is működik. A fájlban a magyarázatok eligazítanak a függvények használatában, de nem fogjuk ezeket használni.

3. Az `utils.js` fájl elején található `$` függvény viszont hasznos minden esetben.

Az eseménykezelő beállítására vonatkozó példák megtalálhatóak az előadás példái (Moodle), illetve az ELTE tananyag példái között.

A laborhoz való felkészüléshez szükséges az előadáson megadott összes segédanyag, valamint az ELTE tananyag A böngészőablak szolgáltatásai című fejezetének néhány objektuma.

Tartalom

1. A location és history tulajdonságok

Olvassuk el az ELTE tananyag location objektumra vonatkozó alfejezetét.

4. Változtassuk meg a `window.location` objektum `href` tulajdonságát úgy, hogy az új elérési címet megkérdezzük a felhasználótól, miután egy adott HTML elemre kattint. Az elem legyen pl. egy paragrafus, amelyben ez a szöveg áll: „Ugrás”. A lekérdezéshez használjuk a `window` objektum `prompt()` függvényét.

```
var addr = prompt ("Kérek egy webcímet:");
```

Mielőtt beírnánk a webcímet a `href` tulajdonságba, ellenőrizzük reguláris kifejezéssel, hogy `http://nev.top` alakú, ahol `nev.` ismétlődhet, `top` pedig felső szintű tartomány név (ro, hu, info,

stb.).

5. Írjunk az ugrás paragrafus elé és után még két paragrafust: „Vissza” és „Előre”, és ugrassuk el az oldalt a `history` objektum `back()` és `forward()` függvényeivel ha ezekre kattintunk.

2. Időzítés

Olvassuk át az ELTE tananyag window objektum időzítőire vonatkozó fejezetét.

1. Nézzük át a `seconds.html` példakódot.
A kód a `window` objektum `setInterval (code, millisec )` és `clearInterval(id )` függvényeit használja, amellyel időintervallumonként lehet egy függvényt meghívni. Módosítsuk a kódot úgy, hogy a szekundumok helyett a teljes dátumot írja ki fix hosszúságú mezőkbe (a mezők minden pislogásnál legyenek ugyanakkorák, hogy az oldal megjelenítéskor ne „ugráljon” ide-oda!

2. Az előadás `DOM/setinterval_cj.html/.js` példában a pislogás clojure megoldással van megoldva.

3. A `window` objektum `setTimeout()` függvényét használva, oldjuk meg az alábbit:
Írjunk ki egy zöld színű paragrafust. Amikor a paragrafusra kattintunk váljon azonnal pirosra 4 másodpercig, utána legyen ismét fekete.
A paragrafus színváltásának módját megtaláljuk a `style.html/.js` példában, ha ezt módosítjuk megoldható a feladat.

3. CSS tulajdonságok elérése

Olvassuk át a stílusok kezelésére vonatkozó fejezetet az ELTE tananyagból. Minden HTML elemnek van egy `style` nevű tulajdonsága, ez egy `CSSStyleDeclaration` típusú objektumot tartalmaz. Ennek tulajdonságaiként jelennek meg a CSS tulajdonságok. (a stílus beállítási lehetőségeket lásd itt). Ennek tulajdonságaiként jelennek meg a CSS tulajdonságok, a neveket nem ugyanazzal a jelöléssel kell írni mint a CSS-ben. Pl., ha `el` egy paragrafusra való referencia, akkor az alábbi módon állíthatjuk be a stílusokat:

```
el.style.fontWeight = "bold";//CSS:font-weight helyett fontWeight
el.style.color = "red";
el.style.display = "block";
```

Tehát a CSS nevek „Teve” (*Camel*) jelöléssel érhetőek el. Ezen az oldalon megtalálhatóak a jelölések. A tulajdonságok értékei általában sztringek:

`"red"`, `"100px"`, `"block"` .

1. A `style.html/.js` oldalt és a konzolt használva keressük meg a paragrafust, és ellenőrizzük le, hogyan jelennek meg a CSS tulajdonságok. Módosítsunk CSS tulajdonságokat.

Fontos: betöltés után JavaScriptből nem olvashatóak le direkt az elemről a html fejlécben (head elem) beállított stílusok, csak azok, amelyeket az elemeken levő `style="..."` tulajdonságban állítottunk be. Az aktuális kiszámított stílusokat a következőképpen kapjuk meg:

```
var el = document.getElementById("para");
var cStyle = window.getComputedStyle(el);
console.log("color:", cStyle.color);
console.log("fontSize:", cStyle.fontSize);
```

Ha a `style.html/.js`-ben a konzolban meghívjuk a `getStyles($("para"))` függvényt, akkor kiírhatjuk vele a fenti tulajdonságokat. A tulajdonságok számmal és névvel is indexelve vannak, ha lefuttatjuk a függvény /**/ közötti részét, akkor látni fogjuk ezt.

Miután JavaScrit-ből ráírtunk egy tulajdonságra, az elem style tulajdonságát használva, vissza is olvashatjuk azt.

2. Írjunk be még egy paragrafust, és az egyik paragrafusra kattintva tüntessük el a másik paragrafust az elem `display` tulajdonságának `'none'` értékre való állításával. A következő kattintásra a paragrafus jelenjen meg ismét.

3. A `runwin.html/.js` példában a zöld dobozra kattintva, a doboz elindul a piros doboz bal oldala felé, és megáll amikor eléri azt. Módosítsuk a kódot úgy, hogy megállás után a második kattintásra a piros doboz lépegessen felfelé a zöld doboz bal oldalán, a harmadikra végig balról jobbra a piros doboz tetején, a negyedikre pedig térjen vissza a helyére a piros doboz jobb oldalán. Innen kezdve ismétlődjön a folyamat.

4. Összeadó példa

Módosítsuk az előadáson bemutatott `plus.html` példát úgy, hogy:

1. Oldjuk meg, hogy ha az egeret az input cellák fölé visszük (`mouseover` esemény) váljon a hátterük rózsaszínné (`elem.style.backgroundColor="pink"` , ha elem a referencia az input elemre), és ha elvisszük az elem fölül az egeret (`mouseout` esemény), akkor legyen ismét fehér (elég, ha ilyenkor üres sztringet írunk a tulajdonságra, örökölni fogja). Ha az input elemen fogjuk ki az eseményt, az eseménykezelőben a `this` érték az elemre mutató referenciát tartalmaz. Mivel az eseménykezelő `addEventListener()`-el lesz hozzáadva, a kezelő első paramétere az esemény objektum, pl. `e` , ennek `e.target` tulajdonsága az elemre mutat.

2. Oldjuk meg, hogy az `input` elemek kiválasztásakor (`focus` esemény) váljon a hátterük (`style.backgroundColor`) halványszürkére (JavaScript-ben: `"lightgray"`), és a kurzor elvitelekor más elemre (`blur` esemény) ellenőrizzük, hogy valóban számot írtunk-e be. Ha nem szám, akkor írjuk át pirosra az elemben levő sztring értéket (`"red"` szín az input elemen). Ha a felhasználó kijavítja, és ismét szám van ott, legyen az elem tartalmának színe ismét legyen fekete.

3. Vegyük ki a + gombot, és akkor végezzük el az összeadást, amikor valamelyik cellában befejeztük a szám beírását és `ENTER` billentyűt nyomunk. A `key.html` példát tanulmányozva kideríthetjük, hogy milyen billentyű esemény és milyen kódot kell figyelni a megvalósításhoz

5. Paragrafusok mozgatása

A `view_para.html` oldalban a bal oldali `div` elemben csak egy paragrafus jelenik meg láthatóan, a többi `display` tulajdonsága `none` . Oldjuk meg a következőket:

Az *Előző* és *Következő* gombokat nyomogatva, a forgassuk a paragrafusokat előre, illetve visszafelé

a bal oldali ablakban. Pl. A *Következő* gomb első lenyomásakor jelenjék meg a második paragrafus és tűnjön el az első, majd az *Előző* lenyomásakor ismét az első. A paragrafusokat körkörösen lehessen forgatni mindkét irányba, pl.: 1,2,3,4,5,1,2,...

Ha az előző pont működik, oldjuk meg, hogy a látható paragrafus előtt mindig jelenjen meg a paragrafus szekvenciában elfoglalt sorszáma: 1. , 2. , ... stb.

Ötlet: A gombok eseménykezelőjében keressük meg a bal oldali `div` elemet, majd annak a referenciáján hívjuk meg a `getElementsByTagName()` függvényt:

```
var bal = $("bal");
var p = bal.getElementsByTagName("P");
```

Ezek után egy `p` HTML elem tömböt kapunk, a tömb elemek értéke a paragrafusok referenciái. A tömb hossza a paragrafusok száma. Így meg tudjuk oldani a feladatot, ha az egyes paragrafusok `style.display` tulajdonságát átállítjuk, pl.:

```
p[i].style.display = "none"; // vagy "block" ha meg akarjuk jeleníteni
```

Az éppen kiválasztott paragrafus sorszámát egy globális változóban tárolhatjuk, vagy átírjuk az eseménykezelőt *clojure* megoldásra.

6. Számláló 1

Oldjuk meg a TAMOP tananyag 5. lecke végén található feladatot (a `szamlalo.html` fájlból kiindulva):

Készítsünk egy számláló komponenst:

- A számláló egy csak olvasható szöveges beviteli mezőből és két gombból (plusz, mínusz) áll.
- Az oldal betöltésekor kerüljön egy 0 a szám mezőbe.
- A gombok megnyomásával a szöveges beviteli mezőben lévő szám eggyel nő vagy csökken.
- Definiálj a szkriptben egy minimum és egy maximum értéket (pl. 10 és -10)! Ha a számláló eléri valamelyik értéket, akkor a megfelelő gomb ne legyen elérhető!
- Ha a gombokat hosszan nyomjuk le (pl. több mint 1,5 szekundum), akkor a számláló a gomb elengedése után automatikusan kezdje el az értéket változtatni felfelé vagy lefelé számolva a gombtól függően. Addig számoljon, amíg eléri a határértéket, vagy megnyomjuk az egyik gombot.

Ötletek: Egy gomb nem lesz elérhető, ha a `disabled` tulajdonságát `true` -ra állítjuk.

Az utolsó pont megoldásához használjuk a `mousedown` és `mouseup` eseményeket, valamint szükséges lesz a `Date` objektum használata:

```
var t = Date.now();
```

megadja a pillanatnyi időbélyeget milliszekundumban (1970. Január elseje itt is a referencia akárcsak a UNIX-ban). Pl. a `mousedown` eseménykor mentsük el az időt, és a `mouseup` eseménykor ellenőrizzük, mennyi időt volt a gomb lenyomva.

A gomboknak van egy `click()` nevű függvénye. Ha ezt meghívjuk a gombon, akkor kiváltódik rajtuk a `click` esemény.

7. Számláló 2

A feladat előtt nézzük át ismét a `window` objektum `setInterval()` és `clearInterval()`, valamint `setTimeout()` függvényeit.

a. A `button_timer.html` sablont használva, írjunk eseménykezelőket a következők megvalósítására:

- ha megnyomjuk a Start gombot, induljon el egy számláló, és növelje a `szamlalo` nevű mező értékét fél másodpercenként 1-gyel,
- ha megnyomjuk a Stop gombot a számlálás álljon le,
- a számlálást lehessen megismételni, ha a „Kezd nullától” jelölőnégyzet ki van választva akkor 0-tól, ha nem akkor folytassa ott ahol abbahagyta.

b. A HTML5 bevezette a `range` típusú input elemet (lásd a `button_timer.html` fájlban). A sablon `range` eleme az értéket 100 és 3000 közt szabályozza. Oldjuk meg, hogy menet közben változtassuk a számlálás sebességét a `range` által megjelenített csúszkát húzogatva. A `range` elem értéke a `value` tulajdonságával olvasható le, és figyelhető rajta a `change` esemény ha az érték megváltozik.

8. Delegálás paragrafusokkal

Tekintsünk egy `div` elemet több paragrafussal (`modify_para.html`). Oldjuk meg az alábbi feladatokat úgy, hogy eseménykezelőt csak a `div` elemre tegyünk (feladat delegálás). A paragrafusokra való kattintás eseményként megjelenik a `div` elemen is (buborékolás).

Az esemény objektum `target` tulajdonsága megadja az elemet amelyre kattintottunk, ennek indexét kikereshetjük összehasonlítással a paragrafusok listájában (amit megkapunk, ha a `div` alatti paragrafusokat keressük ki). A cél elem `tagName` tulajdonságával ellenőrizhetjük, hogy milyen típusú elemről jött a kattintás. Az egér esemény objektumnak van `ctrlKey, shiftKey, altKey` logikai tulajdonsága: ha ezek igazak, akkor kattintáskor le volt nyomva a Ctrl, Shift vagy Alt billentyű.

- Ctrl+kattintás: tegyük a paragrafust egy paragrafussal feljebb a szövegben. A cserét oldjuk meg a paragrafusok `innerHTML` tartalmának cseréjével. Az első paragrafus kerüljön a szöveg végére.
- Shift+kattintás: tegyük a paragrafust egy paragrafussal lejjebb a szövegben . Az utolsó paragrafus kerüljön a szöveg elejére.
- Ctrl+Shift+kattintás: Változzon meg a paragrafus háttérszíne (pl. legyen világosszürke) és legyen „szerkeszthető”: modern böngészőkben minden szöveges elemnek van egy `contentEditable` nevű logikai tulajdonsága: ha ezt `"true"`-ra állítjuk az elem szerkeszthető lesz, tehát írhatunk a paragrafusba. Ha az átállítás után azonnal meghívjuk a paragrafus `focus()` függvényét, akkor el is kezdhetjük a szerkesztést. A szerkesztést befejezhetjük: ha elvisszük a paragrafusról az egeret (`mouseout` esemény) vagy ha más paragrafusra kattintunk a `div` elemben (és `contentEditable="false"`). Ha elvittük az egeret, kerüljön vissza az eredeti szín a paragrafusra.

9. Billentyű események

Három fontos billentyű esemény van a szöveges input elemeken: `keydown`, `keypress`, `keyup` ezek ebben a sorrendben jelennek meg:

- `keydown`: a billentyű lenyomásának pillanatában jelenik meg,
- `keypress`: a lenyomás után rövid idővel,
- `keyup`: a billentyű elengedésénél jelenik meg. Ennél az eseménynél érzékelhető az `input` elem `value` értékében az új karakter.

Az esemény objektum `keyCode` tulajdonsága tartalmazza klasszikusan a lenyomott billentyű által bevitt kódot tartalmazza, `shiftKey`, `ctrlKey`, `altKey` logikai értékek a megfelelő billentyűk állapotát. Egyes böngészők megadják az esemény objektumban a bevitt karakter kódot, pl. a Firefox a `key` tulajdonságban mint UNICODE sztring.

Jelenleg minden böngésző a lenyomott billentyű kódot az esemény **`which`** tulajdonságában is eltárolja. A `keypress` esemény esetében ez a bevitt kód a karakter kódja, a `keydown` és `keyup` esetében a lenyomott billentyű kódja (a kettő nem ugyanaz, pl. `a` lenyomása esetén a billentyű kód 65, a karakter kód 97). A billentyűk eseménykezelése eléggé különbözik a böngészőkben, lásd itt a keyCode tulajdonság leírásánál.

Input elemen való billentyű lenyomás esetében az esemény implicit következménye az, hogy a karakterkód bekerül az elem `value` tulajdonságának végére.

Az előadáson bemutatott `key.html` példával tanulmányozhatjuk a billentyű eseményeket.

Feladat: Oldjuk meg a TAMOP 6. lecke végén található feladatot: Készítsünk egy csak számokat elfogadó mezőt:

- Gépelés közben meg se jelenjenek a számoktól eltérő karakterek.
- A megoldás működjön minden olyan szöveges beviteli mezőre, amelynek `szam` stílusosztály be van állítva. Ezeknek a `className` tulajdonság értéke `"szam"` lesz.

Kiinduló HTML `keypress.html`.

10. Táblázatok

Ismételjük át a HTML táblázat elem felépítő elemeit: `table`, `tbody` , `tr` , `th`, `td`. CSS-el megjelenített táblázatot tartalmaz a gyakorlat könyvtárában található `table_css.html` példa. A HTML5 jelenlegi táblázat elem a Mozillánál itt található leírva: `<table>`. Írjunk be kézzel is egy táblázatot, vagy használjuk a labor könyvtárban található példákat: módosítsuk a sorokat, oszlopokat.

Nézzük át a böngészőben felépített táblázat elem tulajdonságait és függvényeit, a Mozillánál itt: HTMLTableElement . A TAMOP tananyag is tartalmaz egy rövid leírást itt.

a. Oldjuk meg az **5. TAMOP lecke végén található feladatot**: Adott egy három oszlopból álló táblázat. A táblázat felett 3 szöveges beviteli mezővel és egy gombbal. A gombra kattintva a 3 beviteli mező értéke új sorként szúródjon be a táblázatba. Kiinduló HTML: `insert_row.html` .

Az új sort beszúrhatjuk HTML `tr` elemként megírva a táblázat `innerHTML` tulajdonságának utolsó `</tr>` sztringje mögé, mint sztring, sztring műveletekkel. Vigyázzunk, a böngésző `innerHTML` tulajdonsága nem lesz pont olyan, mint a HTML szöveg: ha kiírjuk a konzolra, látjuk a különbséget (az oldal betöltése után `var tab=$("tab"); tab.innerHTML`). A `tbody` elem, bár nem adtuk meg a HTML-ben mindig ott lesz a táblázatban.

b. Táblázat szerkesztő: készítsünk egy sorokat és oszlopokat összeadó táblázatot. Kiinduló HTML az előadás példái közt: `event/edit_table.html` , tehát ez a pont meg van oldva. Olvassuk el a feladatot, nézzük át a b. pontot, és oldjuk meg a feladat c. pontját.

Jelenítsük meg a HTML-t a böngészőben. A következőket kell megoldani:

- A táblázat fehér cellái szerkeszthetővé változtathatóak legyenek (lásd alább), és valós számokat engedélyeznek majd beírni. A többi cella ne legyen szerkeszthetővé változtatható.
- A *Számol* gomb megnyomásakor az utolsó oszlopban és az utolsó sorban (mokaszin színű cellák) a fehér hátterű cellák sorok és oszlopok szerinti összege kell megjelenjen.
- A paradicsom színű cella az összes fehér mezőkben levő szám összegét fogja tartalmazni a *Számol* gomb megnyomása után.
- Szerkesztéskor a szerkesztett cella háttere váljon halványszürkévé (`lightgray`), és ha nem valós számot írunk be, akkor a cella szerkesztésének befejezésekor tegye vissza az előző értéket.
- A táblázat alatti `msg` azonosítójú paragrafusban jelenjen meg az utolsó elvégzett művelet, illetve itt jelenjenek meg a hiba üzenetek is: ezek pirossal: "Cella szerkesztve", "Táblázat újraszámolva", "Hiba: nem valós szám, cella nem módosult", stb.
- Ha elkezdtünk szerkeszteni egy cellát, de nem módosult a tartalma, az `msg` üzenet ne változzon meg.
- A táblázat celláiban ne jelenjenek meg két tizedesnél többel kiírt számok.

A HTML5-ben (modern böngészők) minden szöveget tartalmazható elem szerkeszthetővé tehető a böngészőben. Ehhez a HTML objektum `contentEditable` tulajdonságát kell `"true"`-ra átírni (így, szövegesen, mert HTML elemre tehető tulajdonság is, és vissza: `"false"`). Ugyanakkor az elemeken van egy `isContentEditable` logikai tulajdonság, amivel ezt az állapotot le lehet kérdezni. A különböző böngészők más és más beépített műveletet biztosítanak a szerkesztéshez, de a szöveget mindegyikkel át lehet írni. A tulajdonság átállítása után ajánlott az elem `focus()` függvényét is meghívni, és akkor az elem rögtön szerkeszthető. Ezek a tulajdonságok megtalálható a táblázaton és cellán is, tehát a feladatot meg lehet oldani úgy, hogy csak egy cellát szerkesztek, és úgy is, hogy az egész táblázatot.

Ötletek: a megoldás kidolgozásakor figyelembe vehetjük az alábbiakat:

- Eseménykezelőt amennyiben lehet csak a táblázat elemre érdemes tenni, és nem külön a cellákra.
- A táblázat elemnek van egy `rows` tulajdonsága, amelyik egy csak olvasható `HTMLCollection` tömböt tartalmaz a táblázat soraira való referenciákkal:
```
var T = $("tab");
var rows = T.rows;
```
- Minden sornak van egy `cells` tulajdonsága, amelyik a cellákat ábrázoló HTML elemeket tartalmazza, pl.:

```
var cells=rows[0].cells
```

- A sorokat ábrázoló elemeknek van egy `rowIndex`, a cellákat ábrázoló elemeknek pedig egy `cellIndex` tulajdonsága: ezek a sor és oszlop indexeket tartalmazzák (az indexek 0-tól indulnak).
- A celláknak van egy `textContent` nevű tulajdonságuk, amely a cellában levő szöveget tartalmazza.
- Vigyázni kell arra, hogy ha egy cella vagy a táblázat szerkesztés alatt van, akkor ha másodszorra jön egy szerkesztést indító esemény, azt nem kell kiszolgálni (pl. ha kattintással indítjuk az eseményt, és szerkesztés közben ismét kattintunk a cella szövegén belül, átvisszük a kurzort egy másik karakterre).
- A *Szerkeszt* gombbal el lehet kezdeni a szerkesztést, nem kell feltétlenül használni, el lehet indítani kattintással is a szerkesztést `click` eseményre is.
- A táblázaton és a cellákon is kezelhetőek az egér események (`click`, `mouseover`, `mouseout`, stb.), ezek a cellákról fel is buborékolnak. A billentyű események (`keydown`, `keyup`, `keypress`) és a `focus` , `blur` események viszont nem buborékolnak fel a celláról a táblázatra, csak a cellákon foghatóak ki. A cellákon történő `focus` és `blur` események kifoghatóak mint „*capturing event*”, az esemény kezelőt ekkor külön kell beállítani az `addEventListener` metódussal, harmadik paraméter `true`. A `change` esemény nem jelenik meg a táblázat cellákon.
- A cellánként való szerkesztést el lehet kezdeni `click` eseményre, a cella otthagyását jelző `blur` esemény viszont nem buborékol fel a celláról a táblázat elemre, így két megoldás lehetséges. Első: a cellán kezeljük. Az erre szolgáló eseménykezelőt a szerkesztés elkezdésekor kell a cellára tenni, és levenni onnan, ha befejeződik a szerkesztés (a `blur-t` kiszolgáló eseménykezelőben). A másik megoldás a `blur` táblázaton való kifogása mint „*capturing event*”.

c. Módosítsuk a táblázat szerkesztőt:

- Oldjuk meg, hogy ne lehessen újsor karaktereket bevinni a táblázat celláiba. A megoldott feladatban, be lehet gépelni egy számot két sorban is egy cellába, pl. 22 helyett:
  2
  2
- A kódban található egy táblázat HTML előállító függvény, `createTable()`. Ellenőrizzük le, hogy jól működik, ha hiba van benne javítsuk ki.
- Oldjuk meg a következőt: tegyünk az oldalra még egy gombot, és egy szöveges mezőt. A mezőbe írjuk be egy új táblázat sorainak és oszlopainak számát, pl:
  `Új táblázat`*`(gomb)`*`:` **`8,12`** *`(szöveges mező)`*
  A gombot megnyomva, cseréljük ki a táblázatot egy megadott sor és oszlopszámú táblázatra (8 sor, 12 oszlop), a számolási működés lépjen ezen életbe.
- A `blur` esemény kiszolgálása a megoldásban a cellákon történik, az eseménykezelő minden `focus` esemény lefutása során kerül a cellákra. Töröljük le a `click()` eseménykezelő eseményt beállító sorát (`elem.addEventListener("blur", blur, false);`), és oldjuk meg, hogy a `blur` esemény *capuring event* fázisban legyen kiszolgálva a táblázat elemen (a `blur` esemény nem buborékol).
- Oldjuk meg, hogy ha újsor karaktert ütünk az adatbevitel végén a cellába, akkor fussanak le a cella lezárásának és számításának funkciói.

- Az összeg számoló `szamol()` függvényben oldjuk meg, hogy a sorokat és cellákat az `Array.prototype` `forEach()` (végigjárás) és `reduce()` (összeg számítás) függvényeivel járjuk végig. Ehhez a `rows` és `cells HTMLCollection` típusú *array like* objektumokat tömbbé kell alakítani, a szükséges függvény az `Array.from()` .