

I. HTML5 és CSS3 – általános tudnivalók



A HTML nyelv neve egy **mozaikszó**, amelyet az angol **Hyper Text Markup Language**, azaz a **hiperszöveges jelölőnyelv rövidítésből alkottak**. Egy leírónyelv, amely strukturált dokumentumok leírására szolgál, weboldalak előállításához fejlesztették ki, és a W3C (World Wide Web Consortium) ajánlásával, támogatásával az Internet legfőbb jelölő nyelvéné és szabványává vált. **Segítségével logikusan szervezett és felépített dokumentumokat lehet készíteni, alkalmas logikai kapcsolatok kiépítésére.** A kódot szöveges formában kell begépelni, amelyet akár egy editor segítségével is előállíthatunk, és .html (korábban .htm) kiterjesztéssel mentünk. Egyszerűségének okán könnyű vele dolgozni, hiszen rövid címkéket (tag) tartalmaz, amelyek utasításokat adnak a böngészőprogramnak, hogy az adott oldalt hogyan kell megjeleníteni.



A HTML fokozatosan formálódott fejlesztése alatt, így számos verzió létezik: 2.0, 3.2, 4.0, 4.01, a legfrissebb verziója pedig a HTML5. A HTML 2.0 és 3.2 szabványban a latin-1 kódot írták elő, emiatt elvileg nem lehetett a helyes magyar ékezetes betűket használni. A 4.0 szabvány szerint már az UCS (Universal Character Set, ISO10646 szabvány) készlet bármelyik karakterét használhatjuk, amely ugyanazt a több ezer betűt tartalmazza, mint az UNICODE.

Az igények növekedésével a HTML számos hiányossága a felszínre került. Egyrészt a megváltozott munkaképességű emberek támogatására és a megjelenítő eszközök sokfélesége mellett az oldalak elérhetőségének biztosítására a leírókód használata nem hatékony, másrészt néhány évet követően az egyéni elemek kódolására a HTML kezdett túlméretezetté válni (ld. mobiltelefonok, PDA-k), harmadrészt pedig pusztán a leírókódot használni a megjelenítés szabályozására nem elég hatékony. **Mindezek hatására szükségessé vált a HTML átírása, amelynek eredményeképpen lehetővé vált a tartalom és a megjelenítés különválasztása úgy, hogy a megjelenítést úgynevezett stíluslapokban (CSS) átláthatóbban és hatékonyabban lehet szabályoznunk.**



A HTML másik hiányossága az volt, hogy a szabályrendszerének következtetlensége és többértelműsége miatt csak speciális feldolgozóval (parser) lehetett elemezni. Annak érdekében, hogy a forráskód más XML feldolgozókkal együttműködjön és belőle modern nyelvekben előforduló dokumentum típusú objektum példány készíthető legyen, 2000-ben megalkották az XHTML-t (eXtensible HTML), ami lényegében a HTML újírása volt XML alapokon, tehát egy újabb generációs HTML-szabvány. Így módon lehetővé vált olyan korszerű keretrendszerek használata is, melyek nem támogatták a HTML közvetlen módosítását. A HTML-hez hasonlóan az XHTML-fájlban is megadható közvetlenül a HTML-kódban a megjelenítés beállítása (ld. `<p align="center">`, de lehetőség van stíluslapok használatára is. **Az XHTML szabályai szerint nem lehet a tulajdonságoknak olyan értéke amit nem teszünk zárójelben, akkor sem ha szám érték. Nem lehet olyan tulajdonság amelynek nincs értéke. A címkéket és a tulajdonságokat kisbetűvel kell írni. Nem lehetnek lezáratlan címkék.** Az XHTML azonban nem lett népszerű és jelentősebb böngésző gyártók nem is követték az XML szabványt teljességgel. Az Internet Explorer böngésző megjeleníti ugyan az XHTML dokumentumot, de csak akkor ha nem szabályos XML dokumentum. A W3 szövetség így 2009-ben az XHTML jelenlegi 2-es verziójának fejlesztése után a HTML legújabb 5-ös verziójának fejlesztésére helyezte a hangsúlyt.

A nyelvet ezt követően ismét átdolgozták, felújították és fejlesztették, ez **a legfiatalabb specifikáció lett a HTML5.** Az új szabvány eközben kompatibilis maradt a régebbi verzióival is, így a korábbi HTML szabványokhoz készült elemzők a szokásos elemeket megérthetik. Az újragondolt nyelv sokkal rugalmasabb lett. **Az újítások között lerövidítettek és egyszerűbbé tettek elemeket (címkéket), illetve újakat hoztak be.** Néhány fontos újítás a teljesség igénye nélkül: minden nyelven az UTF-8 alkalmazását javasolja; a típus (type) attribútum helyett CSS; vászon (canvas) címke, egy tartalom és keret nélküli téglalap, amelyre JavaScriptből rajzolhatunk, alakzatokat, szöveget, de akár képet és videót is; térinformatikai hely megadásának (geolocation) lehetősége elsősorban a mobil felhasználóknak IP cím, vagy GPS koordináta alapján, azaz a megbízható oldalakban megadhatjuk, hogy hol tartózkodunk. A leglényegesebb változás viszont a kiegészítő alkalmazások (plug-in) szükségletének eltávolítása volt a nyelvből. A weboldalakon általánossá vált a grafikus elemek, mozgó képek, animációk és a 3D-k alkalmazása, melyek sokfélesége megnehezítette a böngészők általános fejlesztését, mivel minden fejlesztő a saját fájl-formátumát támogatja, melyek nem kompatibilisek egymással. Ez tette szükségessé a plug-in programokat (pl. Adobe FlashPlayer, Microsoft Silverlight, JavaFX), amelynek a telepítése nélkül nem tudjuk használni az oldalon egy részét ami, lehet videó megtekintése, diagram megjelenítése, vagy akár egy képregény elolvasása.

A CSS (Cascading Style Sheets, magyarul: lépcsőzetes / rangsorolt stíluslapok) a számítástechnikában egy stílusleíró nyelv, amely a HTML vagy XHTML típusú strukturált dokumentumok megjelenését írja le. Ezenkívül használható bármilyen XML alapú dokumentum stílusának leírására is (pl. SVG, XUL). A CSS (és a HTML) specifikációját a World Wide Web Consortium felügyeli. **Stíluslappal határozhatjuk meg egy weboldalon az oldal vagy a szöveg színét, a betűk típusát, stílusát, az elrendezéseket, a táblázatok és minden elem megjelenését.** A CSS-t a weblapok szerkesztői és olvasói egyaránt használhatják, hogy átállítsák vele a lapok színét, betűtípusait, elrendezését, és más megjelenéshez kapcsolódó elemeit. A CSS elsődleges célja az, hogy szétválassza a dokumentumok megjelenését a tartalomtól. A CSS előtt a HTML dokumentumok csaknem minden megjelenéshez kapcsolódó része a HTML kódon belül volt; a betűtípusok, színek, háttérstílusok, elrendezések, dobozok, keretek és méretek külön meg voltak adva, gyakran ismétlődően, a HTML-kód közepén. A CSS használatával a webfejlesztők ezeket az információkat áthelyezhetik a stíluslapra, mely így egy sokkal egyszerűbb, kevésbé redundáns HTML-kódot eredményez. A HTML-dokumentumok kisebbek lesznek, és mivel a webböngészők gyakran tárolják a CSS stíluslapokat a gyorsítótárban, a hálózati forgalom is jelentősen csökkenhet.

A tervezése során a legfontosabb szempont az volt, hogy elkülönítsék a dokumentumok struktúráját (melyet HTML vagy egy hasonló leíró nyelvben lehet megadni) a dokumentum megjelenésétől (melyet CSS-sel lehet megadni). Az ilyen elkülönítésnek több haszna is van, egyrészt növeli a weblapok használhatóságát, rugalmasságát és a megjelenés kezelhetőségét, másrészt csökkenti a dokumentum tartalmi struktúrájának komplexitását. A CSS ugyancsak alkalmas arra, hogy a dokumentum stílusát a megjelenítési módszer függvényében adja meg, így elkülöníthető a dokumentum formája a képernyőn, nyomtatási lapon, hangos böngészőben (mely beszédszintetizátor segítségével olvassa fel a weblapok szövegét), vagy Braille-készüléken megjelenítve. Egyszerre több stíluslapot is importálhatunk, valamint megadhatunk alternatív stíluslapokat, így a felhasználó választhat közülük. A megjelenítés helyétől függően különböző stílusokat alkalmazhatunk, például a nyomtatási stílus teljesen különbözhet a képernyőn megjelenő változattól.

A stíluslap megadható egy weblapon belül, de lehet külön állományba is. Nagyon sok weblap esetén célszerű külön állományba tenni a stílusinformációkat, így minden weblap hivatkozhat arra. Ha minden weblap kinézetét szeretnénk megváltoztatni, akkor csak az egyetlen stílusinformációkat tartalmazó állományban kell megváltoztatni a szükséges tulajdonságokat. **A CSS információkat tartalmazó állományok neve tetszőleges, a kiterjesztésük a szokásos .CSS.**

Összefoglalva: **a HTML meghatározza a tartalmat, hogy mi jelenjen meg** (pl. szövegek, táblázatok, képek, felsorolások, linkek); **a CSS pedig meghatározza a megjelenítést, azaz azt, hogyan jelenjen meg a tartalom** (milyen legyen a formátuma). Ezzel válik lehetővé, hogy

- átlátható kódot készítsünk, mert abban már nem szerepelnek formázást leíró jellemzők, csak az oldal szerkezetét deklaráló kódok;
- egyszerűbben adhassuk meg a különböző eszközökön használható stílusinformációkat, így például más beállításokat használjunk nyomtatásnál, képernyős megjelenítésnél;
- az esetleg több száz vagy akár több ezer weboldallal rendelkező webhely esetén is csupán egyetlen helyen szükséges egy-egy elemtípus vagy a közös tulajdonsággal rendelkező elemek formai jegyeit meghatározni;
- gyorsan és egyszerűen lehet a webhely összes oldalát módosítani.



A HTML-t kezdetben úgy tervezték, hogy az csak a weblapok szerkezetét írja le. A weboldal megjelenésének stílusának leírását azonban más nyelvnek szánták; ez lett végül aztán a CSS. A stíluslapok azonban már a HTML kezdete, 1990 óta jelen vannak valamilyen formában, azonban az első népszerű böngészők sokáig nem támogatták a stíluslapokat. A böngészők a stílus módosítására létrehozták saját leíró nyelvüket, melyet a webes dokumentumok megjelenésének módosítására lehetett használni. Eredetileg a stíluslapokat a felhasználók használták, mivel a HTML korai verziói még csak kevés prezentációs attribútumot tartalmaztak, így gyakran bízták a felhasználóra, hogy a webes dokumentumok hogyan jelenjenek meg. A webfejlesztők igényei folyamatosan növekedtek a stilisztikai lehetőségek irányába, így a HTML nyelvbe egyre több ilyen elem került. Ilyen lehetőségek mellett a stíluslapok kevésbé voltak fontosak, és egyetlen külső stílusleíró nyelv sem lett széleskörűen elfogadva a CSS megjelenése előtt.

A CSS eredetileg Håkon Wium Lie ötlete volt 1994-ben. Bert Bos időközben egy Argo nevű böngészőn dolgozott, amely saját stíluslapokat használt; végül ők ketten döntöttek a CSS kifejlesztése mellett. Ekkor már több stílusleíró nyelv is létezett, de a CSS volt az első, ami a kapcsolás ötletét használta fel, vagyis a dokumentum stílusa több különböző stíluslapból tevődhetett össze. Ezáltal lehetőség

nyílt arra, hogy a felhasználó által megadott stílus bizonyos esetekben felülírja a szerző stílusát, míg a többi esetben örökli azt. A stíluslapok ilyen kapcsolása mind a szerző, mind a felhasználó számára rugalmas vezérlést biztosított, mivel megengedte a vegyes stílusztikai beállításokat.

Hákon ajánlata a "Mosaic és a Web" konferencián került bemutatásra Chicagóban, először 1994-ben, majd 1995-ben. Ebben az időben alakult meg a Word Wide Web Consortium is, mely később fellépett a CSS érdekében, és megalapított egy bizottságot a részletes kidolgozására. Hákon és Bert volt az elsődleges technikai vezetője a projektnek, amelyhez további tagok csatlakoztak, többek között Thomas Reardon a Microsofttól. 1996 decemberében a CSS level 1 ajánlata hivatalosan is megjelent. 1997 elején a CSS egy Chris Lilley vezette csoporthoz került a W3C-nél. A csoport azokkal a problémákkal foglalkozott, melyeket a CSS Level 1 kihagyott. A CSS Level 2 mint hivatalos ajánlat 1998 májusában jelent meg. A CSS Level 3 még 2014-ben is fejlesztés alatt állt, bár bizonyos funkcióit már használták a legújabb böngészők.

Új perspektívát nyitottak a CSS3 által lehetővé tett megjelenítési megoldások, amelyek a képszerkesztők és a szkript-nyelvek egyes funkcióit átvéve egységesebb szerkezetű és gyorsabban működő weblapokat eredményeztek

Egy oldal tervezésénél és megvalósításánál tehát akkor járunk el megfelelően, ha a megjelenést elválasztjuk a tartalomtól. Így a kód átláthatóbbá válik, egy arculati módosítás miatt nem feltétlenül kell hozzányúlni a HTML-forráskódhoz, illetve a tartalom módosítása is egyszerűbb, hiszen nincsenek a prezentációra vonatkozó zavaró tagek és paraméterek a forráskódban. A tartalom és megjelenés elkülönítésének számos más, módszertani előnye is van.



A **WCAG** (Web Content Accessibility Guidelines) a W3C akadálymentességi munkacsoportok (WAI Working Groups) ajánlása az akadálymentesítéshez, amelynek 1.0 verziója (WCAG 1.0) 1999-ben lett webes szabvány. 2008. év végén jelent meg a frissített változata (Web Akadálymentesítési Útmutató 2.0: <http://w3c.hu/forditasok/WCAG20/#guidelines>).

Ebben a dokumentumban a számítógépes alkalmazások (beleértve a weblapokat is) akadálymentesítésére vonatkozó irányelvek találhatók, valamint egy olyan feltételrendszer (teljesítési feltételek), amelynek egy akadálymentes alkalmazásnak meg kell felelnie. **A WCAG 2.0 három szintet különböztet meg, A, AA (két a) valamint AAA (három a). A legalacsonyabb az egy A-s szint, ez a minimum követelményeket határozza meg egy honlappal szemben, hogy azt akadálymentesnek lehessen nevezni.** Egy akadálymentes honlap esetében meg kell felelni minimum a WCAG 2.0 "A" szintű teljesítési feltételeinek. A szintek egymásra épülnek, így a AA-s szint eléréséhez teljesíteni kell az A-s szintet, a AAA-s szinthez pedig az összes A-s és AA-s kritériumnak is meg kell felelni. A W3C részletes útmutatót dolgozott ki, hogy miként kell értelmezni az irányelveket (WCAG 2.0 Értelmezése) és miként lehet azokat betartani (Technikák a WCAG 2.0-hoz), ezeknek az anyagoknak a fordítása megtalálható W3C Magyar Iroda honlapján.

Info-kommunikációs szempontból számos felhasználó hátrányos helyzetűvé válhat, amikor például a weben böngészik. Ebbe a körbe tartoz(hat)nak a fogyatékossgal élő felhasználók is, de technológiai okok miatt (pl. régi böngészőprogram használata, okostelefonok használata arra fel nem készített oldalakon), bárki hátrányos helyzetbe kerülhet. Ha a honlapunk kialakításánál ügyelünk az akadálymentes elérhetőségre, akkor azzal nagyon sok felhasználó dolgát megkönnyítjük.



A vak felhasználók (jellemzően) ugyanolyan böngészőprogramokat használnak a weben történő böngészéshez, mint a látó felhasználók, csak emellett a számítógépre telepítve van egy úgynevezett képernyőolvasó alkalmazás is, amely felolvassa a weblap tartalmát annak forráskódja alapján. A képernyőolvasó alkalmazások használatához a vak felhasználóknak számos gyorsbillentyűt meg kell tanulniuk: ö. különböző billentyűkombinációkkal tudják kigyűjteni az oldal címsorait, amelyet tartalomjegyzékként használhatnak, és az adott címsorra ugorhatnak, illetve a hivatkozások is kigyűjthetők önálló ablakba a navigáció megkönnyítésének érdekében. Számos képernyőolvasó alkalmazás létezik, de a világon az egyik legelterjedtebb a JAWS nevű fizetős szoftver. A magyar nyelvű demó változat letölthető az Informatika a látássérültekért Alapítvány honlapjáról, amely 40 percig képes működni, majd az újbóli használatához újra kell indítani a számítógépet, amely egy weboldal teszteléséhez nagyon-nagyon kevés idő. (Ez a változat nem is erre szolgál, hanem arra, hogy a kipróbálás után eldönthesse az ember, hogy megvásárolja-e vagy sem.) Mivel a szoftver igen drága, nem valószínű, hogy egy webfejlesztő csak azért meg fogja vásárolni, hogy időnként egy-egy tesztet végrehajtsa vele, pedig ez fontos lenne annak érdekében, hogy meg lehessen győződni arról, hogy a vak és gyengénlátó felhasználók által jelenleg (még) legtöbbször használt alkalmazással használható-e az oldal vagy sem. De szerencsére van ingyenes alternatíva is, ami tudásában (egyes területeken) ugyan elmarad a nagyobb vetélytársától, viszont helyenként olyan funkciókat is tartalmaz, amit a nagyok nem, vagy csak későbbi változatokban. Ilyen pl. az NVDA alkalmazás, amelyet akár pendrive-ra is lehet telepíteni. Ez az a képernyőolvasó, amely egy web-fejlesztő számára a tesztelés során kényelmesen használható, hiszen nincsen időbeli megkötés a használatára, bármikor elindítható.



Amikor megismerkedünk a tagekkel, tapasztalhattuk, hogy vannak olyanok közöttük (pl. a bekezdés <p> tagja vagy címsorok <h> .> címkéi), amelyek előtt és mögött térköz vagy ki a böngészőprogram és mindig új sorban kezdi a tartalmát. Ezeket **blokkszintű elemek**nek neveztük: **megjelenésüket dobozszerűen kell elképzelni úgy, hogy azok tartalma külön sorokban jelenik meg és tartalmazhat más dobozszerű vagy sorszerű elemeket.** De vannak olyan címkék is, amelyek elé és mögé nem kerül térköz és folyamatosan, ugyanabban a sorban jelenik meg az elem (pl. a félkövérre formázó tag vagy a képbeszúrás címkéje).

Ezek a **soron belüli (inline) elemek**: a szövegen belül folyamatosan jelennek meg és csak sor-szintű elemeket tartalmazhatnak.

A HTML-ben saját magunk is definiálhatunk strukturális: **<div>**, ****, **<figure>**. A közöttük lévő különbség a tartalom típusában, illetve a megjelenítési sémában rejlik, és a segítségükkel megkönnyítjük a hivatkozást és a CSS-sel végzett formázást. Ezek az objektumok a böngésző számára nem hordoznak tartalomra vonatkozó információt, ezért ezek használata olyan esetekben ajánlott, amikor az előre definiált objektumok nem alkalmasak a céljainkra!

- **<div> ... </div>** (div = division, szakasz): a weboldal egy logikailag összetartozó részét alkotó tömböt, azaz egy **blokk szintű elem**et jelölnek ki. A benne elhelyezett tetszőleges tartalmakra (pl. szövegekre, listákra, képekre, táblázatokra, hivatkozásokra) egységesen lehet hivatkozni és formázást rendelni, változatos módon lehet pozicionálni, ezért gyakran a weboldalak szerkezeti kialakítását szolgáló általános eszközként szolgálnak. (Ha több is van belőlük, akkor az egyes tárolóelemeket az azonosítójuk (**id**) értékével különböztethetjük meg egymástól.)
- ** ... **: segítségével jelölhető meg olyan inline, azaz **soron belüli tartalom**, amely külön formázható (az egyes **span** elemeket – hasonlóan a **div** elemekhez – is egyedileg formázhatjuk, ha **id** tulajdonságuknak egyedi értéket adunk meg). E címke megadásával önmagában nem történik változása a HTML-oldal megjelenítésekor (ellentétben a **div** elemmel, ahol új sorban kezdődik a következő **div** elem tartalma), de így biztosíthatunk lehetőséget arra, hogy a soron belüli rész formázható legyen.
- **<figure> ... </figure>**: használatával tárolóelemben foglalhatók a képek, diagramok, ábrák, listák és a hozzájuk tartozó cím, felirat vagy magyarázat (szövege), így együtt kezelhetők, pozicionálhatók, formázhatók. Alkalmazása: a **figure** páros címkébe beágyazva helyezzük el pl. az **img**, az **ul** vagy a **table** elemet és a kapcsolódó magyarázatot is egyúttal, de a feliratot a **<figcaption>** és **</figcaption>** címek között adjuk meg.

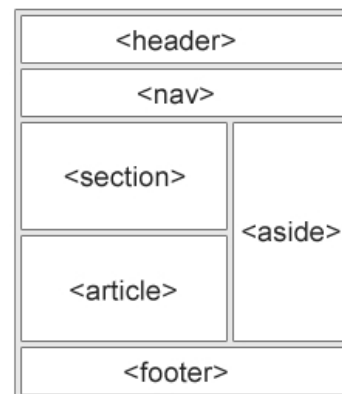
A weboldal főbb szerkezeti elemeit gyakran a **div** általános tárolóelemekben helyezik el: egy weblap törzsének tipikus szerkezete a fejlécből (*header*), a navigációs részből (*nav*), a tartalomból (*content*), az oldalsávból (*sidebar*) és a láblécből (*footer*) áll, amelyek mindegyikét általában egy közös befoglaló elem (*container*) tartalmazza. Formázásukról, azaz arról, hogy ezek az elemek egymáshoz képest hol és hogyan helyezkedjenek el, majd a CSS-ben kell gondoskodnunk.

```
<body>
  <div id="container">
    <div id="header"> ... </div>
    <div id="nav"> ... </div>
    <div id="content"> ... </div>
    <div id="sidebar"> ... </div>
    <div id="footer"> ... </div>
  </div>
</body>
```

A HTML5 újabb szemantikus, azaz jelentéssel bíró elemeket is hozott magával – ezek egyértelműen meghatározták azok belső tartalmát. (A szemantikus elemeket minden modern böngésző támogatja.) Mivel a legtöbb weboldal az előzőekben bemutatott alapvető elemekből áll, ezért az új szemantikus elemek között megjelentek a szerkezetre vonatkozó elemek is.

A leggyakrabban alkalmazott, oldalszerkezeti szemantikus elemek:

- <article>**: cikk, blog-bejegyzés, hozzáfűzött információ
- <aside>**: oldalsáv tartalma (a fő gondolatmenethez érintőlegesen csatlakozó tartalom)
- <details>**: részletek (látható vagy elrejtett)
- <footer>**: lábléc
- <header>**: fejlécs
- <main>**: fő tartalom (weblapé)
- <nav>**: navigációs linkek
- <section>**: szakasz (a weboldal egy címmel is rendelkező része)
- <summary>**: látható összegzés a **<details>** elemnél



II. A CSS szabályrendszere (nyelvtana)

A CSS stíluslapnyelv a HTML jelölőkódokon belüli elemekhez megjelenítési tulajdonságokat (pl. színt, hátteret, szegélyt, margót, pozíciót) **rendel.** Az így kialakítható stíluslapok sokféle elemre vonatkozóan tartalmazhatnak stílusjellemzőket, amelyeket **formázatlan szöveggént, rendszerint önálló fájlban helyezünk el (.css kiterjesztéssel).** Nagymértékben megkönnyítik a szövegformázást, segítségükkel ugyanis egységes formátumosztályokat és azonosítókat alakíthatunk ki úgy, hogy saját stílusokat hozunk létre vagy a már létezőket módosíthatjuk. Ráadásul a stílusokat kombinálni is lehet egymással.

A stílusdefiníció szabályai

A CSS egyszerű szintaxissal rendelkezik, csak néhány angol nyelvű kulcsszót használ a stílusok leírásához. A stíluslap maga a stílust leíró szabályok sora. A stíluslapokon stílusdefiníciókat, azaz a weblapok HTML-elemeinek vizuális megjelenéséről adunk információkat a CSS szabályrendszerének alkalmazásával – ezek a CSS-utasítások.

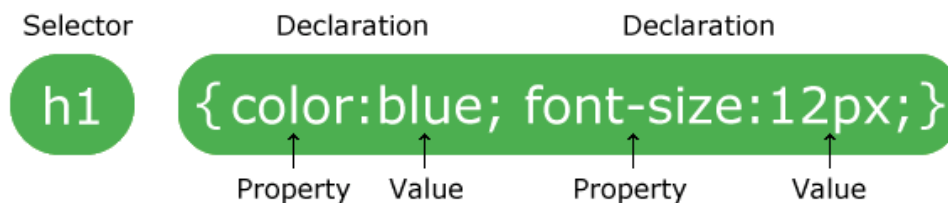
Egy CSS-utasítás két részből áll: a szelektor (*selector*) mondja meg, hogy mely HTML-elemeket kell megformázni és a deklarációs rész (*declaration*) adja meg, hogy milyen formai beállításokat kell alkalmazni.

egy stíluslap (CSS-fájl) tartalma:

```
kijelölőnév1 { leíró rész; }
kijelölőnév2 { leíró rész; }
kijelölőnév3 { leíró rész; }
```

A szabály egy HTML-szelektorra vagy egy szelektorlánccal kezdődik, ami egy egyszerű szelektor esetén az a HTML-tag, amelynek stílusát szeretnénk beállítani. Gyakran előfordul, hogy egy szelektorlánc áll a szabály elején, amelyről a kijelölők részletezésénél találhat további információkat.

A szelektort egy kapcsos zárójelpár által határolt deklarációs blokk követ, amelyben tetszőleges számú CSS-deklaráció szerepel. Egy CSS-deklaráció két részből áll: egy tulajdonságból (*property*), ami a tulajdonság nevét tartalmazza és egy értékből vagy értéklistából (*value*), ami azt a számot vagy szöveget tartalmazza, amelyet a tulajdonság értékül kap. (Bizonyos esetekben egy tulajdonságnak több értéket is megadhatunk, akkor ezeket szóközzel vagy vesszővel választjuk el egymástól.) A tulajdonságot és az értéket kettőspont választja el egymástól. Minden CSS-deklarációt egy pontosvessző zár le. (Az utolsó deklaráció végén a pontosvessző elhagyható, de mégis ajánlott, mert később esetleg hibákhoz vezethet.)



Az értékeket kizárólag akkor szabad idézőjelbe tenni, ha az érték több szóból áll és szóközt is tartalmaz: `p { font-family : "Times New Roman", "Courier New", Times, serif; }.`

A stílusok elhelyezésének módszerei

CSS-kódokat többféle módon is definiálhatunk:

- megadhatjuk a HTML-dokumentumhoz kapcsolt **önálló fájlokban** (külső stíluslapon = *external style sheet*),
- letölthetjük egy másik webhely stílusait (importálás = *@import rule*),
- meghatározhatjuk a **HTML-dokumentum fejrészeiben** (belső stíluslap = *internal style sheet*),
- elhelyezhetjük **közvetlenül a HTML-címke nyitó tagján belül** (sorközi stílus = *inline style*).



A fenti módszerek közül a legjobban a legelsőt lehet kihasználni, ezért sok weboldalból álló honlap esetén a **külső (csatolt) stíluslappal történő formázás alkalmazása javasolt.** A külső stílusfájlok használatának ugyanis több előnye van: a tartalomfájlok és a stílusfájlok sok esetben különböző személyek által párhuzamosan készíthetők, így gyorsulhat a fejlesztés; ha egy stílusfájl több weboldalhoz csatolunk, akkor abban egyetlen módosítással az összes csatolt

weboldal kinézetét megváltoztathatjuk; ugyanazokat a megjelenítést szabályozó parancsokat nem kell minden HTML-fájlban ismételten leírni, így a HTML-fájlok mérete kisebb, tartalmuk átláthatóbb lesz és gyorsabban letöltődik; (bizonyos források szerint) a keresőprogramok is relevánsabbnak tekintik a forrásfájlban hamarabb előforduló tartalmat, ezért is jobb, ha abban minél kevesebb más jellegű tartalom (formázás vagy programkód) található; egyetlen tartalmat leíró fájlhoz több megjelenítést szabályozó fájl is társítható. Ez akkor lehet előnyös, ha ugyanazt a tartalmat különböző helyzetekben (pl. másfajta megjelenítőn) másképp kell megjeleníteni.

Ezen túlmenően a böngésző beállításai között is vannak alapértelmezett stílusjegyek (kliens stílus), de ezeket a felhasználó akár meg is változtathatja (felhasználói stílus).

A) Saját külső stíluslap csatolása (external style sheet)

Külső stíluslap alkalmazása esetén egy külön fájlban (CSS-fájlban) helyezzük el és mentjük el a stílusokat, majd a `<LINK rel="stylesheet" type="text/css" href="stíluslapnév">` HTML-kód segítségével, a `head` címkén belül rendeljük (kapcsoljuk) a weblaphoz a stíluslapot.

```
<HTML>
  <HEAD>
    <TITLE>Stílusok</TITLE>
    <LINK rel="stylesheet" type="text/css" href="stilus.css">
  </HEAD>
  <BODY>
    <H1>Csatolt CSS-példa</H1>
    <P>Ez a weblap a stilus.css stíluslapot használja a megjelenítéshez.</P>
  </BODY>
</HTML>
```

A `<LINK>` tulajdonságai:

- **rel** (kötelező): az aktuális dokumentum és a csatolt dokumentum közötti kapcsolatot határozza meg (stíluslapoknál: `stylesheet`)
- **type**: a kapcsolt dokumentum típusát definiálja (stíluslapoknál: `text/css`)
- **href**: a csatolt dokumentum helyét (elérési útját) tartalmazza
- **media**: milyen média esetén alkalmazza a kapcsolt stíluslapot (lehetséges értékei: *all* = mindegyikhez, *print* = nyomtatáshoz, *screen* = képernyős megjelenítőkhöz, *speech* = képernyőolvasó szoftverekhez)

Ha egy stíluslapot több weboldalhoz is hozzákapcsolunk, akkor a stíluslapon végzett módosítással egyszerre az összes kapcsolt weboldalon változást érhetünk el.

Ha több stíluslapot csatolunk ugyanahhoz az oldalhoz, akkor az ugyanazon elemre vonatkozó stílusdefiniciók közül az lesz érvényes, amelyet a legutoljára kapcsolt stíluslapon definiáltunk, hiszen a weboldal betöltése felülről lefelé történik.

B) Másik webhely stíluslapjának csatolása

Ha egy másik webhelyen (tehát nem a sajátunkon) találtunk egy számunkra megfelelő stíluslapot, akkor azt is felhasználhatjuk, ha a `head` elemen belül a `style` páros címke között importáljuk a `@import` szabállyal: `@import URL("fájlnév elérési úttal");`. Ebben az esetben is megadható az, hogy pl. milyen eszközhöz vagy felbontás esetén alkalmazza a böngésző a stíluslapot.

```
<HEAD>
  <LINK rel="stylesheet" type="text/css" href="...css">
  <STYLE>
    @import URL(http://www.../stilus.css) print;
    @import mobil.css screen and (max-width:768px);
  </STYLE>
</HEAD>
```

A `@import` mellett alkalmazható a `@media` is, amelynek a segítségével adott eszközre, felbontásra és egyéb tulajdonságokra korlátozhatjuk az megadott stílusjellemzők alkalmazását. Ebben az esetben a `@media eszköznév and (feltételek) { stílusjellemzők megadása }` formában történik a tulajdonságok definiálása.

C) Beágyazott (belső) stílusok (internal style sheet)

Egyoldalas vagy egy különálló weboldalak esetén, illetve ha egy webhelynek csak egyetlen oldalán kell a külső stíluslap(ok)hoz képest formai módosítást végezni, akkor lehet hasznos a beágyazott stílusok használata. **A beágyazott stílusok esetén a head-ben, a <style> és </style> címké-pár között definiálhatjuk a stílusok jellemzőit.**

```
<HEAD>
  <STYLE>
    #kiemeles { font-family : "Times New Roman", Times, serif;
                font-size : xx-large;
                color : #FFFF00;
                font-style : italic;
                font-weight : 900; }
  </STYLE>
</HEAD>
<BODY>
  <P id="kiemeles"> szöveg </P>
</BODY>
```

Sajnos ezzel a módszerrel a stíluslapok azon előnye veszik el, hogy egyszerre lehet sok fájl formázását megváltoztatni, de **hasznos lehet abban az esetben, ha a külső stíluslaphoz képest csak néhány kisebb változtatást szeretnénk elvégezni.**



Ha elképzelhetőnek tartjuk, hogy az oldalt olyan böngészővel is fogják nézni, amely nem ismeri a CSS-t, akkor a megjelenítés szabályozását a **style** tagon belül a **<!--** és a **-->** HTML megjegyzések közé tehetjük azzal a céllal, hogy a böngésző ne vegye figyelembe az általa értelmezhetetlen parancsokat. A CSS-t ismerő böngészők azonban figyelembe veszik a HTML megjegyzések között levő stílusmegadásokat is, ezért **ha a CSS-ben akarunk megjegyzéseket tenni, akkor azokat a /* és a */ jelek közé kell írunk a stílusmegadásoknál.**

Ha az oldalhoz külső stíluslapot kapcsolunk és belső stílusokat is definiálunk, akkor a **head**-ben az alábbi sorrendet kell követni: elsőként a külső stílusfájlt kell csatolni, majd azt követően kell a belső stílusokat meghatározni. Erre azért van szükség, hogy a beágyazott stílusok közelebb legyenek a dokumentum törzséhez, mint a külső stílus, így a belső stílus felül fogja írni a külsőt.

D) Szövegekői stílusok (inline style)

Az utolsó lehetőség a legkisebb jelentőségű, de csak tényleg speciális esetekben ajánlott: egyetlen weblapon, egyetlen weblapelemre alkalmazott, felhasználó által definiált stílusok összességét tartalmazza, amely csak arra az egy konkrét elemre vonatkozik. Lényege, hogy a stílus hatóköre pontosan addig terjed ki, amíg a tag lezárásra nem kerül. Ebben az esetben **az adott HTML-tag nyitó címkéjében a style kulcsszót követően adjuk meg a stílusleírást: <nyitótag style="...; ...; ...;">.**

```
<P style="margin-left:10px; line-height:2; text-align:center;"> szöveg </P>
```

Ez a módszer visszatérést jelent a HTML és a CSS szétválasztása előtti állapot irányába.

A stílusok hatóköre



A böngészők rendelkeznek alapértelmezett stílusokkal, amelyek meghatározzák, hogy az egyes objektumok hogyan jelenjenek meg akkor, ha nem szabályozzuk azok megjelenítését. Például ilyen alapértelmezések a fehér színű háttér, fekete színű betűk, aláhúzott, kék színű linkek, a **<h1>** félkövér-nagyméretű szövege stb. Ugyanakkor a felhasználónak is van lehetősége a böngészőben beállítani néhány tulajdonságot (pl. a betűtípust vagy a háttérképek nyomtatását), illetve többnyire saját stíluslapot is megadhat.

A böngésző az oldal tartalmát fentről lefelé olvassa be. Ha közben külső fájlcsatolást talál, akkor annak a tartalmát azon a ponton beolvassa, majd folytatja a weboldal beolvasását fentről lefelé. Ily módon egyszer a külső stíluslapok, majd a beágyazott stílusinformációk, végül az objektumok nyitó tagjában elhelyezett stílus-meghatározások olvasódnak be. **Egymásnak ellentmondó stílusinformációk esetén az utoljára beolvasott lesz érvényes.** Így lehet a fejrészben felülírni a közös külső stílusfájlban foglaltakat, illetve az objektumok nyitó tagjában csak arra az objektumra vonatkozóan módosítani az egész weboldalra megadott megjelenítési jellemzőket.



Tudjuk, hogy az objektumok többségét egymásba ágyazhatjuk. **A beágyazott objektumok örökölhetik a beágyazó objektum formázásait, többszintű beágyazás esetén is.** Pl. ha egy bekezdésben van egy kiemelt szövegrész, akkor arra nemcsak a saját stílusmegadása lehet érvényes, hanem a bekezdésre megadott formázás is. Azonban az öröklés nem mindig történik meg. Egyrészt vannak olyan tulajdonságok, amelyek nem öröklődnek (pl. a szegély vagy a margó), másrészt egy objektum megjelenítési tulajdonságokat kaphat a beágyazó objektumon kívül az egyedi, az osztály-, illetve összetett kijelölők segítségével megadott stílus-meghatározások által is, amelyek között ellentmondás állhat fenn.

Példa: Az alábbiakban látható pelda.css stílusfájl alkalmazását követően milyen színnel jelennek meg a bekezdések?

```
BODY { background-color : blue; color : black; }
P { color : red; }
.kepek { color : blue; }
#elso { color : white; }
```

A példa szerint a bekezdés betűszíne az öröklés alapján feketének, elemkijelölés alapján pirosnak (**p**), osztályazonosító szerint kéknek (**.kepek**), azonosító-kijelölő alapján pedig fehérnek (**#elso**) kellene lennie. Hogyan fog megjelenni? A szabály az, hogy mindig az adott objektumra legjellemzőbb stílusmegadás lesz érvényes – vagyis a bekezdés szövege fehér színű lesz, mivel az örökölt, elemkijelölővel, illetve osztályazonosítóval megadott tulajdonságok sok más objektumra is érvényesek lehetnek, de az azonosító-kijelölővel kifejezetten erre a bekezdésre vonatkozó megjelenítést adtunk meg. Ha törölnénk a **#elso** azonosító-kijelölő megadást, akkor az öröklés alapján az osztálykijelölő lesz a legspecifikusabb. Ha még a **.kepek** osztálykijelölő megadást is eltávolítanánk, akkor a **P** elemkijelölő lesz érvényes, mert az adott objektumra nézve ez a legjellemzőbb.

A CSS-rangsor

A CSS-rangsor azt szabja meg, hogy melyik sablon / stílus legyen érvényes egy adott elemre, ha többféle is meg van adva. Vagyis melyik írja felül a másikat. Általánosságban elmondhatjuk, hogy a HTML-forráskódon előről a vége felé haladva megy végig a böngésző. Ennek következtében, ha van két azonos vagy átfedő stílusunk, akkor minden esetben az lesz az érvényes, amelyik közelebb helyezkedik el ahhoz a ponthoz, ahol alkalmazzuk a stílust. Tehát: ha a **head**-ben először importálunk egy külső stíluslapot és az utána lévő sortól kezdve még írunk pár plusz szabályt, melyek közt van átfedő, akkor a plusz szabály lesz az érvényes, hiszen az felülírja a korábban a külső stíluslapon megadottat. A közvetlenül megadott CSS-stílusok pedig felülírják a **head**-ben megadott akár külső, akár belső stíluslapokat. Tehát általános szabályként elmondható: **mindig a formázandó HTML-kódhoz legközelebbi stíluslap végzi a formázást.**

Az alábbi táblázat az imént elmondottakat mutatja be: olyan sorrendben jelzi a stílusokat, amilyen sorrendben a kódban előfordulnak, és amilyen közel állnak a formázandó címékhez. A lentebb, azaz a formázandó címkéhez közelebb álló stílus felülbírálja a hátrébb állót.

<u>CSS-rangsor</u>	<i>megjegyzés</i>
• a böngésző alapértelmezett stíluslapjai	minden böngészőnek megvan a maga alapértelmezett megjelenítési módja (ezek a különféle böngészőkben általában megegyeznek – pl. betűszín, háttérszín, linkek megjelenítési módja)
• felhasználói stílusok	a böngésző alapbeállításainak megváltoztatásával jönnek létre (néhány alapbeállítást a böngésző beállításai között is át lehet állítani, másokat csak kézi szerkesztéssel: pl. Eszközök / Internetbeállítások... / Általános / Színek, betűkészletek; Kisegítő lehetőségek)
• külső stílusok	a HTML-oldal fejlécében LINK taggal elhelyezett, külső stíluslapra mutató hivatkozás (ez lenne a legideálisabb)
• beágyazott stílusok	a HTML-oldal fejlécében a külső stíluslap-hivatkozás után található a <STYLE> és </STYLE> címkék által közrezártan – csak arra a dokumentumra érvényesek, amelyben elhelyeztük azokat
• szövegekzi (belső) stílusok	néhány HTML-címke nyitó tagjában a STYLE jellemzővel megadott tulajdonságok jellemzője, amelyben csak a konkrét objektumra vonatkozó stílust adhatunk meg

A megjelenítési tulajdonságok megadásakor érdemes figyelembe venni azt is, hogy **a százalékban megadott értékek a korábbi beállítások alapján értékelődnek ki.** Az is megemlítenő, hogy sok esetben a CSS hierarchiáját úgy építjük fel, hogy a külső CSS önmagában is épít az átfedésre.

A dokumentumfa

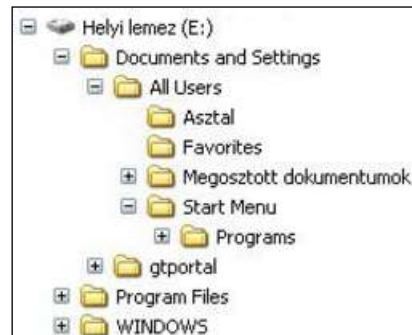


Aki webfejlesztésre vállalkozik, annak számára nem jelentenek újdonságot az egyes elemek kapcsolatát leíró faszerkezetek. Az egyik leggyakrabban használt példa erre a háttértárolókön kialakított könyvtárszerkezet. Ebben az esetben a gyöker könyvtárban található alkönyvtáraknak is lehetnek alkönyvtárai, és minden alkönyvtárban további alkönyvtárakat hozhatunk létre.

A HTML-oldalakat felépítő elemek kapcsolatát is ilyen hierarchikus szerkezetben szokás ábrázolni.

```
<!DOCTYPE html>
<html lang="hu">
  <head>...</head>
  <body>
    <div class="div_1">
      <p> A DIV gyermeke vagyok</p>
    </div>
    <p> A body gyermeke vagyok.</p>
  </body>
</html>
```

A böngésző egy-egy elemnek dokumentumfán elfoglalt helyének ismeretében határozzák meg, hogy miként kell megjeleníteniük azt. A dokumentumfa segítségével az alábbi információkat tudjuk meghatározni: melyek az öröklött tulajdonságaik, hol helyezkedjenek el a képernyőn, melyik elemben helyezkednek el, melyik elem van előttük, melyik van mögöttük, mely további elemeket tartalmaznak. Emiatt szükséges az, hogy az adott elem tulajdonságainak meghatározásánál a webfejlesztőnek is ismerni kell az elemek dokumentumfán elfoglalt helyzetét. Ennek alapján lehet beállítani például csak egy meghatározott elem belsejében lévő (azaz a dokumentumfán az elem alatt elhelyezkedő) elemeknek a tulajdonságait.

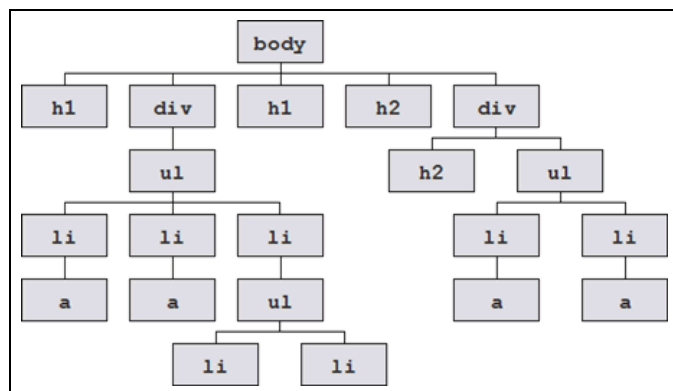


A HTML-dokumentumainkat felépítő címkéket fastruktúrába rendezhetjük úgy, ahogy elhelyezkednek egymásban - ezt a fát nevezzük dokumentumfának. A dokumentumfában az egy szinten lévő elemeknél az kerül bal oldalra, amelyik hamarabb szerepel a kódban. A fa gyökere a `html` tag, a levelei pedig beágyazott (inline) vagy olyan blokk elemek, amelyek nem tartalmaznak további tageket. A fa csomópontjaiban elhelyezkedő tageket az egymáshoz képesti hierarchikus viszonyuk szerint névvel is hivatkozhatjuk.

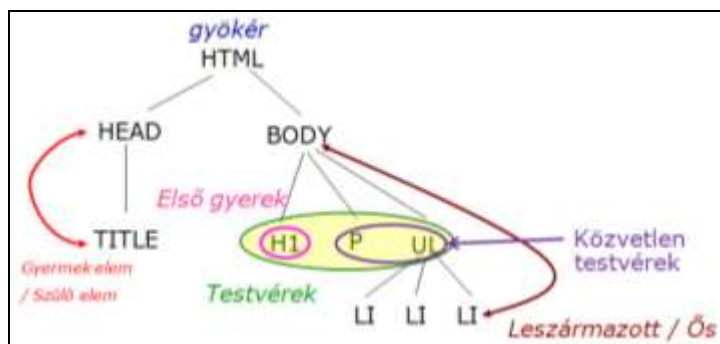


A `html` elem minden HTML5 dokumentum gyöker eleme, egyben az összes többi elem őse a `html` elem. A HTML5 szerkezeti felépítésénél láttuk, hogy a gyöker elem nyitó és záró címkéi közé egy `head` és egy `body` elemet ágyazunk be, vagyis a `head` és a `body` a `html` gyermekei, a `html` elem pedig a `head` és a `body` szülő eleme. A dokumentumfájba (`head`) ágyazott további elemek nem kerülnek megjelenítésre a weboldal tartalmaként, így azokhoz nem tartoznak stílusdefiníciók. A dokumentumtörzsben (`body`) található a megjelenítendő tartalom, így a továbbiakban a `body` elembe ágyazott elemek stílusjellemzőivel foglalkozunk.

példa egy dokumentumfára



a dokumentumfa struktúrája alapján használható elnevezések



A fában egy elem alatti részfa elemeit az **elem leszármazottai**nak, a fa az elem felett elhelyezkedő elemeit annak **ősei**nek nevezzük. **Szülő**nek hívjuk azokat az elemeket, amelyek tartalmaznak más tag-eket, **gyermekek**nek pedig a szülők első szintű leszármazottjait. (Minden elemnek pontosan egy szülő eleme van, kivéve a gyökérelemet, amelynek nincs szülő eleme.) Azokat a gyermekeket, amelyek közös szülővel rendelkeznek, **testvérek**nek nevezzük. Azok a testvérek, amelyek közvetlenül egymás után helyezkednek el a dokumentumban, **közvetlen testvérek**nek nevezzük. Egy elem gyermekeinek a gyermekeit, annak **unokái**nak nevezzük, fordítva pedig egy elem szülőjének a szülője az elem **nagyszülője**.



Öröklés és szűkítés

Amikor tehát a HTML-dokumentum objektumait egymásba ágyazzuk, azok egy fastruktúrát alkotnak és a beágyazott elemek (a leszármazottak) **öröklik** a szülő tulajdonságait.

Példa: adott az alábbi két stílusdefiníció:

```
H1 { text-color : green; }
em { text-style : italic; }
```

A `<H1>`A címsor ``igen`` fontos!`</H1>` sorban az „igen” szóra vonatkozó tag be van ágyazva egy másik (a H1) címkébe, így arra mindkét tulajdonság vonatkozik.

Ha egy attribútum értékét százalékban adjuk meg, akkor a gyermekelemek nem a szülőben megadott relatív értékeket öröklik, hanem azok számított értékét!

Példa: adott az alábbi két stílusdefiníció, amelyben a sormagasság (line-height) tulajdonság a betűméret (font-size) tulajdonsághoz képest relatív módon van megadva, tehát a sormagasság tényleges értéke 12pt (azaz a 10pt 120%-a).

```
P { font-size : 10pt; }
P { line-height : 120%; }
```

Az öröklést megszakíthatja olyan stílus, amely csak az adott objektumra vonatkozik, illetve vannak olyan tulajdonságok, amelyek nem öröklődnek: pl. padding, border, margin.

Azonban nem minden beállítás öröklődik automatikusan a szülő elemtől. Ha azonban szeretnénk, hogy egy elem a szülőtől örököljön olyan tulajdonságokat, amely alapesetben nem öröklődik, akkor az adott tulajdonságra használjuk az **inherit** értéket.

Példa: egy blokkelemre szegélyt állítunk be és azt szeretnénk, hogy a benne elhelyezett bekezdés is örökölje a szegélyre vonatkozó tulajdonságokat (mert alaphelyzetben a szegélytulajdonságok nem öröklődnek)

CSS:

```
div { border : 1px solid red; }
p { border : inherit; }
```

HTML:

```
<DIV>
  <p>Első bekezdés szövege.</p>
  <p>Második bekezdés szövege.</p>
</DIV>
```

Minden tulajdonságnak van egy kezdeti (initial) értéke, amely egy tulajdonság definíciós táblában van elhelyezve. Ha tehát vissza akarjuk állítani az alapértéket, de nem tudjuk, hogy az pontosan micsoda, használhatjuk az **initial** értéket.

A **szűkítés** esetén a nagyobb súlyú, azaz a pontosabb stílusok ütközés esetén felülbírálják a kisebb súlyúakat. Például a `body`-ra beállított stílusjellemzőket felülbírálják az egyes elemekre egyedi (elem-, osztály-, azonosító- vagy más módon) meghatározott stílusjellemzők.

Összefoglalóan tehát azt mondhatjuk, hogy **a böngésző a HTML-oldalt a stíluselemekkel együtt fentről lefelé és kívülről olvassa be és rangsorolja.** Ha nincs felhasználói stíluslap, akkor először a laphoz kapcsolt külső fájlokban lévő stílusokat,

- majd a dokumentum fejlécébe ágyazott, `<STYLE>` elemekben lévő stílusokat,
- végül a szövegközi elemekben elhelyezett `STYLE` jellemzőben található stílusokat olvassa be.

A CSS-értékek megadására vonatkozó szabályok

A CSS használata során több tulajdonságnál is megadhatunk számszerű értékeket (ide értve például a színeket is vagy bármely más értéket, amely nem egy kötött halmazból való választási lehetőséget jelent). Ezeknél az értékeknél a megadás módját mi választhatjuk meg, több alternatíva is rendelkezésünkre áll. Speciális stíluslapok esetén (melyek a hangzásokat vezérlik), szöveket, időtartamokat, frekvenciákat is megadhatunk. **A CSS-tulajdonságoknak adható értékek két nagy csoportba oszthatók: vannak előre definiált kulcsszavak és használunk bizonyos szabályoknak eleget tevő kifejezéseket.**

Az adott tulajdonságtól függően megadhatunk negatív és pozitív értékeket is. Ezek jelzésére a + és a - jeleket használhatjuk (ha nem írunk semmit, pozitív az alapértelmezett, ahogyan ezt más területeken már megszokhattuk). A mértékegységet mindig közvetlenül a szám után kell írunk, minden elválasztó nélkül. Ha nem egész számot szeretnénk használni, akkor tizedespontot kell használnunk (és nem a magyar helyesírás szerint tizedesvesszőt).

A) Hosszúságértékek

A méretek megadásakor számos mértékegység áll rendelkezésünkre, beszélhetünk abszolút és relatív értékekről. (A mértékegység megadása egy esetben elhagyható, ha az érték 0, mivel ilyenkor mindegy, hogy melyik mértékegységre gondolunk, hiszen az mindegyikben ugyanazt jelenti.)

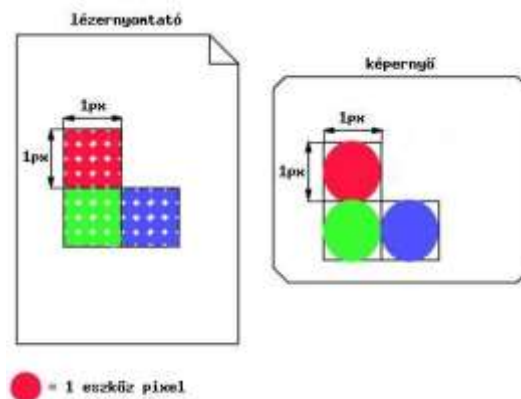
A megadott méretekből nem mindig következik az ahhoz hű megjelenés az egyes böngészőkben. Általában minden böngészőben megadható pl. egy alapértelmezett betűnagyság, ami a kizárólag relatívan megadott méretek alapjául szolgál. Az Internet Explorer nem engedi átméretezni a képpontokban megadott méreteket, a Mozilla család átméretezi, az Opera pedig zoom-olni képes, azaz még a képeket is átméretezi. Szabványos viselkedés ebből a szempontból nincsen meghatározva, pontosabban a szabványos viselkedés talán az lenne, ha semmi sem lenne átméretezhető. Az átméretezés lehetőségével szerencsére nem csorbul a szabványosság, a felhasználók számára azonban egy hasznos eszközt adnak ezzel a böngészők.

Pixel értelmezés (forrás: W3C)

Relatív mértékegységek

A relatív hosszúságértékek önmagukban sosem fejeznek ki semmit, mert értékük más érték(ek)től függ. A segítségükkel egy tulajdonság aktuális vagy megörökölt értékét változtathatjuk meg. Ide sorolhatók a px, em, ex, rem, vw, vh, vm, ch és a százalékban (%) megadott értékek.

A pixelben (px), azaz képpontban megadott értékek a megjelenítő egységtől függenek¹, általában egy konkrét képpont méretét jelölik, bár előfordulhat, hogy több pontot is magába foglal egy adott pixelyi érték (például lézernyomtatóknál a beállított felbontástól függően, vagy egy folyadék-kristályos kijelzőn, ha kisebb felbontásra van állítva, mint amennyit meg tud jeleníteni). A pixelek jele a px (egy lehetséges beállítás: pl. font-size: 20px).



Az em egy relatív értéket jelöl (ez az egyik legősbibbi mértékegység a weben), tipikus tipográfiai mértékegység: az adott betűtípusban az M betű magasságához viszonyított érték. Egy lehetséges érték például az 1.2em. (Segítségével más objektumok is méretezhetők a felhasználó által beállított betűmérethez viszonyítva.) Az 1em az adott környezet betűméretét jelöli = a font-size aktuális értéke, illetve ha konkrétan egy font-size definíción belül használjuk, akkor a szülő elem betűmérete. (1em = aktuális betűméret és pl. ez 12px ⇒ 2em = 24px.) Alapértelmezett esetben, ha a dokumentum gyökerében használjuk ezt a méretet, 16 pixelt szokott jelenteni² (böngésző beállításoktól függően). Egyetlen problémája az öröklődés miatt kialakuló érték duplázódás. (Ha például a <div> elemre ráteszünk egy 20px-es betűméretet és a <div>-en belül, szerepel egy újabb <div>, akkor annak a betűmérete már 40px lesz.)

Az ex szintén egy relatív értéket jelöl, és hasonlít az em-hez, azzal a kivétellel, hogy ez nem az adott betűkészlet magasságához viszonyított méretet, hanem a betűkészlet kis x betű magasságát jelöli (bár



¹ Bár a CSS szabvány a pixelt (képpontot) relatív mértékegységek közé sorolja, kerülendő a használata a betűméret megadása során. Sajnos több böngészőprogram nem képes nagyítani az így megadott méretű szöveget, ami hátrányos a gyengén látó felhasználóknak. Használjuk inkább a többi relatív méretmegadást ebben az esetben!

² Népszerű megoldás a body méretét 62,5%-ra állítani, ami 10px értéknek felel meg. Innentől 1em = 10px, amely leegyszerűsíti a számolgatást, hiszen akkor pl. 1.1em = 11px, 2.5em = 25px stb.

azoknál a betűkészleteknél is definiálva van, ahol nem létezik x betű). **Az x-magasság a középvonaltól és az alapvonal közötti távolság, ami tipikusan az "x" betű magassága.** A **ch** is hasonló, csak **a nulla karakterhez igazítja a betűméretet.** Általában a böngészők az értékét egyszerűen az em felének veszik, így alapesetben általában 8 px-nyi méretet jelöl. Egy lehetséges érték például az 2ex.



Az egyik fórumon úgy definiálta valaki, hogy "ugyanaz mint az em csak magasság kiadásban". **A lényege az lenne, hogy a kisbetűs részhez tudjuk igazítani a szöveg magasságát.** Jó példa lehet hozzá az, ha a felső és alsó index böngésző által renderelt pozícióját szeretnénk megváltoztatni.

Példa: `<style> sup, sub { position: relative; } sup { bottom: 1ex; } sub { bottom: -1ex; } </style>`

A **rem** (root em) **a <html> elemet veszi alapul**, tehát 1rem = a <html> elemre beállított betűméret. A betűméret nem öröklődik, hanem mindig a HTML elem méretéhez képest lesz kiszámolva, ami alapértelmezetten 16px. Ha a rem mértékegységet használjuk és később meg akarjuk növelni a betűk méretét, elegendő a <html> elemen átírni és az egész oldal ezzel arányosan fog módosulni. (Nem csak betűk méretezéséhez használható, hanem szélességet, padding-et és margin-t is állíthatunk vele.)

A **százalékos értékek** egy kicsit bonyolultabbak: **mindig relatív értéket képeznek valamely más értékhez viszonyítva, a kérdés csak az, hogy éppen milyen értéket vesznek alapul.** Ez általában a leglogikusabb érték, de hogy egyértelmű legyen, minden tulajdonságnál, ahol használhatunk százalékos megadást, definiálva van, hogy mihez képest értendő. Egy lehetséges érték pl. az 50%.



Az ex, em és a százalékos megadások egy érdekes tulajdonsága, hogy öröklődés esetén (például egymásba ágyazott listáknál) **a hatásuk az lehet, hogy egyre kisebb értéket vesznek fel**, hiszen itt a szülő értékéhez viszonyítja a százalékos (ha éppen a szülőhöz történik a viszonyítás, és a szülőnek is már százalékos érték volt adva, akkor a két százalékos érték eredője lesz az új érték, stb).

A **viewport** tulajdonságot és annak állíthatóságát főként a mobil eszközök megjelenésének köszönhetjük. **A viewport nem más mint a böngésző azon területe, ahol a weboldalak megjelennek.**³



A mobil eszközök térhódításával a tervezőknek szembesülniük kellett a telefon kisebb fizikai mérete által okozott megjelenítési korlátokkal. Ha mobil eszközön akarunk megnézni egy nagy méretű weboldalt, akkor csak a beállított viewport értéknek megfelelő részt fogunk látni az oldalból (ami alapesetben a kijelző mérete lenne), a többi esetlegesen túl csordul a megjelenítési területen. A mobil böngészők a jobb használat érdekében felül írják a kezdeti viewport-ot (vagyis a kijelző fizikai méretét), mégpedig jóval nagyobbra. A Safari mobil böngészője esetében a szélesség 980 px lesz. Ezen a szélességen a legtöbb mai oldal bőven elfér, igaz az így megjelenített oldal nem a 100%-os méretében jelenik meg, hanem kicsinyített méretben, amit aztán nagyíthatunk. Ez a megoldás a navigálás szempontjából sokkal kézenfekvőbb, mint az amikor a weboldal területéből csak annyi jelenik meg, amennyi épp elférne a kijelzőn.

Alaphelyzet: elkészítettük életünk első responsive oldalát és megnézzük mobil eszközön. Érdekes módon az oldal a nagyobb kijelzőkre beállított szerkezettel jelenik meg, nem pedig az erre a méretre beállított szerkezettel. Ezt a jelenséget a nézetablak meta tag hiánya okozza. Nem állítottuk be, hogy hogyan jelenjen meg az oldalunk a kisebb kijelzővel rendelkező eszközökön. Nincs más dolgunk, mint a következő sort hozzáadni a HTML dokumentumunk fejlécébe:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

A fent bemutatott jelenséget, a már említett, a mobil böngészőkben deklarált alap viewport beállítás okozza, vagyis hogy bepréselik a megjelenítendő oldalt a kijelző keretei köze (baloldali ábra). Tehát a viewport (nézetablak) meta elem segítségével felülírhatjuk a böngésző (user agent) által deklarált nézetablakot.

Hogyan lehet tesztelni szabni a viewport meta-tagot? **A meta elemen belül a content tulajdonság értékadásával szabályozhatjuk a fő tulajdonságait és azok értékét, amelyben a tulajdonság – érték párokat vesszővel elválasztva kapcsolhatjuk össze.**

A width tulajdonsággal értelemszerűen a viewport terület szélességét szabályozhatjuk. Például, ha a telefonra készített oldalunk szélessége 320px, akkor a következő meta-val igazíthatjuk a nézetablak területét az oldalunkhoz.

```
<meta name="viewport" content="width=320">
```

Ha több felbontású eszközre készítettük az oldalunkat, akkor választhatjuk a dinamikus érték megadást, az eszköz fizikai kijelzőmérete alapján: `<meta name="viewport" content="width=device-width">`.

A width-hez hasonlóan természetesen **itt is van height tulajdonságunk**, amit azonban kevésbé használunk, mivel a napjainkban készített oldalaknál a vertikális irány (le/fel görgetünk) a meghatározó, így az esetek többségében az oldalainknál csak a szélességet kell megadnunk: `<meta name="viewport" content="width=device-height">`

³ Forrás: Bevezetés a responsive tervezésbe – viewport (Laki Ádám honlapja: <https://adamlaki.com/hu/bevezetes-responsive-tervezesbe-viewport/>)

A megfelelő megjelenítés érdekében tudjuk manipulálni a nagyítás mértékét.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Az *initial-scale* tulajdonság segítségével megadhatunk min és max értéket egyaránt:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
    minimum-scale=1.0, maximum-scale=3.0">
```

A *user-scalable* tulajdonságát "no" értékre állítva letilthatjuk a felhasználói zoomolást. Ilyenkor célszerű a *minimum/maximum scale* tulajdonságot az *initial-scale* tulajdonsággal azonosra állítani. Valószínűleg nem túl szerencsés, ha a felhasználótól elvesszük a nagyítás lehetőségét. Figyeljünk rá, hogy csak nagyon indokolt esetekben használjuk!

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
    minimum-scale=1.0, user-scalable=no">
```

A W3C elkészítette a hivatalos viewport szabványt. Mivel a viewport lényegében a megjelenésért felel, így az általuk kidolgozott szabványban a CSS-ben van a helye. A viewport-ot CSS-ben ugyanazokkal a tulajdonságokkal használhatjuk, mint HTML-ben a meta tagot. Főbb eltérés, hogy az *initial-scale* tulajdonság helyett itt *zoom*-ot kell használnunk.

```
@viewport { zoom:1.0; width:device-width; min-zoom:1; max-zoom:3; zoom:fixed; }
```

Azzal, hogy a viewport-ot a CSS-ben kezeljük lesz egy előnyünk. Egészen egyszerűen állíthatunk be Media Query segítségével különböző kijelző felbontásokhoz, más és más viewport értéket.

```
@media screen and (min-width: 640px) and (max-width: 1024px) {
    @viewport { width: 640px; } }
```

A nézet ablak beállításoknak köszönhetően akár dinamikusan is beállíthatjuk a megjelenítéshez használt terület méreteit, így a viewport meta egy nagyon fontos része a responsive tervezésnek.

A nézetablak nagysága tehát az eszköztől függően változik, pl. kisebb lesz a mobiltelefonon, mint a számítógép képernyőjén. A táblagépek és a mobiltelefonok előtt a weboldalakat csak a számítógép képernyőjére tervezték, és gyakori volt, hogy a weboldalak statikus kialakításúak és rögzített méretűek voltak. Aztán amikor elkezdtünk szörfözni az interneten táblagépek és mobiltelefonok segítségével, a rögzített méretű weboldalak túl nagyok voltak ahhoz, hogy elférjenek a nézetablakban. Ennek kijavításához az adott eszközök böngészői kicsinyítették a teljes weboldalt a képernyőnek megfelelően. Ennek alapján a mértékegységeket ma már **a nézetablak (viewport) méreteihez (szélességéhez vagy magasságához vagy a kisebbik / nagyobbik mérethez) viszonyítva** is meghatározhatjuk: ilyen a **vw** (viewport width), **vh** (viewport height), **vmin** (viewport minimum) és a **vmax** (viewport maximum) mértékegység, azaz

- 1vw = a viewport szélességének 1%-a,
- 1vh = a viewport magasságának 1%-a,
- 1vmin = a viewport rövidebbik oldalának 1%-a,
- 1vmax = a viewport hosszabbik oldalának 1%-a,
- tehát ezek a mértékegységek egy egysége mindig a látható viewport századrésze.

Fekvő képarány esetén $1vmax = 1vw$, álló képarány esetén $1vmax = 1vh$.

A szülőelemhez képest tudjuk csökkenteni vagy növelni betűk méretét a *smaller* és a *larger* kulcsszavakkal. (Ezek 1,2-szeres szorzót használnak, tehát ha a szülő 16px nagyságú, akkor a *larger* kulcsszóval a beágyazott elem $16 \cdot 1,2 = 19.2$ px méretet fog kapni.)

Abszolút mértékegységek

Az abszolút mértékegységek neve onnan ered, hogy az adott értékhez hozzátartozik a hossz-mérték (egység), amiben értelmezendő, így elvileg minden esetben a szemlélő számára ugyan méretet jelentenek. Ezek az in, a cm, a mm, a pt és a pc.⁴

- Az **in** az inch (col, hüvelyk) méretet jelöli, azaz 25.4 millimétert.
- A **pt** a pont tipográfiai mértékegység rövidítése, ahol egy pont 1/72 hüvelyket jelöl. (Gyakran összekeverik a pixellel, pedig nem sok közük van egymáshoz.)
- A **pc** a pica rövidítése, ahol 1 pica = 12 pont.

⁴ Például 1 cm az mind a kijelzőn, mind nyomtatásban egy centiméter (kellene, hogy legyen). Egy gond van ezzel: az operációs rendszer nem mindig tudja a megjelenítőről (főleg a kijelzők esetén) hogy mekkora a pontos felbontása, és így egy általános vagy becsült értékkel dolgozik.

Az abszolút mértékegységek között is találunk előre definiált kulcsszavakat a CSS-hez, amelyek mindegyikéhez tartozik egy konkrét méret, amit a böngésző határoz meg:

- **xx-small** = 9px (többszörös kicsinyítés)
- **x-small** = 10px (kicsinyített méret)
- **small** = 13px (kicsi méret)
- **medium** = 16px (közepes méret)
- **large** = 18px (nagy méret)
- **x-large** = 24px (nagyobb méret)
- **xx-large** = 32px (többszörösen nagyobb méret)
- **inherit** = a szülőelemtől örökölt méret



Melyik célszerű alkalmazni? Képernyőn való megjelenítéshez nem ajánlott a méret pontban történő megadása, mert nem egyértelmű, hogy egy pontnak hány pixel fog megfeleltetni. Hasonlóan, az előre definiált méretnevek értelmezése is böngészőfüggő lehet. Ugyanakkor a pixelben való méret megadás sem ajánlott, mert elvileg nem teszi lehetővé gyengén látók számára a betűk méretének növelését. A böngészők ebben az esetben is biztosítják a betűméret növelésének lehetőségét, de ez nem várható el tőlük. Az em-egység az adott betűtípusban az M betű magasságát, az ex-egység pedig az x betű magasságát jelöli. A böngészők az x méretét többnyire az em felének veszik. Ezen mértékegységek segítségével más objektumok is méretezhetők a felhasználó által beállított betűmérethez viszonyítva. A százaléktérték számítása a szöveget beágyazó szülőelem betűmérete alapján történik. A fentiek miatt a betűméret beállításánál ajánlott az em-egység vagy százaléktérték használata. Továbbá figyelni kell arra, hogy a betűméret növelésekor is a teljes szöveg látható maradjon. A kinyomtatott lapon nincsenek pixelek, ezért nyomtatáskor más mértékegységeket használunk, mint például cm, mm, in, pt, pc, tehát az abszolút hosszsmértékeket.

B) URL-k megadása

Az url(...) segítségével külső objektumokra hivatkozhatunk, ami a teljesség igénye nélkül lehet: CSS-fájl, JavaScript-fájl, HTML-fájl, videó vagy akár kép. **Egy ilyen érték az url kulcsszóval kezdődik, amit zárójelek között, a hivatkozott objektum pontos relatív vagy abszolút címe követ:**

```
background-image: url('kepek/hatter.jpg');
list-style: url("http://www.vala.hol/kep.png") disc;
```

Néhány kontextusban az URL megadása stringként (karakterláncként) is lehetséges az url() használata nélkül. Például a stíluslap importálásánál egyaránt működik az alábbi két változat:

```
@import url("nyomtat.css");           @import "nyomtat.css";
```

(Az aposztróf helyettesíthető idézőjellel, illetve el is hagyható, ha a hivatkozott URL nem tartalmaz szóközt és egyéb speciális karaktert.)

C) Színek megadása

Színeket többféle objektumnak megadhatunk: színt kaphatnak például a karakterek, vonalak, sőt még a háttér vagy a szegély is, azonban a szín kialakítása minden esetben ugyanazon elvek alapján történik. (Maguknak a színeknek a megadása többféle módon is történhet. Az arculattervezés során használt grafikai programok is számos színkoordinárendszer ismernek, így szabadon választhatunk, hogy melyik szerint adjuk meg a színeket.) A színt meghatározó érték sokféle lehet, megadhatjuk kulcsszóval, de akár kódokkal is.

A színkeverés három alapszínből valósítható meg. A színek keverésében kétféle keverési módot különböztetünk meg.

- **Az egyik az additív, vagyis összeadó színkeverés, amely a színes fények keverését jelenti. Három alapszíne a vörös, a zöld és a kék, ezért hívják RGB-színsémának is.** Általában ezt alkalmazzuk akkor, ha valamilyen számítógépes kijelzőn szeretnénk színeket használni.
- **A másik módszer a szubsztraktív, azaz a kivonó színkeverés⁵, amikor a fehér fényből kivonunk színeket (hullámhossztartományokat).** Ez a színes anyagok, folyadékok és pigmen-

⁵ Ez történik minden olyan esetben, amikor saját fényforrással nem rendelkező összetevők színei keverednek (pl. papírra vitt festéknél). Ilyenkor a megvilágító fehér fény adott spektrumainak elnyelésére való képességek adódnak össze, vagyis egy új színösszetevő új hullámhossztartományt von ki a visszaverődő fényből – innen a kivonó színkeverés elnevezés.

tek keverése. **A szubtraktív színkeverés nyomdászatban használt változatának alapszínei a türkizkék, a bíbor és a sárga.** A színekódolásnál a **CMYK** (cyan, magenta, yellow, key) rövidítést használják, amelyben a „key” szó a fekete színre, mint nyomdászati kulcsszínre utal. A színkeverési módszer lényege, hogy ha a piros színt szeretnénk létrehozni, akkor sárga és bíbor színeket keverünk, mivel ezek elnyelik a kéket, a türkizkékét és a zöldet, a visszavert fényben csak a piros szín hullámhossza marad meg.

Színek nevei

A CSS2 szabványban 16 színt már **az angol nevükkel** is megadhattunk: white (fehér), black (fekete), red (piros), green (zöld), blue (kék), gray (szürke), silver (ezüstszerű), yellow (sárga), purple (lila), aqua (akvamarinkék), navy (mélykék), teal (pávakék), fuchsia (mályvaszín), maroon (gesztenyebarna), lime (világoszöld), olive (olajzöld). **A mai modern böngészők már 147 színnév használatát támogatják** (ld. https://www.w3schools.com/cssref/css_colors.asp), ebből 17 az alapszínek száma.

Hexadecimális RGB-kód

A legelterjedtebb módszer szerint a három RGB-alapszín (vörös, zöld, kék) intenzitását kétjegyű hexadecimális számokkal⁶ adjuk meg # jellel az elején. Így egy szín kódja

r₁ r₂ g₁ g₂ b₁ b₂

alakú, amelyben r₁r₂ a piros, g₁g₂ a zöld és b₁b₂ a kék szín intenzitását adja meg. A #r₁r₂g₁g₂b₁b₂ megadásnál **az r₁r₂, g₁g₂ és b₁b₂ értékek hexadecimálisan, a 00 és FF között adják meg az adott szín erősségét.** Közöttük a 00 a legsötétebb és az FF pedig a legvilágosabb. A hexadecimális számoknál a 9 feletti értékeket jelképező betűket mind kisbetűvel, mind nagybetűvel megadhatjuk.

Például a #FF0000 a piros, a #00FF00 a zöld, a #0000FF a kék, a #000000 a fekete és a #FFFFFF a fehér színeket határozzák meg. Ily módon összesen 256 * 256 * 256 = 16.777.216 színárnyalatot kaphatunk. Abban az esetben, **ha r₁=r₂=r, g₁=g₂=g, és b₁=b₂=b, akkor rövidítve is írható a #rgb formában.** Például a #A0C jelentése #AA00CC.

A leggyakoribb színekódok:

#FF0000	piros	#0000FF	kék	#00FF00	zöld
#000000	fekete	#FFFFFF	fehér	#FFFF00	sárga
#FF00FF	magenta	#00FFFF	türkiz		

(Itt található a W3C színválasztója: https://www.w3schools.com/colors/colors_picker.asp.)

Mivel a (256*256*256=) 16 777 216-féle színárnyalatot még az emberi szem sem képes megkülönböztetni, illetve az egyes megjelenítőtípusoknál egyébként is kisebb-nagyobb különbségek vannak a színek megjelenítése között, valamint az, hogy nem mindegyik eszköz támogatja az összes színárnyalat megjelenítését, ezért célszerű a színek egy köréből választani az alábbiak szerint. **A 00, 33, 66, 99, CC, FF karakterpárok kombinációiból előállított összesen 216 színt webtűró (web-biztos) színeknek nevezzük**, mert ezek elvileg minden eszközön és operációs rendszeren majdnem azonos árnyalatban jelennek meg (azaz számítógéptől függetlenek).

Decimális RGB-kód⁷

Az alapszínek intenzitását a 0 és a 255 közötti egész számokkal is megadhatjuk: rgb(r,g,b), pl. rgb(122, 205, 15).

Százalékos RGB-kód⁸

Az RGB-kódok megadása százalékos értékekkel is lehetséges: rgb(rrr%,ggg0%,bbb%), ekkor az egyes komponensek értékei csak a 0% és 100% közötti zárt intervallumba eshetnek. (A 0% éppen nullát, a 100% pedig 255-öt jelent, illetve a százalékjelet közvetlenül a szám után kell írni.)

⁶ Az értékeket 16-os számrendszerben adhatjuk meg, azaz egy számjegy 0-tól F-ig vehet fel (összesen 16-féle) értéket.

⁷ Nem mindegyik böngésző támogatja, ezért használata nem ajánlott.

⁸ Nem mindegyik böngésző támogatja, ezért használata nem ajánlott.

HSL-kód

A színeket megadhatjuk úgy is, hogy a színárnyalatot (hue), a színtelítettséget (saturation) és a világosságot (lightness) határozzuk meg. A HSL-kód szintaxisa: `hsl(hhh, sss%, lll%)`. A színárnyalat értékét a [0;360] intervallumba tartozó egész számmal, a telítettséget és a világosságot pedig a [0%;100%] intervallumból vett egész százalékos érték jelenti: pl. `hsl(240, 100%, 50%)`. A színárnyalatoknál a 0 a piros, a 120 a zöld és a 240 a kék színt jelölő érték. A telítettség esetén a 0% a szín szürke árnyalatát, a 100% pedig a teljes színt jelenti. A világosságnál a 0%-os érték a teljesen sötétet, az 50%-os érték a világosabb színt, míg a 100% az egészen világos színre utal.

RGBA-kód, HSLA-kód

A CSS3 egyik gyakran használt új funkciója az átlátszóság. A grafikai programokban is rendszeresen alkalmazunk átlátszó színeket vagy rétegeket. Az átlátszóság mértékét *transparency*, az átlátszatlanságot *opacity* néven találhatjuk általában a programokban, illetve (mint itt is) az úgynevezett **alfa csatorna (rövidítve A) segítségével tudunk átlátszóságot beállítani.** A CSS-szabványban az alfa csatorna értéke a [0;1] intervallumból vett (valós) érték, amelynél a 0 érték a teljesen átlátszót, az 1 pedig a teljesen átlátszatlan változatot jelenti.

Ha egy RGB-kódhoz szeretnénk alfa csatornát megadni, akkor az rgb helyett az **rgba szerinti kód megadást kell használni úgy, hogy a három színösszetevő mögött negyedikként az átlátszóság mértékét jelző értéket kell megadni:** `rgba (red, green, blue, alpha)`, pl. `rgba(100,25,42,0.3)`.

A hsl esetén ugyanígy kell eljárni: **a hsla szerinti kódban a negyedik jelöli az átlátszóság mértékét:** `hsla (hue, saturation, lightness, alpha)`, pl. `hsla(120,100%,25%,0.8)`.

D) Szögek megadása

Több CSS3-tulajdonságnál is szükséges lehet szög értékének meghatározására. Ezt háromféle módszerrel is megadhatjuk: **szöggel** (`deg`, pl. 120deg), **radiánnal** (`rad`, pl. 1.5rad) vagy **gradienssel** (`grad`, pl. 100grad).

E) Számított értékek

A `calc(kifejezés)` függvény egy számítást hajt végre, amelyet azt követően tulajdonságértékként kell felhasználni. Pl. a `width: calc(100% - 100px)`; stílusdefiníció egy olyan szélességet jelöl, amely a szülőelem teljes (100%) szélességénél 100 képponttal kisebb méretet határoz meg. A számítás megadásánál a kifejezés egy olyan matematikai kifejezést tartalmazhat, amelyben a +, a -, a * és a / műveleti jelek alkalmazhatók.

Megjelenítési jellemzők

A `<link>`, illetve a `@import` típusú stíluslap-használat mindegyike tartalmazhatja a mediára utaló jellemzőt, amellyel egy konkrét eszközökhöz szánt stíluslapok összerendelésére van lehetőség.

A `media` jellemző lehetséges értékei:

- `all` (minden)
- `aural` (hangeszközök)
- `Braille` (Braille-írás megjelenítésére alkalmas eszközök)
- `embossed` (domborított megjelenítés)
- `handheld` (kézi eszközök)
- `print` (nyomtató)
- `projection` (kivetítő)
- `screen` (képernyő)
- `TTY` (szövegkijelzős telefon)
- `TV` (televízió)

Még a korszerű böngészők is csak néhányat ismernek ezek közül, tehát nem tudják az összeset kezelni, de az `all`, a `print` és `projection` értékek az esetek többségében használhatók, többségük viszont már elavultnak számít a HTML5-ben.

A CSS-stíluslapok csatolásánál a HTML-ben a <link> tag media jellemzőjének print értékével meg lehet adni, hogy nyomtatáskor melyik stíluslap legyen használva. Abban az esetben, ha általában minden eszközre vonatkozó stílushoz képest csak néhány módosítást szeretnénk, akkor egyszerűbb a nyomtatási stíluslapot a mindenre érvényes (media="all") stíluslap után beilleszteni és abban csak a különbségeket leírni. Például:

```
<LINK rel="stylesheet" type="text/css" href="stilus1.css" media="all">
<LINK rel="stylesheet" type="text/css" href="stilus2.css" media="print">
```

Milyen egyéb jellemzőket lehet még meghatározni?

Érték	Magyarázat, leírás
aspect-ratio	A megjelenítő szélességi és magassági aránya („képarány”), amelyhez a min- és a max- előtag is használható. Példa: <code>media="screen and (max-aspect-ratio:16/9) "</code>
color	A színmélység, azaz a megjelenítő esetén alkalmazott bitek száma színteleppontként, amelyhez a min- és a max- előtag is használható. (Nulla esetén a kijelző nem színes.) Példa: <code>media="screen and (min-color:3) "</code>
color-index	A megjelenítő által kezelhető színek száma, amelyhez a min- és a max- előtag is használható. Példa: <code>media="screen and (min-color-index:256) "</code>
device-aspect-ratio	Elavult! Méretarány = a device-width szélesség osztva a device-height magassággal, amelyhez a min- és a max- előtag is használható. Példa: <code>media="screen and (device-aspect-ratio: 16/9) "</code>
device-width	Elavult! Folyamatos média esetén a képernyő szélességét jelenti, lapozható média esetén az oldallap szélességét, amelyhez a min- és a max- előtag is használható. Példa: <code>media="screen and (device-width: 600px) "</code>
device-height	Elavult! Folyamatos média esetén a képernyő magasságát jelenti, lapozható média esetén az oldallap magasságát. Példa: <code>media="screen and (device-height: 400px) "</code>
grid	Megadja, hogy a kimeneti eszköz rács vagy bitkép: a lehetséges értékek a rács esetében "1", különben a "0" érték. Példa: <code>media="handheld and (grid:1) "</code>
height	Folytonos média esetén a viewport magasságát jelenti a gördítősávot (ha van) is beleértve, lapozható média esetén ez az oldal doboz magasságát jelenti, amelyhez a min- és a max- előtag is használható. Példa: <code>media="screen and (max-height:700px) "</code>
monochrome	A pixelenkénti bitek számát határozza meg monokróm keretpufferben, amelyhez a min- és a max- előtag is használható. Példa: <code>media="screen and (min-monochrome:2) "</code>
orientation	A kimeneti eszköz (megjelenítő vagy papírlap) orientációját, azaz tájolását jelenti, amely álló vagy fekvő lehet. Akkor álló (portrait) az elrendezés, ha a magasság nagyobb vagy egyenlő a szélességgel, ellenkező esetben az fekvő (landscape). Példa: <code>media="all and (orientation: landscape) "</code>
resolution	A megjelenítő felbontása (dpi vagy dpcm), amelyhez a min- és a max- előtag is használható. Example: <code>media="print and (min-resolution:300dpi) "</code>

Érték	Magyarázat, leírás
scan	A frissítés (szkenelés) típusát határozza meg, amelynek lehetséges értékei: progressive vagy interlace. Példa: <code>media="tv and (scan:interlace)"</code>
width	Folytonos média esetén a viewport szélességét jelenti a gördítősávot (ha van) is beleértve, lapozható média esetén ez az oldaldoboz szélességét jelenti, amelyhez a min- és a max- előtag is használható. Example: <code>media="screen and (min-width:500px)"</code>

A CSS-stíluslapok csatolásánál a CSS-ben a @media alkalmazásával adhatunk meg szabályokat az egyes médiatípusok vagy eszközök használata során az eltérő stílusok alkalmazásához.

Ennek segítségével sokféle információt ellenőrizhet: a nézetablak szélességét és magasságát, az alkalmazott eszköz szélességét és felbontását, a tájolást (pl. hogy a tablet vagy telefon tájkép vagy álló módban van-e), a felbontást. A *média-lekérdezések használata egy népszerű technika a szélmélyre szabott stíluslapok (érzékeny webdesignok) különféle típusú eszközök (asztali számítógépekre, laptopokra, táblagépekre és mobiltelefonokra) történő alkalmazásához*. Ezzel azt is megadhatja, hogy bizonyos stílusok csak nyomtatott dokumentumokhoz vagy képernyőolvasókhoz legyenek használhatók (mediatype: nyomtatás, képernyő vagy beszéd). A média-lekérdezések során a média típusa mellett a média jellemzőit is meg lehet tudni, így lehetővé válik, hogy például egyes stílusokat csak azokhoz a képernyőkhöz alkalmazzunk, amelyek nagyobbak vagy kisebbek, mint egy bizonyos szélesség.

Például: létrehozunk három egyforma oszlopot, amelyek egymás mellé vannak úsztatva:

```
.column {
  width: 33.33%;      /* minden oszlop szélessége a befoglaló elem szélességének harmadával egyezik meg
  float: left;        /* az egyes oszlopokat egymás mellé „csúsztatjuk” balra „húzva”
  padding: 15px;
}
```

Egy reszponzív elrendezés kialakítása során a három oszlopot egymás alá helyezi ahelyett, hogy azok egymás mellett lennének, de csak akkor, ha a megjelenítő mérete egy adott értéket elér vagy kisebb:

```
@media screen and (max-width: 600px) {      /* 600px vagy kisebb felbontású képernyő esetén
  .column {
    width: 100%;      /* minden oszlop szélessége a befoglaló elem szélességével egyezzen meg
  }
}
```

Milyen fontosabb változtatásokat érdemes elvégezni a nyomtatási stíluslapban?

Egyrészt **érdemes elrejtetni az olyan tartalmakat, melyeket a felhasználó a kinyomtatott oldalon úgyse tudna hasznosítani**. Ilyenek lehetnek például a keresőmezők, menük, linkek és gombok is.

Az elrejtésnek két módja van:

- Az egyik esetben a **visibility** tulajdonságot használjuk, amelynek lehetséges értékei **inherit** (örökölt), **visible** (látható) és **hidden** (rejtett).
- A másik lehetőség a **display** tulajdonság használata a **none** értékkel.

A két megoldás között az a különbség, hogy a **display: none;** kiveszi az objektumot a normál szövegfolyamból, míg a **visibility: hidden;** esetén csak üresen marad az objektum helye. Így az objektumok **display**-el történő elrejtése a felhasználó számára papírspórolást is eredményez.

Papírtakarékosság szempontjából ugyanakkor hasznos lehet az is, ha **a nyomtatási oldalon, a weben megszokott keskeny sorokkal szemben, szélesebb sorokat használunk**. Általános vélemény, hogy míg **a képernyőn a talp nélküli betűket** (pl. Verdana, Arial, sans-serif), **nyomtatásban a talpas betűket** (pl. Georgia, Times, serif) könnyebb olvasni. A kinyomtatott lapon nincsenek pixe-

lek, ezért **nyomtatáskor más mértékegységeket használunk**, mint például cm, mm, in (inch vagy hüvelyk, kb. 25.4 mm), pt (pont = in/72), pc (pica = 12 pt).

A nyomtatási stílusban ajánlott megadni a margókat és a betűméreteket is!



Ugyanakkor érdemes figyelembe venni, hogy a felhasználó számára a böngészőben számos nyomtatási beállítás rendelkezésre áll (pl. nyomtasson-e háttérképet, oldalszámot, URL címetek stb.). A felhasználó többnyire nem nyomtatja ki a hátteret, ezért egyrészt ne bízunk abban, hogy a nyomtatásban is lesz háttérkép, másrészt gondoskodjunk arról, hogy a nyomtatásban a fehér lapon is láthatók legyenek a betűk. Végül gondoljunk azokra a felhasználókra is, akik nem színes nyomtatóval nyomtatnak, így olyan színeket adjunk meg, amelyek akkor is látszani fognak, ha azokat fekete-fehér nyomtatóval nyomtatják ki.

Példa: e mintában egy olyan HTML-oldal fejléce látható, amelyhez többféle stíluslapot is kapcsoltunk

```
<!DOCTYPE html>
<html lang="hu">
<head>
  <title> Részletes információk </title>
  <link rel="stylesheet" type="text/css" href="alapstilusok.css">
  <link rel="stylesheet" type="text/css" href="kiegstilus.css" media="all">
  <link rel="stylesheet" type="text/css" href="nystilus.css" media="print">
  <meta charset="utf-8">
</head>
```

kiegstilus.css

```
body { background-image: url(kepek/tel.jpg); }
p.elter { line-height: 1; }
p.nenyomtassa { font-size: 2.2em;
  font-style: italic;
  text-transform: capitalize;
  text-align: center;
  text-indent: 0;
  line-height: 2em;
  background-color: #fc9;
  color: #60f;
  width: 75%;
  margin: 0 10% 0 10%;
  position: 0 0 0 10%;
  border: double 1em #f39; }
dt { font-weight: bold;
  line-height: 1.2em; }
dd { font-style: italic;
  line-height: 1.5em; }
```

nystilus.css

```
body { background-color: #fff;
  color: #000;
  margin: 0.5in;
  font-family: "Times New Roman", HTimes, serif;
  font-size: 14px; }
.lapcim { font-style: italic;
  font-weight: bold;
  font-size: 20px;
  color: #000; }
h2 { color: #000;
  font-size: 16px; }
p, dt, dd, p.bev, .kiemel { color: #000;
  text-align: left;
  line-height: 1.5; }
p.nenyomtassa { display: none; }
p.visszalink{ font-size: 10px;
  text-indent: 0; }
```

III. A kijelölők fajtái

A CSS-ben a **kijelölő (selector)** adja meg, hogy a stílus a HTML-lap melyik elemére vonatkozik.

A CSS választók öt kategóriába oszthatók:

- **egyszerű** kijelölők: elemek kiválasztása név, azonosító, osztály alapján
- **kombinátor** kijelölők: elemek megjelölése a közöttük fennálló kapcsolat alapján
- **attribútum** kijelölők: elemek kiválasztása attribútum vagy attribútumérték alapján
- **pszeudo-osztály** kijelölők: elemválasztás annak egy adott állapota alapján
- **pszeudo-elem** kijelölők: az elem egy részének kiválasztása

Egyszerű kijelölők

Az egyszerű kijelölők csoportját az elem-, az azonosító, az osztály- és az univerzális kijelölők alkotják.

Ebbe a csoportba sorolható az a változat is, amikor az azonos tulajdonsággal rendelkező kijelölőkre vonatkozó beállításokat összevonjuk és ezáltal lerövidíthetjük és áttekinthetőbbé tehetjük a CSS-kód tartalmát.

Az egyszerű kijelölők		
Kijelölő	Példa	A példa magyarázata
element	p	Az összes <p> elem kijelölése
#id	#firstname	Az id="firstname" tulajdonságú elem kijelölése
.class	.intro	Az összes class="intro" tulajdonságú elem kiválasztása
*	*	Az összes elem kijelölése
element, element, element, ...	div, p	Kiválasztja az összes <div> és az összes <p> elemet

A) Elemkijelölő (element/type selector)

Ha több azonos objektumot (pl. bekezdések, képek, táblázatsorok) **formázunk, akkor elemkijelölőt használunk úgy, hogy kijelölőként a HTML-tag nevét adjuk meg a CSS-ben.** Ekkor a kijelölés a HTML-dokumentumban található összes adott típusú objektumra, azaz tagra vonatkozik.

Példa: a dokumentum összes bekezdésében ugyanazt a betűtípust, betűméretet és szövegszint szeretnénk alkalmazni

```
p { font-family : Arial, sans-serif;
    font-size : small;
    color : blue; }
```

Az esetek többségében azonban szükség lehet a kevésbé általános, azaz pontosabb kijelölésre, ezért ilyenkor javasolt az azonosító-, az osztály- vagy az összetett kijelölők alkalmazása.

B) Azonosító kijelölő (id selector)

Azonosító kijelölő esetén csak az egyedi azonosítóval⁹, azaz ID tulajdonsággal rendelkező objektumra vonatkozik a formázás és ekkor a stílusleírásban # jelet követően adjuk meg az ID értékét, majd a formai jegyeket. Alkalmazásának előfeltétele az, hogy a HTML-dokumentumban a kijelölni kívánt objektum nyitó tagjában szerepel az ID="..." beállítás.

Példa: a „fontos” azonosítójú objektum tartalmát zöld színű betűkkel szeretnénk formázni

HTML: <P>Ez itt egy igen jelentős információ.</P>
CSS: #fontos { color : green; }

Mivel az elem azonosítója egy oldalon belül egyedi, ezért az azonosítónevet a HTML-lapon belül csak egyszer lehet használni! Ügyelni kell arra is, hogy ha közös stílusfájl alkalmazunk több weboldalunknál (pl. a webhelyünk összes formai jellemzőjét egyetlen CSS állományba gyűjtjük össze) és oldalanként egyszer, de több oldalunkon is ugyanazt az azonosítónevet használjuk, akkor a közös azonosító névvel ellátott elemek is azonos a tulajdonságokat vesznek fel.

⁹ A HTML 4.01-es szabványban az egyedi azonosítókra az volt igaz, hogy betűvel kezdődhetnek, amelyet számok, betűk és a - _ : jelek követhetnek. A HTML5 szabványban már nem ilyen szigorú a megkötés, az azonosítónak legalább 1 karakter hosszúnak kell lennie, nem tartalmazhat szóközt, de nem kezdődhet számjeggyel! (Ennek ellenére célszerű csak az angol ábécé betűit, a számjegyeket és az aláhúzásjelet használni.)

C) Osztály kijelölő (class selector)

Osztály kijelölő esetén az összes azonos CLASS értékkel rendelkező objektumra vonatkozik a formázás és ekkor a stílusleírásban . (pont) jelet követően adjuk meg a CLASS értékét, majd a formai jegyeket. Alkalmazásának előfeltétele az, hogy a HTML-dokumentumban a kijelölni kívánt objektumok nyitó tagjában szerepel a CLASS="..."¹⁰ beállítás.

Példa: az összes „kiemelt” osztályú objektum tartalmát piros színű betűkkel szeretnénk megjeleníteni

```
HTML: <P>Ez itt egy igen <SPAN CLASS="kiemelt">jelentős</SPAN> információ.</P>
...
<DIV CLASS="kiemelt">Ez is fontos információ...</DIV>
CSS: .kiemelt { color : red; }
```

Gyakori az, hogy egy HTML-elem egyszerre több osztályhoz is tartozik vagy több osztályt is hozzá szeretnénk rendelni: ekkor a HTML-elem nyitó tagjában az osztályneveket szóközzel választjuk el egymástól (pl. <P CLASS="fontos bev">).

D) Univerzális kijelölő (universal selector)

Az univerzális szelektor¹¹ segítségével olyan CSS-szabályok írhatók, amelyek a HTML-dokumentum minden egyes elemére érvényesek lesznek. Az univerzális szelektort egy * (csillag) testesíti meg a CSS-ben, így alkalmazásához nincs szükség arra, hogy a HTML-dokumentumba bármilyen külön tulajdonságot beállítsunk. Használatával könnyedén adhatunk globális stílusjegyeket a formázandó weblaphoz, azaz gyakran éppen azért alkalmazzuk, hogy rövidebben és egyszerűbben állítsuk be az oldalak alapbeállításait (pl. margók, betűjellemzők), majd ezt követően határozzuk meg egy-egy elem, osztály vagy azonosító tulajdonságait.

Példa: az oldal margóértékeinek törlése, azaz 0 értékre történő beállítása

```
CSS: * { margin : 0; padding : 0; }
```

Fontos elvárás, hogy a weblapunk egységesen jelenjen meg a különböző böngészőprogramokban, ezért a fenti szabály ezen elv elérését könnyíti meg (de önmagában még nem elegendő).

E) Csoportos kijelölés (group of selectors)

Ha a weboldalon több objektumnak is ugyanazokat a tulajdonságokat szeretnénk adni, akkor a kijelölőben vesszővel elválasztva felsorolhatjuk azokat.

Példa: az oldal alkalmazott 1. szintű és 2. szintű címsoroknak, valamint bekezdéseknek is egyeznek a jellemzői

```
CSS: h1, h2, p { text-align : center; color : silver; }
```

Kombinátor kijelölők

A kombinátorok (combinators) esetén a kijelölés a HTML-elemek közötti kapcsolat alapján történik. A CSS-ben négy különböző kombinátor van: a leszármazott választó (szóköz), a gyermekválasztó (>), a szomszédos testvérválasztó (+) és a testvér választó (~). Ekkor a CSS-kiválasztó egynél több egyszerű választót tartalmaz.

A) Leszármazott kijelölő (descendant selector)

Ha egy HTML-elem leszármazottaihoz, azaz a beágyazott elemeihez szeretnénk stílust hozzárendelni, akkor egy speciális szelektorlistát kell alkalmazni, amelyben a szelektorokat szóközök választják el egymástól: szelektor1 szelektor2 szelektor3 { stílusdefiníció }.

¹⁰Az osztályok nevének kialakítása során ne a hatás, hanem a funkció alapján határozzuk meg az elnevezést – így a későbbiekben sem kell változtatni rajta. Például a „kek” helyett jobb a „kiemeles” vagy a „színes” megnevezés.

¹¹Az univerzális szelektor a CSS2-es szabványban jelent meg először. Amíg ezt nem vezették be, ugyanezen célból kellett egy külön stíluslapot (CSS reset: ld. <https://meyerweb.com/eric/tools/css/reset/>) alkalmazni, amelyben az összes elem margó, kitöltés stb. értéke be volt állítva az alapértékre.

Ha a listában egy szelektor megelőz egy másikat, akkor az előbb szereplő szelektor a később szereplő szelektor őseit jelenti. Fontos, hogy a szelektorlistában egymást követő szelektorok nem feltétlenül egymás közvetlen gyermekei, illetve szülei. **A leszármazási szelektor tehát megfelel minden olyan elemnek, amelyek egy adott elem leszármazottai.** Nem szabad elfelejteni, hogy csak a szelektorlista utolsó eleme lesz stílussal ellátva!

Példa: az itt található mintában egy 3-szintű egymásba ágyazott lista látható. Ha azt szeretnénk, hogy az első szint elemeinek formátumbeállítása eltérő legyen a második és a harmadik szint jellemzőitől, akkor az egyes szinteket pontosan meg kell jelölni a stílusmegadásban:

CSS:

// összes listaelem formázása:

```
li {
  color : red;
  font-weight : bold; }
```

// belső szint(ek) formázása:

```
ul li li {
  color : green;
  font-style : italic; }
```

// a 3.szint formázása:

```
ul li li li {
  color : black;
  font-style : normal;
  font-weight : normal; }
```

A példa alapján jól látható, hogyan öröklök az egyes szintek a szülőelem tulajdonságait. A **li** szelektorra beállított betűszín és betűvastagság jellemzőit örökli az összes listaelem. Az **ul li li li** kijelölővel meghatározott attribútumok az összes olyan listaelemre érvényesek lesznek, amelyek maguk is felsorolás típusú listába vannak beágyazva, tehát a legalább 2. szinten lévő listaelemekre lesznek hatással. Az **ul li li li li** kiválasztó csak azokra a listaelemekre fog vonatkozni, amelyek legalább 3. szint mélységben vannak elhelyezve a listában.

Az itt látható 4 fázisra a fenti stílusulajdonságok alkalmazásának hatását mutatja be abban a sorrendben, ahogyan azokat a stíluslapon sorban egymást követően meghatároztuk: először a formázás nélküli, majd csak a **li**, utána csak a **li** és csak az **ul li li** elemekre, végül az összes elemre érvényes beállítással megjelenő listák tekinthetők meg.

bölcsőde

óvoda

általános iskola

alsó tagozat

felső tagozat

középiskola

szakiskola

szakközépiskola

szakgimnázium

nyelvi előkészítő

9-12. évfolyam

1-2 éves szakképzés

gimnázium

8 osztályos

6 osztályos

4 osztályos

felsőoktatás

főiskola

egyetem

KIINDULÓ ÁLLAPOT:
formázás előtt
(nincs CSS beállítás)

- bölcsőde
- óvoda
- általános iskola
 - alsó tagozat
 - felső tagozat
- középiskola
 - szakiskola
 - szakközépiskola
 - szakgimnázium
 - nyelvi előkészítő
 - 9-12. évfolyam
 - 1-2 éves szakképzés
 - gimnázium
 - 8 osztályos
 - 6 osztályos
 - 4 osztályos
- felsőoktatás
 - főiskola
 - egyetem

LI tulajdonságokkal:
az összes szint
egységes formázása

- bölcsőde
- óvoda
- általános iskola
 - alsó tagozat
 - felső tagozat
- középiskola
 - szakiskola
 - szakközépiskola
 - szakgimnázium
 - nyelvi előkészítő
 - 9-12. évfolyam
 - 1-2 éves szakképzés
 - gimnázium
 - 8 osztályos
 - 6 osztályos
 - 4 osztályos
- felsőoktatás
 - főiskola
 - egyetem

LI és UL LI LI értékekkel:
az 1. és többi szint
formázása

- bölcsőde
- óvoda
- általános iskola
 - alsó tagozat
 - felső tagozat
- középiskola
 - szakiskola
 - szakközépiskola
 - szakgimnázium
 - nyelvi előkészítő
 - 9-12. évfolyam
 - 1-2 éves szakképzés
 - gimnázium
 - 8 osztályos
 - 6 osztályos
 - 4 osztályos
- felsőoktatás
 - főiskola
 - egyetem

az összes beállítással:
mindegyik szint
külön formázásával

- bölcsőde
- óvoda
- általános iskola
 - alsó tagozat
 - felső tagozat
- középiskola
 - szakiskola
 - szakközépiskola
 - szakgimnázium
 - nyelvi előkészítő
 - 9-12. évfolyam
 - 1-2 éves szakképzés
 - gimnázium
 - 8 osztályos
 - 6 osztályos
 - 4 osztályos
- felsőoktatás
 - főiskola
 - egyetem

B) Gyermek kijelölő (child selector)

Ha azt szeretnénk kifejezni, hogy az egyik szelektor a másik gyermeke, akkor a szelektorlistában a szelektorokat > (nagyobb) jellel választjuk el: szelektor1 > szelektor2 { stílusdefiníció }.

Az előzőhöz hasonlóan, itt is csak a szelektorlista utolsó eleme lesz formázva a megadott stílussal.

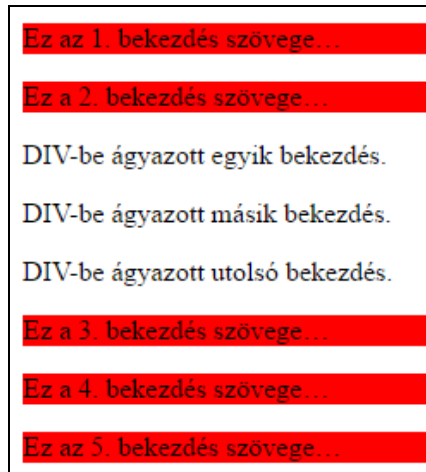
Példa: az itt található mintában egy weboldal **p** gyermekeit (vagyis a közvetlenül az 1. szintű bekezdés típusú leszármazottjait) szeretnénk azonos módon formázni

CSS: body > p { background-color : red; }

A piros háttérszín kizárólag azokra a bekezdésekre fog vonatkozni, amelyek közvetlenül a **body**-ból nyílnak, tehát az 1-2. és a 3-4-5. számmal ellátott egységekre, viszont a **div**-ekbe ágyazott bekezdésekre már nem lesz érvényes a beállítás, mert azok nem a gyermei, hanem az unokái.

HTML :

```
<BODY>
  <P>Ez az 1. bekezdés szövege...</P>
  <P>Ez a 2. bekezdés szövege...</P>
  <DIV>
    <P>DIV-be ágyazott egyik bekezdés.</P>
    <P>DIV-be ágyazott másik bekezdés.</P>
    <P>DIV-be ágyazott utolsó bekezdés.</P>
  </DIV>
  <P>Ez a 3. bekezdés szövege...</P>
  <P>Ez a 4. bekezdés szövege...</P>
  <P>Ez az 5. bekezdés szövege...</P>
</BODY>
```



C) Közvetlen testvér (adjacent sibling selector)

Szelektorlistával lehetőségünk van közvetlen testvérségi viszony kifejezésére is: a listában levő szelektorokat + jellel választjuk el, azaz szelektor1 + szelektor2 { stílusdefiníció }. (A két elemnek ugyanattól a szülőtől kell származnia.) Fontos, hogy ebben az esetben kizárólag arra a szelektor2-vel megadott testvérelemre vonatkozik a stílusbeállítás, amelynek **közvetlen megelőző** testvére a szelektor1-gyel megadott elem!

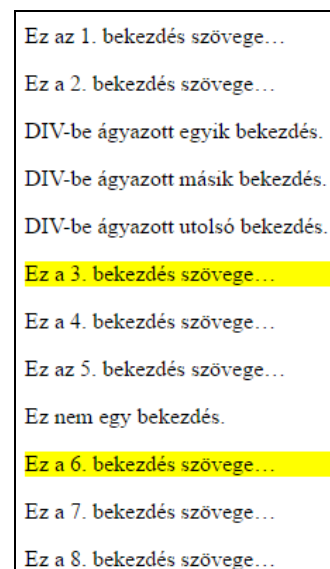
Példa: az itt található mintában egy weboldal **p** gyermekeit (vagyis a közvetlenül 1. szintű **p** leszármazottjait) szeretnénk azonos módon formázni

CSS: div + p { background-color : yellow; }

A sárga háttérszín kizárólag azokra a bekezdésekre fog vonatkozni, amelyek testvérei a **div** elemeknek és közvetlenül a **div** elemet követően állnak. A mintában ilyen bekezdés a 3. és a 6. sorszámmal megjelöltek. Nem fog hatni a formázás a többi bekezdésre, mert azok vagy megelőzik a **div** elemet vagy nem testvéri viszonyban (hanem gyermek kapcsolatban) állnak azzal.

HTML :

```
<BODY>
  <P>Ez az 1. bekezdés szövege...</P>
  <P>Ez a 2. bekezdés szövege...</P>
  <DIV>
    <P>DIV-be ágyazott egyik bekezdés.</P>
    <P>DIV-be ágyazott másik bekezdés.</P>
    <P>DIV-be ágyazott utolsó bekezdés.</P>
  </DIV>
  <P>Ez a 3. bekezdés szövege...</P>
  <P>Ez a 4. bekezdés szövege...</P>
  <P>Ez az 5. bekezdés szövege...</P>
  <DIV>Ez nem egy bekezdés.</DIV>
  <P>Ez a 6. bekezdés szövege...</P>
  <P>Ez a 7. bekezdés szövege...</P>
  <P>Ez a 8. bekezdés szövege...</P>
</BODY>
```



D) Általános testvér (general sibling selector)

Az általános testvérségi viszony kifejezésére is van lehetőség szelektorlistával, mert ekkor a szelektorokat ~ jellel választjuk el: **szelektor1 ~ szelektor2 { stílusdefiníció }**. (A két elemnek ugyanattól a szülőtől kell származnia.) Fontos, hogy ebben az esetben kizárólag arra a szelektor2-vel megadott testvérelemre vonatkozik a stílusbeállítás, amelynek megelőző testvére a szelektor1-gyel megadott elem!

Példa: az itt található mintában egy weboldal **p** gyermekeit (vagyis a közvetlenül 1. szintű **p** leszármazottjait) szeretnénk azonos módon formázni

CSS: `div ~ p { background-color : lightgreen; }`

A világoszöld háttérszín kizárólag azokra a bekezdésekre fog vonatkozni, amelyek testvérei a **div** elemeknek és a **div** elemet követően helyeztünk el a HTML-dokumentumban. A mintában ilyen bekezdés a 3-4-5. és a 6-7-8. sorszámmal megjelöltek. Nem fog hatni a formázás a többi bekezdésre, mert azok vagy megelőzik a **div** elemet vagy pedig nem testvéri viszonyban (hanem gyermek kapcsolatban) állnak azzal.

HTML :

<BODY>

<P>Ez az 1. bekezdés szövege...</P>

<P>Ez a 2. bekezdés szövege...</P>

<DIV>

<P>DIV-be ágyazott egyik bekezdés.</P>

<P>DIV-be ágyazott másik bekezdés.</P>

<P>DIV-be ágyazott utolsó bekezdés.</P>

</DIV>

<P>Ez a 3. bekezdés szövege...</P>

<P>Ez a 4. bekezdés szövege...</P>

<P>Ez az 5. bekezdés szövege...</P>

<DIV>Ez nem egy bekezdés.</DIV>

<P>Ez a 6. bekezdés szövege...</P>

<P>Ez a 7. bekezdés szövege...</P>

<P>Ez a 8. bekezdés szövege...</P>

</BODY>

Ez az 1. bekezdés szövege...

Ez a 2. bekezdés szövege...

DIV-be ágyazott egyik bekezdés.

DIV-be ágyazott másik bekezdés.

DIV-be ágyazott utolsó bekezdés.

Ez a 3. bekezdés szövege...

Ez a 4. bekezdés szövege...

Ez az 5. bekezdés szövege...

Ez nem egy bekezdés.

Ez a 6. bekezdés szövege...

Ez a 7. bekezdés szövege...

Ez a 8. bekezdés szövege...

A kombinátor kijelölők összefoglalása

Kombinátor kijelölők			
A kijelölő neve	A kijelölő szintaxisa	Példa	A példa magyarázata
leszármazott	<code>element element</code>	<code>div p</code>	Az összes olyan <code><p></code> elem kijelölése, amely <code><div></code> elemen belül található valahol a dokumentumban
gyermek	<code>element > element</code>	<code>div > p</code>	Az összes olyan <code><p></code> elem kijelölése, amelynek a közvetlen őse (szülője) <code><div></code> elem
közvetlen testvér	<code>element + element</code>	<code>div + p</code>	Az összes olyan <code><p></code> elem kijelölése, amelynek egy <code><div></code> elem testvére és közvetlenül a <code><div></code> elem mögött található
általános testvér	<code>element ~ element</code>	<code>div ~ p</code>	Az összes olyan <code><p></code> elem kijelölése, amelynek egy <code><div></code> elem testvére és mögötte található valahol a dokumentumban

Attribútum kijelölők

Az attribútum alapú kijelölésnél a kiválasztást annak alapján szabályozzuk, hogy az egyes elemek milyen tulajdonságokkal rendelkeznek és azok milyen értékeket vesznek fel. Jelölése: **szelektor[...]{ stílusdefiníció }**, azaz a kijelölőt követően szögletes zárójelben adjuk meg azokat az információkat, amelyek alapján szűrni lehet a potenciális elemeket.

A) Egyszerű jellemzőkijelölő

A **szelektor[attribútum]** esetén azt vizsgáljuk, hogy **a szelektált elem rendelkezik-e a keresett tulajdonsággal**, azaz hogy a HTML-kódban az elemhez megadásra került-e az adott jellemző.

Példa: az oldalon elhelyezett képek közül azokat formázzuk, amelyeknek van szélességbeállítása

```
CSS: img[width] { border : 2px solid red; }
```

B) Pontos jellemzőérték-kijelölő

A **szelektor[attribútum="érték"]** esetén azt figyeljük, hogy **a szelektált elem rendelkezik a keresett tulajdonsággal úgy, hogy annak értéke megegyezik a megadott értékkel**, azaz a HTML-kódban az elem egy meghatározott tulajdonságának értéke azonos a kijelölőben megadott értékkel.

Példa: az oldalon definiált egysoros szövegbeviteli mezőket formázzuk

```
CSS: input[type="text"] { width : 150px; display : block; }
```

C) Részleges jellemzőérték-kijelölő

A **szelektor[attribútum~="érték"]** esetén azt nézzük, hogy **a szelektált elem rendelkezik a keresett tulajdonsággal, amelynek értékének a megadott érték egy szóközök által határolt részsorozata** (vagyis a megadott érték előtt és hátul szóközökkel kiegészítve előfordul-e a tulajdonságnál).

Példa: az oldalon definiált hivatkozások közül azokat formázzuk, amelyeknek a **title** tulajdonságának értékében szerepel az „SZTE” kifejezés önálló szóként

```
CSS: a[title~="SZTE"] { font-style : italic; }
```

Korábban láthattuk az osztályok (**class**) használatát, amelyet most már attribútum szelektorral is leírhatók: pl. a **p.fontos** szelektor helyett akár azt is írhatnánk, hogy **p[class~="fontos"]**.

A **szelektor[attribútum="érték"]** esetén viszont arra vagyunk figyelemmel, hogy **a szelektált elem rendelkezik a keresett attribútummal úgy, hogy annak értéke tartalmazza a megadott értéket egy kötőjelek által határolt részsorozatként**.

Közelítsük meg onnan ezt a fajta attribútum szelektort, hogy vajon miért volt szükség a bevezetésére? Ha például egy szövegnél jelezni akarjuk annak nyelvét, akkor használhatjuk a **lang** attribútumot. A nyelvkódok között viszont számos olyat találunk, amelyekben kötőjel szerepel, pl. **en-gb**, **en-us**, vagyis nyelvváltozatokra utalnak (brit angol, amerikai angol). De mi van akkor, ha mi azon elemeket akarunk megcímezni, amelyek angol nyelven íródtak, és számunkra érdektelen, hogy melyik nyelvváltozatban? Akkor egy olyan szelektorra van szükségünk, amely a kötőjelek előtti karakterláncra való illeszkedést is tudja kezelni.

Példa: az oldalon bekezdések közül azokat formázzuk, amelyeket az angol egyik nyelvváltozatában adtunk meg

```
CSS: p[lang]="en"] { color : green; }
```

D) Jellemzőérték-eleje kijelölő

A **szelektor[attribútum^="érték"]** esetén azt vizsgáljuk, hogy **a szelektált elem rendelkezik a keresett tulajdonsággal úgy, hogy annak a kezdete a megadott érték**.

Példa: az oldalon elhelyezett blokkok közül azokat formázzuk, amelyeknek az osztályneve a „fej”-jel kezdődik

```
CSS: div[class^="fej"] { text-align : center; }
```

E) Jellemzőérték-vége kijelölő

A **szelektor[attribútum\$="érték"]** esetén azt figyeljük, hogy **a szelektált elem rendelkezik a keresett tulajdonsággal úgy, hogy annak a vége a megadott érték**.

Példa: az oldalon elhelyezett blokkok közül azokat formázzuk, amelyeknek az osztályneve a „teszt”-re végződik

```
CSS: div[class$="teszt"] { color : magenta; }
```

F) Jellemzőérték-részlet kijelölő

A **szelektor[attribútum*="érték"]** esetén azt vizsgáljuk, hogy **a szelektált elem rendelkezik a keresett tulajdonsággal úgy, hogy annak az értéke a megadott értéket tartalmazza.**

Példa: az oldalon elhelyezett blokkok közül azokat formázzuk, amelyeknek az azonosítójában szerepel a „resz” szó

```
CSS: div[id*="resz"] { margin-left: 20px; }
```

Pszeudo-osztályok (ál-osztályok, látszólagos kijelölők)

Egy dokumentumnak lehetnek olyan külön formázást igénylő részei, amelyekhez nem tartozik önálló elem. Ilyen lehet például egy egysoros beviteli mező színe a fókuszba kerüléskor, egy bekezdés első karakterének iniciáléként való formázása vagy az, amikor egy elemre rámutatáskor például más méretben kellene annak tartalmát megjeleníteni. Ennek tipográfiáját nem lehet a korábban tárgyalt szelektorok egyikével sem definiálni, ezért vezették be a pszeudo-osztály és -elem fogalmát.

A HTML-elemek egy része szelektálható állapotokkal rendelkezik és az ezen állapotokra illeszkedő szelektorokat is le lehet írni a CSS segítségével. **A pszeudo-osztályok (pseudo classes) egyes elemek bizonyos állapotban történő elérését teszik lehetővé**, amelyeket az alábbi szintaxissal használhatunk: **szelektor:pszeudo-osztály { stílusdefiníció }**. Ezt kell használni például akkor, ha az adott elem az első gyermek vagy ha egy hivatkozás esetén a már meglátogatott cím állapotot szeretnénk formázni. Ez utóbbi esetben egy adott elem akár az oldal használata közben el is veszítheti egy adott pszeudo-osztályba tartozását vagy meg is szerezheti ezt a tulajdonságot.

Az ál-osztályok között alapvetően két fő csoportot különböztetünk meg a viselkedésük és használatuk alapján: a dinamikus és a strukturális csoportokat.

A) A dinamikus pszeudo-osztályok és elemeik

A dinamikus pszeudo-osztályokba azokat a megvalósításokat sorolhatjuk, melyekkel az egyes elemek dinamikusán változó állapotait különböztetjük meg és definiálunk formai jegyeket.

Dinamikus látszólagos osztályok		
A kijelölő jele	Példa	Magyarázat
:link	a:link ul a:link	ha a linken lévő helyet még nem látogattuk meg , beállítja a link formai megjelenését (alaphelyzetben: kék, aláhúzott)
:visited	a:visited #nav a:visited	ha a linken lévő helyet már meglátogattuk , beállítja a link formai megjelenését (alaphelyzetbe: lila, aláhúzott)
:active	a:active #container:active	ha a link éppen aktív (azaz éppen lenyomva van tartva rajta az egér gombja), beállítja a link formai megjelenését (alap: piros)
:hover	p:hover div.fontos:hover	annak az elemnek a formai megjelenítéséért felel, amelyik felett az egérkurzor éppen áll (pl. rámutatunk egy bekezdésre és annak hatására egy addig elrejtett szövegrészlet megjeleníthető)
Ha ugyanahhoz az <a> elemhez több dinamikus álosztályt is hozzá szeretnénk rendelni, akkor azok sorredje kötött: a:link a:visited a:hover a:active.		
:focus	input:focus input[type=text]:focus	ha fókuszba kerül az adott elem, akkor felveszi a megadott formátumot (pl. űrlapok adatbeviteli mezőjénél az éppen töltött mező színe eltérő lehet a többi adatbeviteli mező színétől)
:target	a.gomb:target	annak a linknek a formai megjelenítéséért felel, amelyik az aktív link célpontja, azaz ahová a link mutat (pl. a hivatkozásoknál a kattintással elért célpont formai jellemzőit tudjuk meghatározni, hasznos lehet akkor, ha a képernyő egy előre megadott területén a felhasználó választásától függően más-más tartalmat szeretnénk megjeleníteni, megmutatni)

Dinamikus látszólagos osztályok		
A kijelölő jele	Példa	Magyarázat
:enabled	input[type=text]:enabled { background-color: green; }	az engedélyezett elemek (többnyire űrlapelemnek) formai megjelenítését szabályozza (pl. a beviteli űrlapelem esetén ha a mező enabled állapotú, akkor zöld háttérrel jelenítjük meg azt)
:disabled	input[type=text]:disabled { background-color: red; }	a letiltott, azaz nem engedélyezett elemek (többnyire űrlapelemnek) formai megjelenítéséért felel (pl. a beviteli űrlapelem esetén disabled állapotú a mező, akkor piros háttérrel jelenítjük meg a beviteli mezőt, amellyel jelezzük, hogy nem írható)
:checked	input:checked + label	az éppen kijelölt rádiógombok vagy jelölőnégyzetek formai megjelenítéséért felel



A linkek színét mindig úgy kell beállítani, hogy meg lehessen különböztetni, hogy mely linkek voltak már meglátogatva és melyek nem, mégpedig úgy, hogy a már látogatott linkek színe legyen kevésbé élénk.

Kapcsolódó akadálymentességi elvek: ügyelni kell a linkek (különböző állapot)színeinek megadásánál arra, hogy eléggé kontrasztosak legyenek ahhoz, hogy a gyengén látó felhasználók is el tudják olvasni. A Firefox böngészőjébe érdemes telepíteni a WCAG Contrast Checker bővítményt, amellyel meg tudunk győződni arról, hogy a kontrasztarány megfelelő-e az oldalon. Szintén akadálymentességi elv, hogy ne csak színnel különböztessük meg az egyes elemeket a színtévesztő felhasználók igényeit kielégítendő. Ezért érdemes a linkeket nem csak színnel, hanem aláhúzással is megkülönböztetni.

B) A strukturális pszeudo-osztályok és elemeik

A strukturális pszeudo-osztályok közé azon megvalósításokat sorolhatjuk, amelyek a HTML-elemek egymáshoz viszonyított struktúrája alapján, bizonyos kitüntetett viszonyokhoz rendel stílusjegyeket.

Strukturális látszólagos osztályok		
A kijelölő jele	Példa	Magyarázat
:root	:root	a HTML-dokumentum azon eleme, amely nem rendelkezik szülővel, vagyis maga a html elem ¹² (akkor érdemes alkalmazni, ha a body elem nem fedi le teljesen a html elemet)
:empty	div:empty	az üres (tartalom nélküli), azaz a gyermektelen elemek formázására szolgál (pl. ha már egyetlen szóközt is tartalmaz egy div elem, akkor az nem üres)
:nth-child(x)	body :nth-child(4) li:nth-child(3n) p:nth-child(even)	x. testvér: a szülő típusától függetlenül kiválasztja a megadott sorszámú testvért <ul style="list-style-type: none"> ha szóköz van előtte, akkor a szelektor elején lévő elembe található gyermekek közül választja a kért számút ha nincs szóköz előtte, akkor a megadott testvérelemek közül választja a megadott számú elemet a paraméter általános alakja: $an+b$ (a és b egész szám) használható paraméter: odd (páratlan), even (páros)
:first-child	div:first-child	első gyermek: azon elem kiválasztása, amely az összes testvér közül az első (szülőelemtől függetlenül), értéke pontosan megegyezik a :nth-child(1) értékével
:nth-last-child(x)	p:nth-last-child(3)	hátulról az x. gyermek: a szülő típusától függetlenül hátulról visszafelé számolva az x. gyermek
:last-child	div:last-child	utolsó gyermek: azon elem kiválasztása, amely az összes testvér közül a legutolsó (szülőelemtől függetlenül), értéke azonos a :nth-last-child(1) értékével

¹² A :root azért került bevezetésre, mert a stíluslapokat az XML és az SVG állományok formázására is használhatjuk, és ott már mást jelent a gyökér elem fogalma, így e látszólagos kijelölővel nemcsak a <html> elem, hanem ezen fájlok formázása is megadható.

Strukturális látszólagos osztályok		
A kijelölő jele	Példa	Magyarázat
:only-child	p:only-child	egyetlen gyermek: annak az elemnek a kiválasztása, amelyik nem rendelkezik testvérral
:nth-of-type(x)	div:nth-of-type(6)	x. gyermek típuskijelölő: kiválasztja egy szülőelem x. gyermekelemét, ha a gyermek megadott típusú; a számlálást mindig az első adott típusú elemtől kezdi (példa magyarázata: a div testvérek közül a 6. div elem)
:first-of-type	p:first-of-type	első gyermek típuskijelölő: kiválasztja egy szülőelem első gyermekelemét, ha a gyermek a megadott típusú (példa magyarázata: a p testvérelemek közül az első p testvér)
:nth-last-of-type(x)	div:nth-last-of-type(5)	hátról az x. gyermek típuskijelölő: kiválasztja egy szülőelem x. gyermekelemét hátról számolva, ha a gyermek megadott típusú; (példa: a div testvérek közül az 5. div testvér hátról)
:last-of-type	h2:last-of-type	utolsó gyermek típuskijelölő: kiválasztja egy szülőelem utolsó gyermekelemét, ha a gyermek a megadott típusú (példa magyarázata: a h2 testvérek közül az utolsó h2)
:only-of-type	p:only-of-type	egyetlen gyermek típuskijelölő: annak az elemnek a kiválasztása, amelyből nincs hasonló típusú a testvérei között (példa magyarázata: olyan p elem, amelynek nincs p testvére, de lehet más típusú testvérelem, pl. több div elem)

C) Egyéb pszeudo-osztályok

Egyéb látszólagos osztályok		
A kijelölő jele	Példa	Magyarázat
:in-range	input:in-range { background-color: green; color: white; }	ha a mező aktuális értéke a megadott értéktartományba esik , felveszi a formátumjellemzőket (csak input űrlapelemek esetén használható a min és max tulajdonságokkal együtt)
:out-of-range	input:out-of-range { background-color: red; color: white; }	ha a mező aktuális értéke NEM a megadott értéktartományba esik , felveszi a formátumjellemzőket (csak input űrlapelemek esetén használható a min és max értékeivel)
:valid	input[type=date]:valid { border: 3px solid green; }	ha a mező aktuális értéke vagy típusa megfelel a mezőre beállított feltételeknek vagy típusleírásnak , felveszi a formátumjellemzőket (csak űrlapelemek esetén használható, pl. a min és max tulajdonságokkal, az e-mail, a number, a date ... típusok esetén)
:invalid	input[type="e-mail"]:invalid { border: 3px solid red; }	ha a mező aktuális értéke vagy típusa NEM felel meg a mezőre beállított feltételeknek vagy típusleírásnak , felveszi a formátumjellemzőket (csak űrlapelemek esetén használható a min és max tulajdonságokkal, az e-mail, a number, a date ... típusok esetén)
:lang	div:lang(en) { font-style: italic; }	ha a megjelölt elem nyelvi tulajdonsága felveszi a meghatározott értéket , felveszi a formátumjellemzőket (bármilyen elemre alkalmazható)
:not	*:not(div) { font-family: Calibri; }	a megjelölt szelektor kivételével a többi elemet formázza (a not mögött adjuk meg a kivételt tartalmazó elemnevet)
:optional	input:optional { color: green; }	ha az űrlapmező értékét nem kötelező megadni , felveszi a formátumjellemzőket (csak az input, select és textarea űrlapmezők esetén alkalmazható)
:required	input:optional { background-color: yellow; }	ha az űrlapmező értékét mindig kötelező megadni , felveszi a formátumjellemzőket (csak input, select és textarea esetén)

Egyéb látszólagos osztályok		
A kijelölő jele	Példa	Magyarázat
:read-only	input[type=date]:read-only { background-color: silver; }	ha az űrlapmező értéke csak olvasható , azaz a tartalma nem változtatható (csak input elemek esetén alkalmazható)
:read-write	input[type=text]:read-write { background-color: green; color: white; }	ha az űrlapmező értéke írható és olvasható is , azaz a mező tartalma szabadon változtatható (alaphelyzetben minden űrlapmező szabadon módosítható)

Pseudo-elemek (látszólagos elemek)

A HTML-dokumentum struktúrájában más módon le nem írható objektumok elérésére szolgálnak a **pszeudo-elemek** (*psseudo elements*), amelyeket a **szelektor::pszeudo-osztály { stílusdefiniáció }** formában¹³ alkalmazhatunk. Gyakran szükség lehet ugyanis arra, hogy a HTML-elemeknek csak egy-egy részletét formázzuk meg, pl. egy bekezdés első sorát, egy blokk első betűjét, illetve hogy az elem tartalma előtt / mögött automatikusan megjelenítsünk egy generált tartalmat (pl. képet).

Pszeudo-elemek használata a kijelölésben		
A kijelölő jele	Példa	Magyarázat
::first-letter	p::first-letter { font-size: 200%; }	az elem első betűjének formázását biztosítja (ennek felhasználásával oldható meg az iniciálék készítése is)
::first-line	p:first-line { font-variant: small-caps; font-size: 150%; }	az elem első sorának formázását biztosítja (csak blokk-szintű elemeknél alkalmazható) ¹⁴
::selection	::selection { color: red; background-color: yellow; }	a felhasználó által kijelölt részt megformázza a megadott tulajdonságokkal (alkalmazható tulajdonságok: color, background, cursor, outline)
::after	h1::after { content: url(smiley.gif); }	tartalmat helyez el a meghatározott elem után
::before	h1::before { content: url(smiley.gif); }	tartalmat helyez el a meghatározott elem elé

IV. Betűk és szövegek formázása

1. Szövegek tagolása

Igen hasznos és elterjedt tagolási módszer a HTML-dokumentumokban a **vízszintes vonal alkalmazása**. A vonal beszúrásához a `<hr>` (horizontal line) utasítást használjuk.

Ha a `<hr>` utasítást folyamatos szövegen belül alkalmazzuk, úgy az előtte és utána is sortörést eredményez, tehát a vonal csak önálló sorban állhat.

A vonal jellemzői	
tulajdonság neve	leírás
width	a vonal szélessége (hosszúsága)
height	a vonal magassága (vastagsága)
background-color	vastagabb vonal esetén a kitöltőszín értéke
border	a vonal stílusa
position, left, right	a vonal oldalon való elhelyezése

¹³ A CSS3-ban vezették be azt a különbségtételt, hogy a pszeudo-osztályok előtt egy darab kettőspont, a pszeudo-elemek előtt pedig dupla kettőspontot kell írni. Vannak azonban olyan látszólagos elemek (ilyen pl. a first-letter vagy a first-line), amelyeket a CSS korábbi változatában vezették be, ezért azokat egyszeres kettősponttal is lehet írni és alkalmazni. (Ez egyedül a ::selection elemre nem vonatkozik!)

¹⁴ A ::first-line és a ::first-letter pszeudo-elemek esetén az alkalmazható stílusjegyekről érdemes a w3schools oldaláról tájékozódni.

Példa vízszintes vonal formázásához:

```
hr { width: 50%; height: 10px; color: #ff0;
    border-bottom: 10px dashed #f00;
    position: absolute; left: 25%; }
```

Mivel nincs külön HTML-tag függőleges vonalhoz, ezért annak megrajzolásához képet alkalmazunk vagy <div> objektumot helyezünk el és azt hosszú keskeny elemként formázzuk meg:

HTML-kód: <div class="fv"></div>

CSS-kód: .fv { height: 200px; width: 0px;
 background: transparent;
 border-left: 2px dotted #ff0;
 position: fixed; left: 20%; }

A szöveg tördelésével kapcsolatos beállításokat a white-space tulajdonság értékeinek megadásával (azaz a CSS-kódban a white-space: ... sor megadásával) **határozhatjuk meg:**

- **normal** (alapértelmezés): a többszörös üres helyeket (szóközöket, Entereket) egyetlen szóközzé alakítja át, illetve szükség esetén a szóközöknél új sort kezd (= automatikus tördelés)
- **pre**: megőrzi a többszörös üres helyeket (figyelembe veszi az előformázást)
- **nowrap**: a
 hiányában nem lesz sortörés a hosszú sorok végén, tehát ez az opció a szöveg automatikus tördelését tiltja meg (az automatikus tördelés letiltható a <nowrap> és a </nowrap> utasításpár használatával is, az e két HTML-tag közé írt szöveg egyetlen hosszú sorként fog megjelenni - ebben az esetben a <wbr> utasítással törhető meg a hosszú sor)
- **pre-wrap**: megőrzi a többszörös üres helyeket, de engedélyezi az automatikus sortöréseket
- **pre-line**: megőrzi a sortörés, egyébként a *normal* beállításként működik
- **inherit**: örökli a szülőtől annak beállításait

2. Karakterjellemzők

Minden modern grafikus operációs rendszer több fontkészlettel rendelkezik, a felhasználói programok telepítésével pedig további fontkészletek válnak elérhetővé. Az elektronikus dokumentumokban használt betűtípusok helyes megválasztásával az esztétikus megjelenésen túl, áttekinthetővé tehetjük weboldalunkat, kiemelve a legfontosabb információkat. A betűtípusok cél nélküli keverése azonban megzavarhatja az olvasót.

A számítógépes környezetben használt **font** szó a francia fonte szóból ered (jelentése 'olvadt'), amely a hajdani nyomdászok ólomból öntött betűsorozatára utalt. **A betűtípus (= font) olyan karakterek (betűk, számok, írásjelek) együttese, amelyek bizonyos közös jellemzőkkel rendelkeznek.** Forrása lehet helyi vagy letölthető (web fonts). Ma a font egy adott betűtípust jelent, amelynek karakterei egy közös fájlban vannak tárolva.

A HTML oldalak esetén a legnagyobb problémát az jelenti, hogy a webfejlesztő soha nem tudhatja, hogy a látogatói gépén mely fontkészletek hozzáférhetők. **A böngészőfüggetlen webfejlesztés alapvető célja, hogy bárki, bármilyen eszközön, bármelyik operációs rendszer és böngésző használatával tekint meg weboldalainkat, ugyanazzal az esztétikus megjelenéssel találkozunk.** Ha az szeretnénk, hogy a weboldalon használt betűtípusok minden böngészőben és minden eszközön egységesen jelenjenek meg, akkor használjunk web-biztos fontkészleteket, saját fontkészletet vagy az interneten hozzáférhető böngészőfüggetlen fontkészletet (ezek között találhatunk ingyeneseket és licenctíjasokat is). A fontkészlet kiválasztásánál ügyelni kell arra is, hogy tartalmazzák a speciális magyar karaktereket (í,ő,ű...), különben ezek helyett a felhasználó csak egy téglalapot fog látni.

A böngészők mindig megpróbálják a stíluslapon megadott betűtípusokat használni a megjelenítéshez, de nem mindig sikerül. Előfordul, hogy a felhasználó számítógépén betűtípus nem elérhető. Ekkor a böngésző egy hasonlót igyekszik találni. Ebben segítenek az általános fontcsaládok, amelyek csupán a felhasználandó betűtípus alapvető jellemzőit határozzák meg.

A betű-családneveket két fő típusba sorolhatjuk:

- **általános vagy gyűjtő családnevek** (generic family keywords):
 - **serif**: betűlábát lezáró talpas (pl. Times New Roman, HTimes), könnyebb olvasni a nyomtatott szövegekben, ezért elsősorban a nyomtatott anyagokban használják
 - **sans-serif**: talp nélküli (pl. Arial, Calibri), monitoros megjelenítés során célszerű használni
 - **monospace**: valamennyi karakterjelnek azonos, rögzített szélessége van (pl. Courier)
 - **cursive**: kézírást utánozó, dőlt stílusú (pl. Corsiva), karakterei egymással összekapcsolódnak
 - **fantasy**: játékos, dekoratív betűtípusokat tartalmaz (pl. Comic Sans, Curlz MT)
- **specifikus családnevek** (font family name): nagyon sok ilyen van és nem mindegyiket ismerik fel a felhasználó gépén a böngészők – általában prioritási sorrendben felsorolva, a többi CSS-tulajdonságtól eltérően vesszővel elválasztva adandók meg, utolsóként egy általános család-névvel lezárva a sort¹⁵

Jó eséllyel minden böngészőben megfelelő eredményhez vezetnek, ha az alábbiakat alkalmazzuk:

csoportnév	példa
serif fontok (talpas betűk)	"Times New Roman", Times, serif Georgia, serif 'Palatino Linotype', 'Book Antiqua', Palatino, serif 'Bookman Old Style', serif, Times, serif
sans-serif fontok (talp nélküli betűk)	Arial, Helvetica, sans-serif Tahoma, Geneva, sans-serif Verdana, Geneva, sans-serif "Arial Black", Gadget, sans-serif Impact, Charcoal, sans-serif
monospace fontok (azonos betűszélesség)	"Courier New", Courier, monospace "Lucida Console", Monaco, monospace

A betűtípus forrása lehet helyi (ahol fut a böngésző) vagy letölthető (web fonts). Erre az utóbbira kétféle megvalósítási lehetőség áll a rendelkezésünkre:

- **a letöltendő betűtípust válasszuk ki a <http://google.com/webfonts> oldalról** (ezt fogjuk letöltendő stíluslapként használni), majd **a <head>-ben elhelyezzük a betűtípusra mutató linket, mintha külső stíluslapot kapcsolnánk a HTML-oldalhoz** (ez a módszer minden böngészővel működik, szabad hozzáférésű megoldás, amely nem igényel új CSS-ismeretet):

```
<LINK rel="stylesheet"
      href="http://fonts.googleapis.com/css?family=Pacifico">
```

- **a CSS-kódban a @font-face szabállyal definiálható a letöltendő betűkészlet**, amelynek tulajdonságaként a betűcsalád és annak forrása (a teljes URL) adandó meg (a letölthető betűkészletekkel szembeni fontos követelmények: kis fájl méret, másolhatatlanság → korlátozottan használhatók a szerzői jogok miatt):

```
@font-face {
  font-family: ... ;
  src: url(http:// ... .com/fonts/fájl); }
```

A CSS-ben az egyes betűképek (font face) definiálása a betűtípuscsaládból (font-family) és egyéb betűtulajdonságok felsorolásából állnak.

Betűtulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
font-family	betűtípus (név)	body { font-family: Arial, "Arial Black", sans-serif; } a nevek kisbetű/nagybetű érzékenyek; a betűcsaládok neveit csak akkor szabad idézőjelbe tenni, ha azok több szóból állnak

¹⁵ A böngésző addig halad a specifikus családnevek listájában, amíg nem talál egy számára rendelkezésre álló betűtípust, a gyűjtő család név pedig általános tartalékként szolgál arra az esetre, ha a felsorolt specifikus családokból egy sem lenne megjeleníthető.

Betűtulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
font-size	betűméret	<pre>p.bev { font-size: medium; }</pre> <p>megadhatjuk abszolút módon (<i>absolute-size</i>): px, % és em mértékegységekkel¹⁶ vagy megnevezéssel (xx-small, x-small, small, medium, large, x-large, xx-large), illetve relatív módon (<i>relative-size</i>; az öröklött betűmérethez képest növeli vagy csökkenti a méretet): larger, smaller vagy használható még a vw (viewport width) mértékegység is¹⁷</p>
font-style	betűdöntés	<pre>div { font-style: italic; }</pre> <p>a betű dőltsege lehet italic (dőlt), oblique (ferde, hajló), normal (normál)</p>
font-variant	kiskapitálissá alakítás	<pre>p.kicsi { font-variant: small-caps; }</pre> <p>a betűváltozat lehet normal (alapérték) vagy small-caps (kiskapitális = minden betű nyomtatott betűs, de a korábbi kisbetűk kisebb méretűek lesznek, pl. Magyar Köztársaság → MAGYAR KÖZTÁRSASÁG)</p>
font-weight	betűvastagság	<pre>.kiemelt { font-weight: bolder; } .nemfontos { font-weight: 100; }</pre> <p>a karakterjel súlyát vagy a vonalvastagságot specifikálja: értéke megadható névvel vagy számmal</p> <ul style="list-style-type: none"> név meghatározásával: normal (ez felel meg a 400 értéknek), bold (700), bolder (az öröklött értéknél vastagabb), lighter (az öröklött értéknél vékonyabb) számérték megadásával: 100 és 900 közötti szám: 100 (thin), 200 (extra light), 300 (light), 400 (normal), 500 (medium), 600 (semi bold / demi bold), 700 (bold), 800 (extra bold / ultra bold) vagy 900 (black / heavy)¹⁸
font-stretch	betűkiterjedés	<pre>div.szeles { font-stretch: expanded; }</pre> <p>a betűszélességet a normal (alapértelmezett), a condensed (összenyomott) és az expanded (megnyújtott) értékekkel szokás a leggyakrabban meghatározni, de vannak semi-, ultra- és extra- előtagú változataik is</p>
font	összevont tulajdonság (karakterjellemzők)	<p>a font tulajdonsággal összevontan is megadhatók az egyes betűtulajdonságok úgy, hogy a fenti stílusok értékeit egymás után felsoroljuk és szóközzel választjuk el az alábbi sorrend szerint: betűstílus (font-style), betű-változat (font-variant), betűvastagság (font-weight), betűméret (font-size) és betűtípus (font-family):</p> <pre>p { font: italic normal bolder 12pt Arial; }</pre> <p>(a minimálisan elvárt két érték a betűméret és a betűtípus megadása)</p>
color	betűszín	a szöveg betűszínét a színek megadásánál megismert módon többféleképpen is meghatározhatjuk

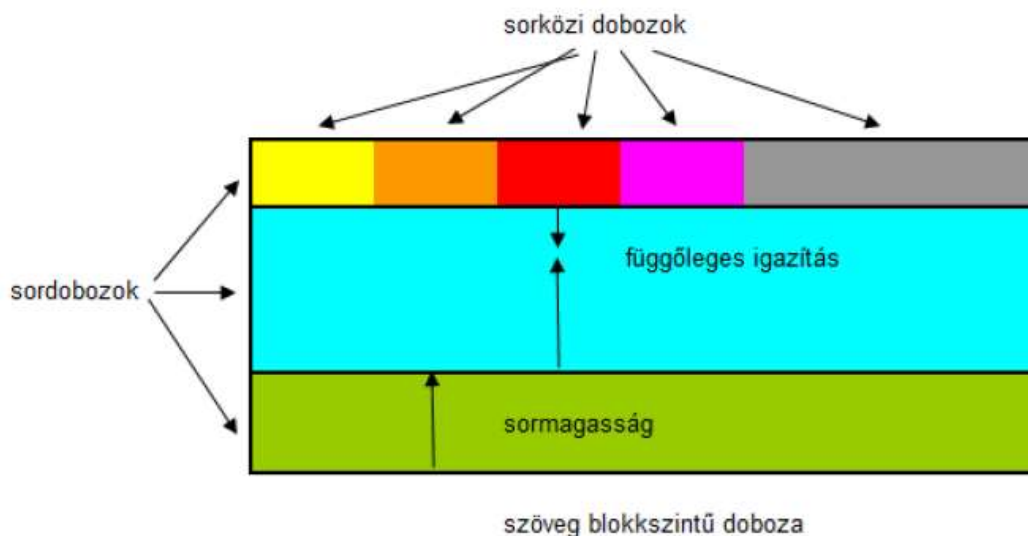
¹⁶ A W3C a % és az em mértékegységgel történő méretmegadást ajánlja, ahol az 1em méret megegyezik az aktuális betűmérettel. A böngészőkben az alapértelmezett szövegméret a 16 képpont, így 1em alapértelmezett méretet 16px. Ennek alapján a pixelméretből az em-beli méretet az alábbi képlettel lehet kiszámolni: pixelméret/16.

¹⁷ Ezzel a mértékegységgel jól biztosítható a reszponzivitás, mert a méreteket nem egy elem fix értékéhez, hanem annak változó nagyságához kapcsoljuk. Így ha a nézetablak pl. 50 cm széles, akkor 1vw = 50 cm * 1% = 0,5 cm.

¹⁸ Gyakran csak néhány vastagság szerepel egy-egy betűcsaládban, ezért a böngésző egy hozzá közeli vastagságot választ.

3. Szövegjellemzők

A nagyobb egységek (pl. bekezdések) sor-dobozokból (pl. egy szöveg sora) és a soron belüli dobozokból (pl. karakterek, szavak, képek egy soron belül) épülnek fel:



Ebben a részben csak a vízszintes sorokból álló, egyhasábos szövegek formázásáról lesz szó.

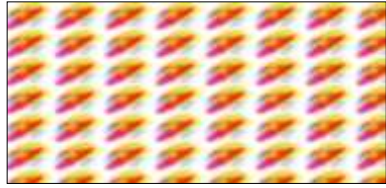
Szövegtulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
text-align	szöveg vízszintes igazítása	<code>p { text-align: center; }</code> a vízszintes szövegigazítás (rendezés) lehet left (balra), right (jobbra), center (középre) és justify (sorkizárt)
text-justify	sorkizárt szövegbeli szóközök kezelése	<code>div { text-align: justify; text-justify: inter-word; }</code> sorkizárt típusú szövegigazítás esetén megadható, hogy milyen módon kezelje a böngésző szavak közötti szóközöket – lehetséges értékei: auto (a böngésző algoritmus szerinti elrendezés), inter-word (csökkenti vagy növeli a szavak közötti teret), inter-character (csökkenti vagy növeli a betűk közötti teret), none (letiltja a változtatás), initial (az alapértelmezett érték alkalmazása), inherit (a szülőtől örökölt érték alkalmazása)
vertical-align	függőleges igazítása	<code>span { vertical-align: middle; }</code> a soron belüli (azaz a sorközi dobozoknak a sordobozon belüli) függőleges irányú elhelyezkedését definiálja, amely lehet baseline vagy auto (alaponálhoz, ez az alapértelmezett érték), top (tetejéhez), bottom (aljához), middle (középre), text-top (szöveg tetején), text-bottom (szöveg alján), super (fölé) vagy sub (alá) – pl. azonos sorban elhelyezkedő szöveg és kép, beviteli mezők egymáshoz történő igazításakor alkalmazzuk
text-decoration	alá- és áthúzások	<code>p { text-decoration: line-through; }</code> szövegdíszítés megadása, amelyben a lehetséges értékek: none (nincs), underline (aláhúzott), overline (föléhúzott), line-through (áthúzott), blink (villogó), inherit (örökölt) – egyszerre több megadhatós
text-transform	szöveg kis- és nagybetűs átalakítása	<code>.kisbetus { text-transform: lowercase; }</code> szövegátalakítás meghatározása, amelyben a lehetséges értékek: none (nincs), inherit (örökölt), uppercase (nagybetűs), lowercase (kisbetűs), capitalize (szavankénti nagy kezdőbetűk) – ez utóbbi esetben böngésző-függő, hogy mit tekint egy szóznak
line-height	sorköz	<code>.masfeles { line-height: 1.5; }</code> a bekezdésben lévő sorok közötti távolságot szabályozza (csak + érték adható meg), lehet: normal, %, számérték vagy abszolút mértékegység

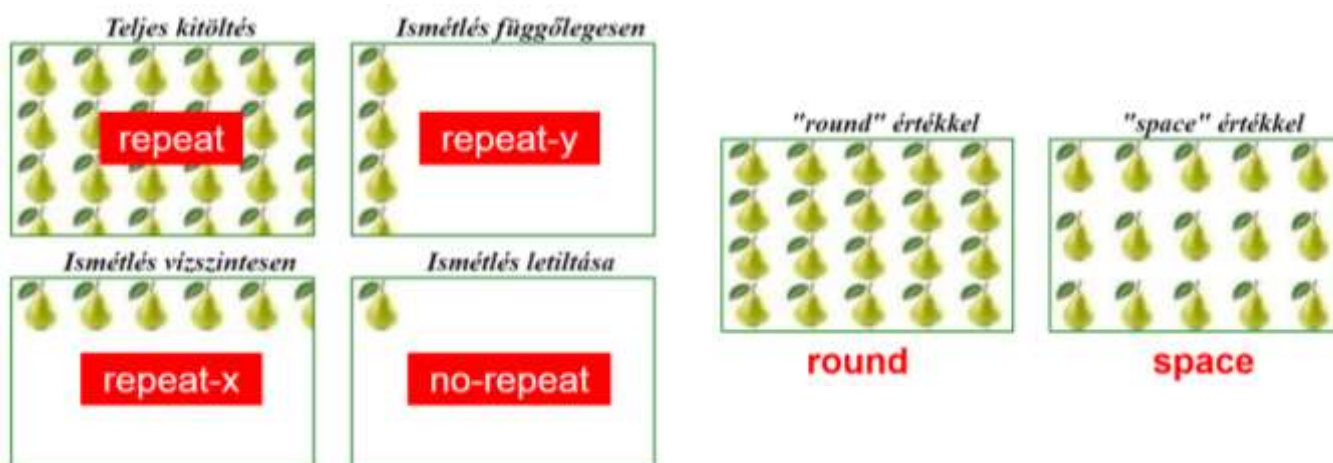
Szövegtulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
text-indent	első sor pozíciója	<pre>.elsosor { text-indent: 20px; } .fuggosor { text-indent: -10px; }</pre> <p>az első sor pozíciójának meghatározása, amelyben a pozitív érték az első sor behúzását, a negatív érték pedig függő behúzás létrehozását eredményezi</p>
text-shadow	szöveg árnyékolása	<pre>#cimarnyek { text-shadow: 15px 10px 2px red; }</pre> <p>egy vagy több, különböző színű, nagyságú, irányú és életlenítési / elhalványulási árnyék rendelhető egy szöveghez – három hosszúsági érték (vízszintes és függőleges kiterjedés, életlenítés) és egy színérték adható meg; ha több, különböző színű, nagyságú, irányú és életlenítési / elhalványulási árnyéket is meg akarunk adni ugyanahhoz a szöveghez, akkor a négyes értékeket vesszővel elválasztva kell felsorolni:</p> <pre>#arnyekok { text-shadow: 5px 1px 2px red, 10px -4px 2px blue, 20px -8px 0px green; }</pre>
word-spacing	szóköz	<pre>.novelt { word-spacing: 30px; } .csokkentett { word-spacing: -10px; }</pre> <p>a szavak közötti elválasztó karakterek (azaz a szóköz) alapértelmezett képesti változását (növelését vagy csökkentését) specifikálja</p>
letter-spacing	betűköz	<pre>.ritkitott { letter-spacing: 10px; } .suritett { letter-spacing: -2px; }</pre> <p>a betűk közötti távolságnak az alapértelmezett betűközhez képesti változását határozza meg: negatív érték esetén a karakterek közelebb (sűrítve), pozitív értéknél távolabb (ritkítva) kerülnek egymáshoz képest</p>
white-space	szóközök kezelése	<p>tagoló karakterek figyelembevételének szabályozása – lehetséges értékei:</p> <ul style="list-style-type: none"> normal: a többszörös üres helyeket egyetlen szóközzé alakítja, sortörés kifejezetten az ilyen parancsot kiváltó (pl. blokkszintű) objektum hiányában akkor lesz, amikor a tartalom már látható módon nem fér el az adott sorban nowrap: a böngésző az ablak szélén sem fogja automatikusan megtörni a sorokat, viszont szükség esetén megjelenít egy görgetősávot (ekkor a sortörést bárhol kiválthatunk
 taggal) pre: a tagoló karakterek mind figyelembe lesznek véve pre-wrap: vegye figyelembe a tagoló karaktereket, de automatikus sortörést is alkalmazzon ott, ahol szükséges (ha nem fér el a sorban) pre-line: megőrzi a sortörést, egyébként alapértelmezés szerint jár el <p><i>Ezek a formázások nemcsak egy oldalra, de azon belül bármilyen blokkobjektumra is így működnek. A böngészők ezeket a lehetőségeket (jellemzően a pre-wrap, ill. a pre-line beállítást) nem mind ismerik, ezért alkalmazásuk előtt érdemes kipróbálni őket a célzott böngészőben.</i></p>
word-wrap	szavak tördelése	<p>a szöveg tetszőleges helyén megtörheti a szót a böngésző, ha nincsen elég hely a folytatáshoz (ellentétben a HTML-ben alkalmazott <wbr> címkével, amely csak előre meghatározott szótörési helyeket tud kódolni), lehetséges értékei: normal (alapértelmezett, csak ott töri meg a szót, ahol egyébként is lehetséges) vagy break-word (bárhol megtöri, ahol a rendelkezésre álló hely miatt arra szüksége van a böngészőnek</p>
hyphens	automatikus szó-elválasztás	<p>automatikus szóelválasztás módjának beállítása; lehetséges értékei: manual (alapértelmezett – a böngésző alapbeállítása szerinti tördelés), none (nincs) vagy auto (ekkor fontos, hogy a nyelv be legyen állítva a <html lang=...> sorral a HTML-kódban, mert az itt meghatározott nyelv szabályai szerint fogja elvégezni az automatikus tördelést</p>

V. Háttértulajdonságok

A weblap valamennyi elemének (karakter, szó, címsor, bekezdés, lista, táblázat, kép stb.) van háttérreje, amely alapértelmezetten

- vagy teljesen átlátszó,
- vagy a háttérszín tulajdonság értékének megfelelően kitöltött színű,
- és/vagy a háttérkép(ek) tulajdonságértéke(i)ként meghatározott kép(ek) alkotja(k).

A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background-color	háttérszín	<pre>body { background-color: #000A00; } body { background-color: yellow; }</pre> <p>a háttérszínt a teljes elemnek beállítja; a színmegadásoknál meghatározottak szerint adhatjuk meg</p>
background-image	háttérkép	<pre>body { background-image: url(tegla.jpg); }</pre> <p>mozaikszerűen ismétlődik a kép (ha nem fér ki az utolsónak ismételt, akkor mindkét irányban levágja a böngésző), mert a böngészők alapértelmezetten <i>a teljes kitöltendő háttérre a megadott kép méretének változtatlanul hagyása mellett, annak vízszintes és függőleges irányú ismétlésével, összefüggően töltik ki</i> (háttérkép és háttérszín együttes használata esetén mindig a háttérkép van felül és a háttérszín alul)</p> <p>Hogyan lehet egyidejűleg több háttérképet is alkalmazni? – A képfájlokat és a tulajdonságokat felsorolásszerűen, vesszővel és szóközzel elválasztva adjuk meg, ahol a felsorolás sorrendje egyben meghatározza a háttérképek egymásra rétegződését (azaz az első háttérkép van legfelül és minden további háttérkép a sorrendnek megfelelően egyre hátrébb kerül).</p> 
background-repeat	háttérkép ismétlődése	<pre>body { background-image: url(tegla.jpg); background-repeat: repeat-x; }</pre> <p>a háttérkép ismétlődése többféle is lehet - lehetséges értékei:</p> <ul style="list-style-type: none"> • repeat (mozaikszerű ismétlődés mindkét irányban; alapértelmezés) • no-repeat (nincs ismétlődés) • repeat-x (csak vízszintes irányban ismétlődik) • repeat-y (csak függőleges irányban ismétlődik) • round (vízszintes irányban egész számú ismétlés úgy, hogy a kép szélességi méretét megnyújtja) • space (egész számú ismétlést végez úgy, hogy a háttérképek között üres helyeket – távolságokat – hagy annak megfelelően, hogy a háttérkép szélességi értéke alapján hányszor fér ki az egymás mellé)



A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background-position-x	háttérkép vízszintes helyzete	<pre>body { background-image: url (teglá.jpg); background-repeat: no-repeat; background-position-x: 40%; }</pre> <p>a háttérkép vízszintes helyzete paraméter lehetséges értékei a szokásos hosszúsági mértékegységek mellett az egész értékű százalék, illetve a left (vízszintesen balra), center (vízszintesen középre) és a right (vízszintesen jobbra) lehetnek – függőlegesen ismétlődő háttérképhez</p>

Függőleges ismétlődés esetén

Vízszintesen balra



left

(alapértelmezés)

Vízszintesen középen



center

Vízszintesen jobbra



right

A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background-position-y	háttérkép függőleges helyzete	<pre>body { background-image: url (teglá.jpg); background-repeat: no-repeat; background-position-y: top; }</pre> <p>a háttérkép függőleges helyzete paraméter lehetséges értékei a szokásos hosszúsági mértékegységek mellett az egész értékű százalék, illetve a top (függőlegesen felülre), center (függőlegesen középre) és a bottom (függőlegesen alulra) lehetnek – vízszintesen ismétlődő háttérképhez</p>

Vízszintes ismétlődés esetén

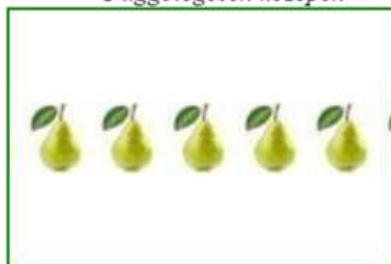
Függőlegesen felül



top

(alapértelmezés)

Függőlegesen középen



center

Függőlegesen lent



bottom

A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background-position	összevont tulajdonság (háttérkép-pozíciók)	<pre>body { background-image: url(tegla.jpg); background-repeat: no-repeat; background-position: 100% 50%; }</pre> <p>a háttérkép vízszintes és függőleges helyzetének (elhelyezkedésének) meghatározása, amelyben egy értékpárt adunk meg: az első a vízszintes, a második a függőleges elhelyezkedést határozza meg – akkor érdemes használni, ha nem ismétlődik a háttérkép</p> <p>alkalmazható beállítások / mértékegységek:</p> <ul style="list-style-type: none"> • elnevezéssel: left (balra), right (jobbra), top (felül), bottom (alul), center (középre) • %-értékkel: pl. 0% 0% a bal felső sarkot, az 50% 50% a szülőelem közepét fogja jelenteni • képpontban: a szülőelemhez képest a megfelelő irányokban a megadott értékkel van eltolva a háttérkép (pl. 50px 100px)



A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background-attachment	háttérkép gördíthetősége	<pre>body { background-image: url(tegla.jpg); background-repeat: repeat-x; background-attachment: fixed; }</pre> <p>a háttérkép kapcsolásának (gördíthetőségének) meghatározása megadja a háttérkép kapcsolatát a dokumentumon belüli szülő-objektummal, amelynek alapértelmezett értéke a scroll (a tartalom gördítésénél a háttérkép is gördüljön), illetve használható a fixed, ekkor a háttérkép a tartalom gördítése esetén sem mozdulhat el (vízjelként helyben marad)</p>
background-size	háttérkép mérete	<p>a háttérképként alkalmazott kép fizikai méreteitől eltérően, más méretekkel is felhasználhatjuk a képet – lehetőségek:</p> <ul style="list-style-type: none"> • kulcsszóval: auto, cover, contain • egyetlen érték megadásával (ez akkor a kép szélességi mérete lesz, a másik auto értéket fog felvenni) • két érték megadásával (az első a szélesség, a második a magasság) • többszörös szintaxissal (vesszővel felsorolt értéksorozattal) <p>lehetséges értékek:</p> <ul style="list-style-type: none"> • auto (alapértelmezett érték): a kép az eredeti fizikai mérete szerint • pixelérték: a megadott képpont szerinti méretekkel jelenik meg • %-érték: a kép mérete a kitöltendő rész nagyságához viszonyítva kerül meghatározásra (a háttérkép ekkor magasságában és szélességében is arányosan kerül megjelenítésre) • cover (lefedés): változatlan képarány mellett a háttérkép mérete úgy változik, hogy a kisebbik mérete teljesen lefedje a háttérként kijelölt területet (tehát a képet nyújtva vagy levág belőle) • contain (magában foglal): változatlan képarány mellett a háttérkép mérete úgy változik, hogy a teljes háttérkép beleférjen a háttérként kijelölt területbe (tehát a képet arányosan átméretezi úgy, hogy kimaradhatnak lefedetlen területek is)

Automatikus méret



Pixelben (200x200)



Pixelben (230 auto)



Százalékban (40%)



Százalékban (85%)



Lefedéssel (cover)



Befoglalással (contain)

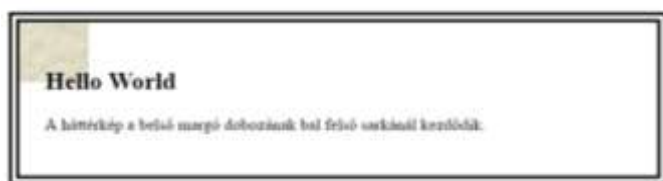


A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background-clip	háttérkép hatóköre	<p>alaphelyzetben a háttérkép és a háttérszín hatása (azaz hatóköre) az elem szegélye által határolt területéig terjed ki – ezen bármikor változtathatunk:</p> <ul style="list-style-type: none"> • border-box: a szegélyek külső élei által határolt teljes területig • padding-box: a belső margó és a tartalom együttes dobozáig • content-box: a tartalom dobozra vonatkozik a megadott érték



A háttérrel kapcsolatos tulajdonságok megadása		
CSS-tulajdonság	Funkció	Példa, magyarázat
background	az összes háttér-tulajdonság összevonása	<pre>body { background: yellow url(kep.jpg) repeat-x fixed; }</pre> <p>összesíti a különböző háttértulajdonságokat, vagyis a fenti stílusok értékei megadhatók egymás után felsorolva (szóközzel elválasztva) úgy, hogy a tulajdonságnév elmarad és csak az értékek vannak megadva egy sorrendben: háttérszín, háttérkép és annak ismétlődése, kapcsolása, helyzete</p>
background-origin	háttérkép referenciapontja	<pre>body { background-origin: padding-box; }</pre> <p>a háttérképek helyzetének megadásához a pozicionálás referenciapontjának és az ahhoz képest meghatározott helyzetnek az ismerete is szükséges, tehát a lehetséges referenciapont</p> <ul style="list-style-type: none"> • padding-box (alapértelmezés: a belső margódoboz bal felső sarka) • content-box: a tartalomdoboz bal felső sarka • border-box: a szegélydoboz bal felső sarka

background-origin: padding-box (default):

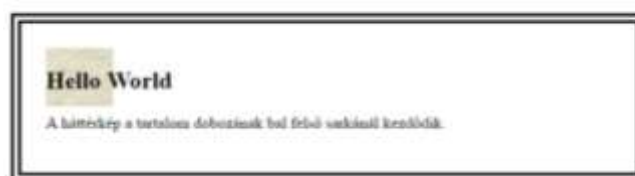


background-origin: border-box:



Ha több, egymásra vagy egymás mellé helyezett képből épül fel a háttér, felsorolás-szerűen, vesszővel és szóközzel elválasztva kell a képfájlokat és azok egyes tulajdonságait megadni. A képek egymásra rétegződésének sorrendjét a felsorolás sorrendje szabja meg – az első fájlnev képe van legfelül (azaz a felhasználóhoz legközelebb) és minden további kép a sorrendnek megfelelően egyre hátrább.

background-origin: content-box:



VI. A dobozmodell és tulajdonságai

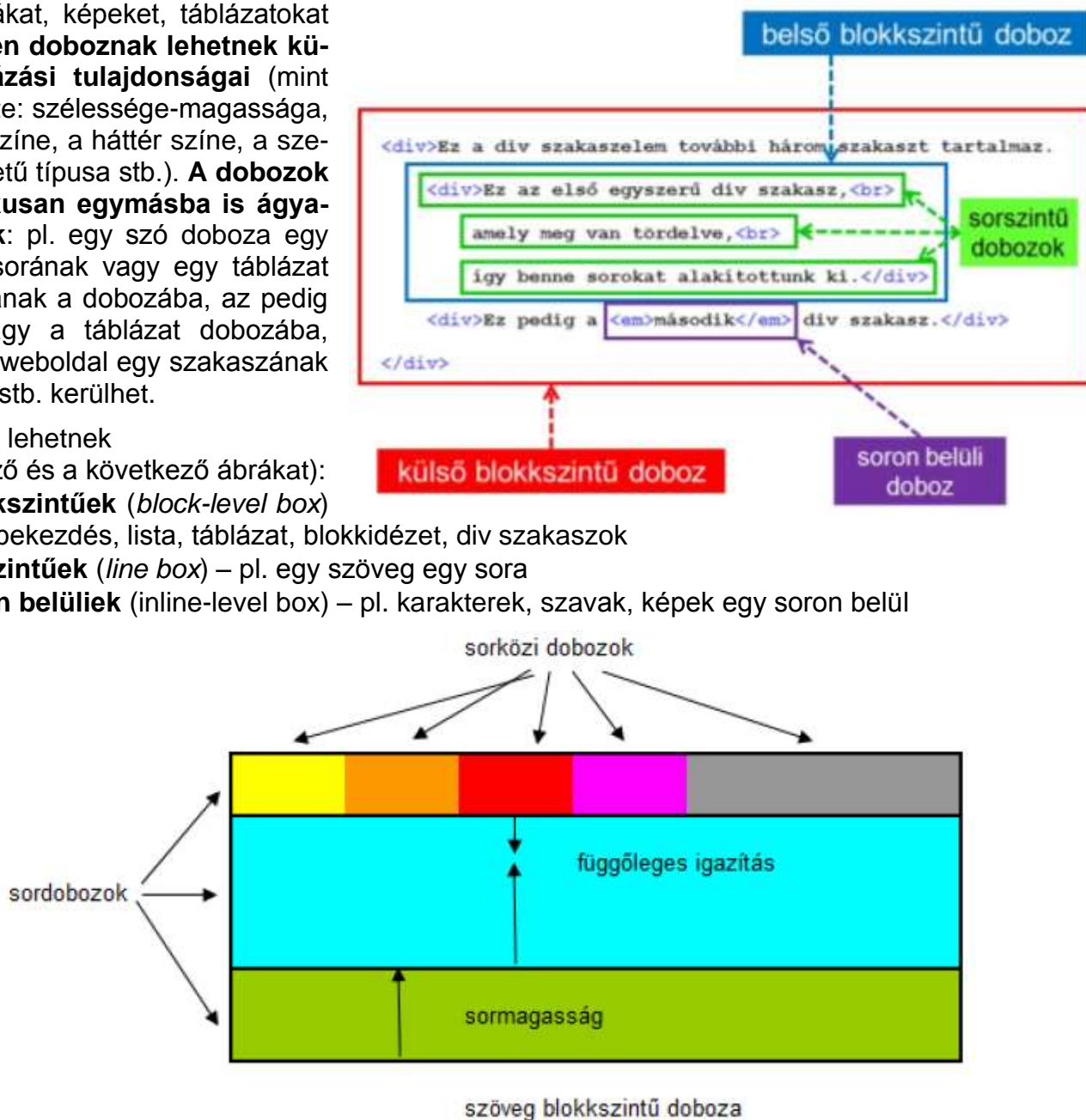
A dobozmodell alapvető jellemzői

A CSS stíluslapnyelv egy HTML-dokumentum megjelenítésekor az egyes elemekhez láthatatlan **dobozokat rendel**, amelyek magukban foglalhatnak karaktereket, szavakat, sorokat, bekezdéseket, listákat, képeket, táblázatokat stb. **Minden doboznak lehetnek külön formázási tulajdonságai** (mint pl. a mérete: szélessége-magassága, az előtér színe, a háttér színe, a szegélye, a betű típusa stb.). **A dobozok hierarchikusan egymásba is ágyazódhatnak**: pl. egy szó doboza egy lista egy sorának vagy egy táblázat egy cellájának a dobozába, az pedig a lista vagy a táblázat dobozába, majd az a weboldal egy szakaszának dobozába stb. kerülhet.

A dobozok lehetnek

(ld. az előző és a következő ábrákat):

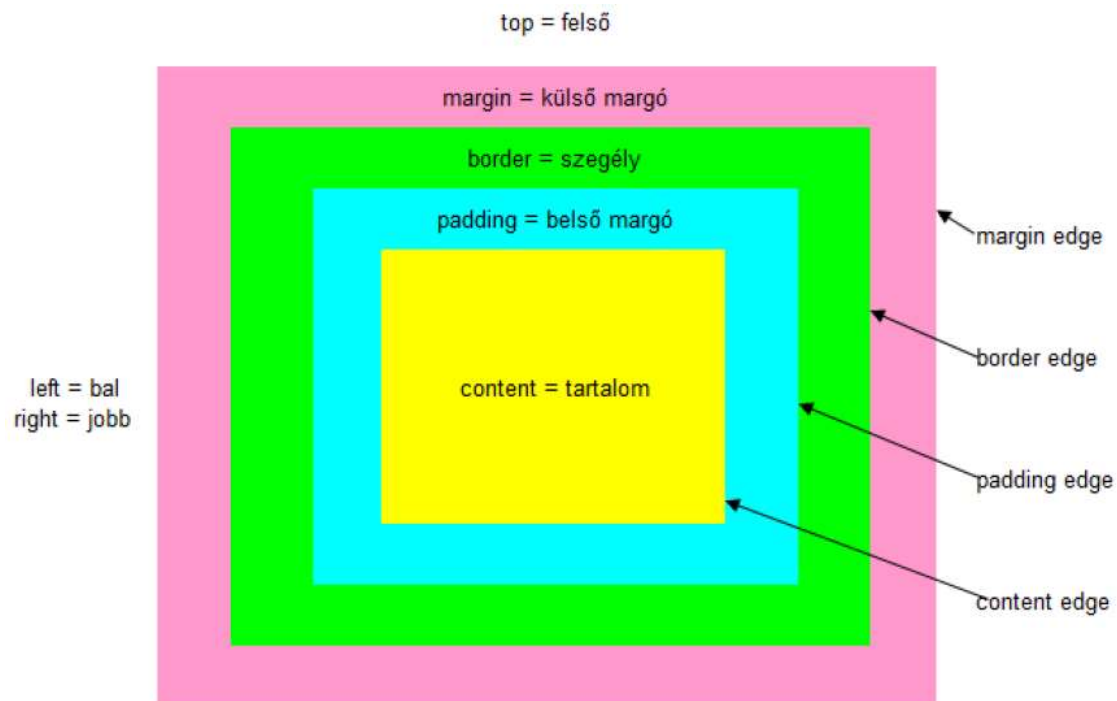
- **blokkszintűek** (*block-level box*) – pl. bekezdés, lista, táblázat, blokkidézet, div szakaszok
- **sorszintűek** (*line box*) – pl. egy szöveg egy sora
- **soron belüliek** (*inline-level box*) – pl. karakterek, szavak, képek egy soron belül



A dobozmodell egy vizuális formázási modell, amely a W3C¹⁹ előírásait tartalmazza a dokumentumban elhelyezett elemek megjelenésére. Ennek értelmében **a törzsrészben elhelyezett minden objektum egy téglalap alakú területen jelenik meg, amelynek kialakítását dobozmodellnek nevezzük**. A weboldal minden eleme egy téglalap alakú területen jelenik meg. Ha írunk egy bekezdést, beszúrunk egy táblázatot vagy egy képet, akkor ezek mind egy téglalap alakú területet, egy „dobozt” fognak elfoglalni a weboldalon. **Ehhez a dobozhoz a W3C szabványának²⁰ megfelelően hozzárendelhetünk belső margót (padding), szegélyt (border), illetve margót (margin)**. Ezek a téglalap alakú területet elfoglaló tartalmat (content) keretként veszik körül, a tartalomtól kifelé haladva, a felsorolt sorrendben (ld. a következő ábrát).

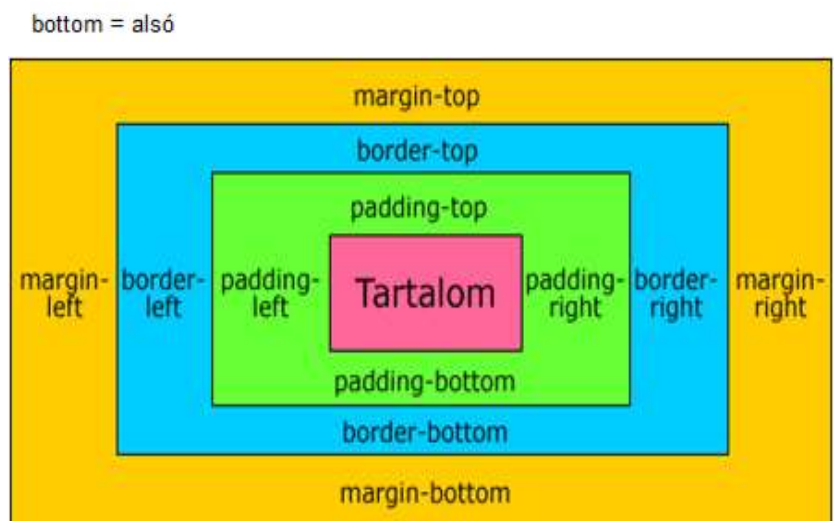
¹⁹ A World Wide Web Consortium (W3C) egy konzorcium, amely nyílt szoftver szabványokat („ajánlásokat”) alkot a világhálóra.

²⁰ További információk a dobozmodellről: <http://www.w3.org/TR/REC-CSS2/box.html>.



Az alábbi ábra szintén a dobozmodell illusztrálja. A belső margón (padding) megjelenik a háttér, ha van. Utána következik a szegély (border) valamilyen stílusban, majd legvégül a margó (margin), ami átlátszó, ezért a szülő elem háttére fog rajta megjelenni. Az ábráról az is leolvasható, hogy az egyes területeket melyik CSS-tulajdonsággal tudjuk elérni.

Összefoglalva: **a téglalapban legbelül helyezkedik a tartalom, amelyet körbevesz rendre a belső margó (kitöltés/bélés, padding), a szegély (border) és a (külső) margó (margin):**



- **content** = tartalom (a dobozban megjelenített tartalom, pl. szöveg, kép, további dobozok, ahol ennek a doboznak szélessége és magassága van)
- **padding** = belső margó, bélés (a tartalom és annak szegélye közötti üres rész nagysága)
- **border** = szegély (a megjelenített tartalmat és a belső margót együtt körülvevő, általában vonallal vagy képből készített rész)
- **margin** = külső margó, távolság (az elem távolságát határozza meg a szomszédos elemektől)

Általános tudnivalók:

- a margó, a szegély, a bélés és a tartalom mérete külön-külön változtatható
- a margó, a szegély és a bélés a doboz egyes oldalain vagy akár mindenütt hiányozhatnak, de egyszerre mindegyik lehet egymástól eltérő tulajdonságokkal meghatározott
- a margó lehet negatív értéke, így az elemek átfedése is létrehozható
- a bélés háttérszíne megegyezik a tartalom színével, de ezen lehet változtatni (ld. **box-sizing**)
- a margó mindig átlátszó, így a szülő elem állandóan látható marad
- a szegély a háttér elé (ha van háttérszín vagy háttérkép megadva), de a tartalom mögé kerül (ha átfedés van a szegély és a tartalom között)

Az elem szélességének és magasságának megfelelő beállításához minden böngészőben meg kell tudni, hogy ott hogyan működik a dobozmodell. **Általános szabály: ha egy elem szélességét és magasságát a CSS segítségével állítjuk be, akkor csak a tartalomterület szélességét és magasságát határozzuk meg. Egy elem teljes méretének kiszámításához (alapértelmezett helyzetben) mindig hozzá kell adni a padding, a border és a margin értékét is.**

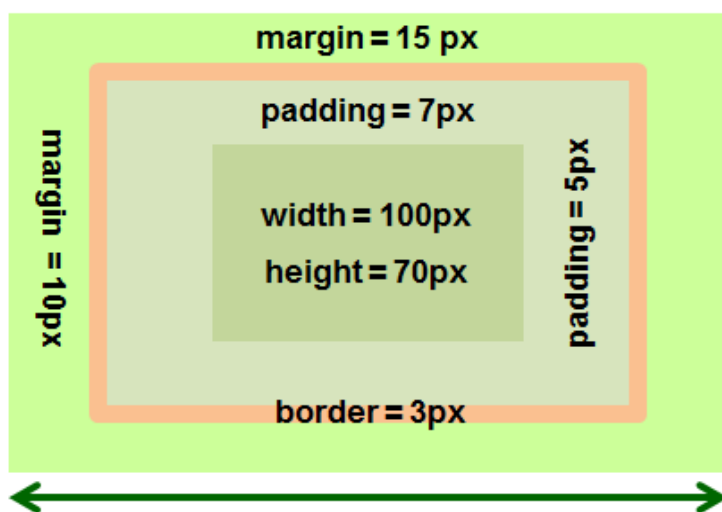
A szabvány szerint:

- a teljes objektum (a doboz) szélességét a jobb és a bal oldali margó, a jobb és a bal oldali szegély, a jobb és a bal oldali belső margó, valamint a tartalom szélességének összege,
- a teljes objektum (a doboz) magasságát pedig a felső és az alsó margó, a felső és az alsó szegély, a felső és az alsó belső margó, valamint a tartalom magasságának összege adja;
- a szélességhez (width) nem tartozik hozzá a padding, a border és a margin.

Megjegyzés: az Internet Explorer 8-nál korábbi verziói nem a W3C ezen előírásai szerint számolják a doboz méretét, hanem a tartalom méretébe a belső margót és a szegélyt is beleszámolták.²¹

Példa: ha egy elemre `width: 300px`, `padding: 30px`, `border: 20px`, akkor pl. Firefoxban az objektum szélessége 400 px (mert $300 + 2 \times 30 + 2 \times 20 = 400$), de pl. az Internet Explorerben a teljes szélesség a 300px, vagyis ebből veszi le a padding és a border részt, tehát a tartalomra rendelkezésre álló szélesség kisebb lesz.

a szélesség értékének kiszámítása példa alapján:



a dobozmodell egyes elemeire meghatározott értékek:

(bal/jobb) margin: 10px

(bal/jobb) border: 3px

(bal/jobb) padding: 5px

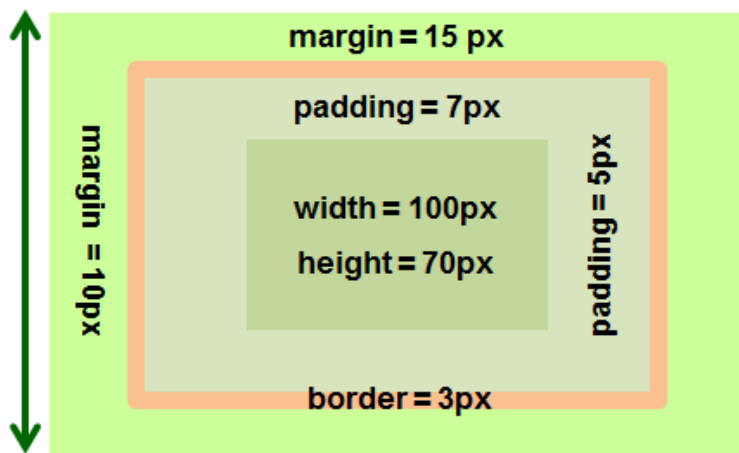
tartalom width: 100px

az elfoglalt szélesség:

$10 + 3 + 5 + 100 + 5 + 3 + 10 = 136\text{px}$

tehát az elfoglalt tényleges szélesség nagysága 136px

a magasság értékének kiszámítása példa alapján:



a dobozmodell egyes elemeire meghatározott értékek:

(alsó/felső) margin: 15px

(alsó/felső) border: 3px

(alsó/felső) padding: 7px

tartalom height: 70px

az elfoglalt magasság:

$15 + 3 + 7 + 70 + 7 + 3 + 15 = 120\text{px}$

tehát az elfoglalt tényleges magasság nagysága 170px

A CSS-dobozmodellben tehát alapértelmezés szerint az elemhez rendelt szélesség és magasság csak az elem tartalomdobozára vonatkozik. Ha az elemnek van szegélye vagy kitöltése, a ténylegesen elfoglalt hely kiszámításakor ezek hozzáadódnak a szélességhez és a magassághoz. Ez azt jelenti, hogy amikor beállítja a szélességet és a magasságot, módosítania kell a megadott értéket, hogy lehetővé tegye az esetlegesen hozzáadható szegélyeket vagy kitöltéseket. Például, ha négy olyan doboza van, amelyek szélessége: 25%; ha valamelyik rendelkezik bal vagy jobb oldali kitöltéssel vagy bal vagy jobb szegéllyel, akkor alapértelmezés szerint nem férnek el egy sorba a szülőtároló korlátain belül.

²¹ Dobozmodell-kijátszásnak (box modell hack-nek) nevezik azt a megoldást, amelynek segítségével mind a régebbi Internet Explorerben, mind a szabványnak megfelelő böngészőkben jól megjelenő weblapot lehet tervezni. (Lásd részletesebben az alábbi címen: <http://www.tutorial.hu/box-model-problema-es-megoldas-ie-alatt/>; http://en.wikipedia.org/wiki/Internet_Explorer_box_model)

A **box-sizing** (dobozméret-) tulajdonság segítségével viszont módosítható ez a viselkedés, mert ezzel az határozható meg, hogy hogyan kell kiszámítani egy elem helyfoglalását, a szélességét és magasságát, azaz legyenek a bélések és a szegélyhatárok benne vagy sem. (Ez a tulajdonság rokon a **background-clip** CSS-tulajdonsággal (ld. a hátterek tulajdonságait), mert az is egyfajta hatókört jelent: azt határozza meg, hogy a háttérkép és a háttérszín meddig terjedjen ki.)

Négyféle beállítást lehet alkalmazni:

- A **content-box** érték az alapértelmezett CSS-dobozméretezési viselkedést határozza meg, azaz sem a béléseket, sem a szegélyeket, sem a margókat nem számolja bele a megadott szélességi és magassági értékekbe. Így tehát ha pl. egy elem szélességét 100 képpontra állítja, akkor az elem tartalomdoboza 100 képpont széles lesz, és a szegély vagy kitöltés szélessége hozzáadódik a végső renderelt szélességhez, így az elem 100 képpontnál szélesebb lesz.
- A **border-box** érték viszont arra utasítja a böngészőt, hogy vegye figyelembe a szegélyeket és béléseket az elemek szélességéhez és magasságához megadott értékekben, vagyis a megadott **width** és **height** értékéből levonja a **padding** és a **border** értékét, majd az így kapott szélességi és magassági értékkel rendelkező terület lesz a **content** mérete. Ha tehát pl. egy elem szélességét 100 képpontra állítja, a szegély 5 képpontos, a bélések pedig 10 képpont nagyságúak, akkor a content szélességére $100 - 2 \cdot 5 - 2 \cdot 10 = 70$ képpont marad, tehát a tartalom doboza „összezsugorodik”, hogy elnyelje az extra szélességet. Ez általában sokkal könnyebbé teszi az elemek méretezését.²² (Ezt az alapértelmezést használják a böngészők a table, a select, a button és az input elemekhez is.)
- Az **initial** érték a korábban meghatározott értéket jelenti.
- Az **inherit** érték pedig azt mutatja a böngészőnek, hogy az elem a szülőelemtől örökli ezt a tulajdonságértéket.

Egy blokk szintű objektum esetén, ha többet akarunk láttatni a háttérképből, akkor megnövelhetjük az objektum méretét. Erre egy lehetőség az is, hogy nagyobb belső margót adunk az objektumnak.

Ha egy blokk szintű elemre nem adunk meg külön szélességi (**width**) és magassági (**height**) értéket, akkor a szélességét a szülőtől örökli (de alaphelyzetben akkora, mint a böngészőablak szélessége), a magasságát pedig a blokk szintű elem tartalmának mérete határozza meg!

A sorközi objektumok méretét viszont mindig a tartalmuk nagysága határozza meg. (A sor- és blokk szintű objektumok közötti átalakítást a **display** CSS-utasítással végezhetjük el!)

A dobozmodellel kapcsolatos tulajdonságok megadása – szélesség és magasság		
CSS-tulajdonság	Funkció	Példa, magyarázat
width	szélesség	<pre>#negyzetkep { width: 150px; }</pre> <p>elem szélességének megadása hosszúsági egységgel (px, pt, cm, em), a szülőelem szélességi nagyságának arányában százaléktékben (%), értéke lehet az auto alapbeállítás (ekkor a böngésző maga határozza meg az értéket a többi érték figyelembevétele mellett), az initial (ez az alapértelmezett értéket veszi figyelembe) vagy inherit (a szülőtől örökölt érték) is</p>
max-width	legnagyobb szélesség	<pre>#negyzetkep { max-width: 500px; }</pre> <p>elem legnagyobb szélességének megadása – nem lehet szélesebbre állítani ennél az értéknél (alapértelmezetten none ez az érték, azaz ha nem adjuk meg, akkor az egyéb tulajdonságoktól függ a szélesség értéke)</p>

²² Gyakran hasznos az elrendezési elemek dobozméretének beállítása **border-box** értékre. Ez sokkal könnyebbé teszi az elemek méretének kezelését, és általában kiküszöböli számos buktatót, amelyekbe belebotlhat a tartalom elrendezése közben. Másrészt, ha pozíciót használunk: relatív vagy pozíció: abszolút, a **box-sizing: content-box** használata lehetővé teszi, hogy a pozicionálási értékek relatívak legyenek a tartalomhoz, és függetlenek a szegély és a kitöltés méretének változásától, ami néha kívánatos.

A dobozmodellel kapcsolatos tulajdonságok megadása – szélesség és magasság		
CSS-tulajdonság	Funkció	Példa, magyarázat
min-width	legkisebb szélesség	<code>#negyzetkep { min-width: 50px; }</code> <i>elem legkisebb szélességének megadása</i> – nem lehet keskenyebbre venni ennél az értéknél (alapértelmezetten 0 ez az érték)
height	magasság	<code>#negyzetkep { height: 150px; }</code> <i>elem magasságának megadása hosszúsági egységgel</i> (px, pt, em) vagy a szülőelem magassági nagyságának arányában százalékértékben (%), lehet még az auto alapbeállítás (ekkor a böngésző maga határozza meg az értéket a többi érték figyelembevételével), az initial (ez az alapértelmezett értéket veszi figyelembe) vagy inherit (a szülőtől örökölt érték) is
max-height	legnagyobb magasság	<code>#negyzetkep { max-height: 300px; }</code> <i>elem legnagyobb magasságának megadása</i> – nem lehet magasabbra venni ennél az értéknél (alapértelmezetten none ez az érték, azaz ha nem adjuk meg, akkor az egyéb tulajdonságoktól függ a magasság értéke)
min-height	legkisebb magasság	<code>#negyzetkep { min-height: 80px; }</code> <i>elem legkisebb magasságának megadása</i> – nem lehet alacsonyabbra venni ennél az értéknél (alapértelmezetten 0 ez az érték)

Minden elem esetén igaz, hogy a tartalmat (*content*) körülvevő részek tulajdonságait (padding, border és margin) megadhatjuk egyszerre mind a négy irányban vagy külön-külön. **Ha egyszerre adjuk meg a négy oldal jellemzőit, akkor azok értékeit az óramutató járásával egyező irányban felsorolva: felül (top), jobbról (right), alul (bottom) és balról (left).** A méreteket mindegyik esetben a szokásos mértékegységekkel határozhatjuk meg, százalékos megadás esetén azt a dokumentum teljes szélességéhez, illetve magasságához viszonyítva számolja ki. A belső margó háttere a tartalom háttere lesz, a külső margó pedig mindig átlátszó. **A belső margót a tartalom szegélytől való eltávolítására, a külső margót pedig a szomszédos elemektől való távolság szabályozására szoktuk használni.**

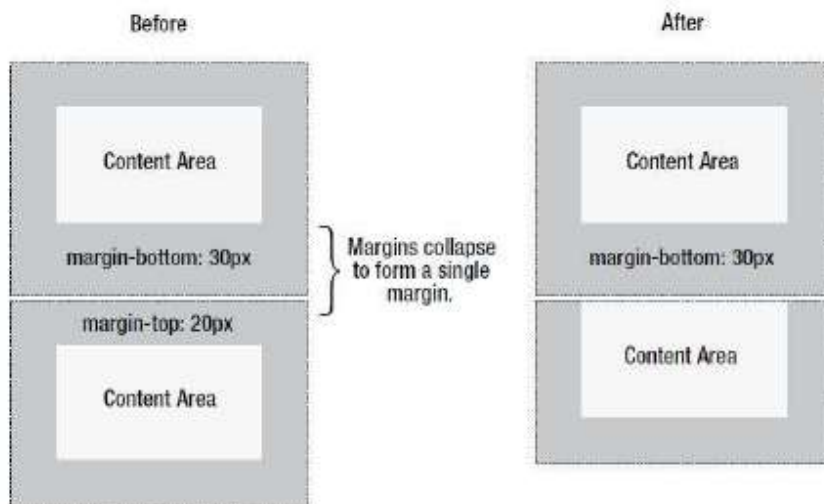
A következőkben a dobozmodell három további tulajdonságának (margin, border, padding) beállításához használt CSS-tulajdonságok leírását találja.

A dobozmodellel kapcsolatos tulajdonságok megadása – külső margók (eltartások)		
CSS-tulajdonság	Funkció	Példa, magyarázat
margin-top margin-right margin-bottom margin-left	külső margó értéke	<code>body { margin-top: 100px; margin-right: 10px; margin-bottom: 0; margin-left: 10%; }</code> felső / jobb / alsó / bal margó nagyságának pozitív és negatív értékű meghatározása , amelynek értéke lehet szám (hosszmérték: px, pt, cm, em vagy %: a befoglaló elem szélességének arányában), az auto érték (ekkor a böngésző automatikusan kiszámítja a bal és jobb margó nagyságát, amellyel az objektumok középre igazítja a szülőhöz képest) vagy a befoglaló elemtől örökölt érték, azaz inherit
margin	összevont tulajdonság a külső margó értékéhez	hasonlóan a font tulajdonsághoz, itt is összevonható a beállítás egyetlen sorba, ha a margin tulajdonság-megnevezést használjuk – ebben az esetben a megadott értékeket a böngésző az óramutató járása szerinti (felső – jobb – alsó – bal) sorrendben fogja azt értelmezni <ul style="list-style-type: none"> négy egyedi érték megadásával: felső, jobb, alsó és bal sorrendben <code>body { margin: 100px 10px 0 10%; }</code> három egyedi érték megadásával: felső (20px), bal és jobb azonos (itt most ez az auto érték), alsó érték (30px) sorrendben meghatározva <code>#kozepre { margin: 20px auto 30px; }</code>

A dobozmodellel kapcsolatos tulajdonságok megadása – külső margók (eltartások)		
CSS-tulajdonság	Funkció	Példa, magyarázat
(folytatás)		<ul style="list-style-type: none"> két egyedi érték megadásával: a felső-alsó és a bal-jobb azonosak <code>#doboz { margin: 50px 20px; }</code> egyetlen egyedi érték megadásával: mindegyik margó azonos értékű <code>.tavolit { margin: 15px; }</code>

Lényeges különbség a belső és a külső margók között, hogy míg a belső margók kizárólag pozitív értékűek lehetnek, addig a külső margók pozitív és negatív értékeket egyaránt felvehetnek. Negatív margó esetén akár átfedés is kialakítható az elemek között.

Függőleges elrendezés esetén két elem közötti távolság együttes megadásakor (tehát ha az első elemhez alsó és a mögötte elhelyezett elemhez pedig felső margót adunk meg, akkor) nem a két margó összegével fog megegyezni a két elem távolsága, hanem csak a nagyobbik értékkel.



A padding segítségével a doboztartalom és a szegély közötti távolságot növelhetjük.

A dobozmodellel kapcsolatos tulajdonságok megadása – bélések (belső távolítás)		
CSS-tulajdonság	Funkció	Példa, magyarázat
padding-top padding-right padding-bottom padding-left	belső margó értéke	<pre>#tartalom_tavolit{ padding-top: 0.2em; padding-right: 0.5em; padding-bottom: 0.2em; padding-left: 0.5em; }</pre> <p>felső / jobb / alsó / bal belső margó (bélés) nagyságának meghatározása, amelyet számmal (px, pt, cm, em, %: a befoglaló elem szélességének arányában) vagy az inherit (öröklődő) értékkel adhatjuk meg</p>
padding	összevont tulajdonság a belső margó értékéhez	<pre>#tartalom { padding: 0.2em 0.5em 0.2em 0.5em; }</pre> <p>hasonlóan a margin tulajdonsághoz, itt is összevonható a beállítás egyetlen sorba, ha a padding tulajdonság-megnevezést használjuk és ugyanolyan szabályok szerint történik a bélés különböző oldali értékeinek meghatározása (4 / 3 / 2 vagy egyetlen értékkel)</p>

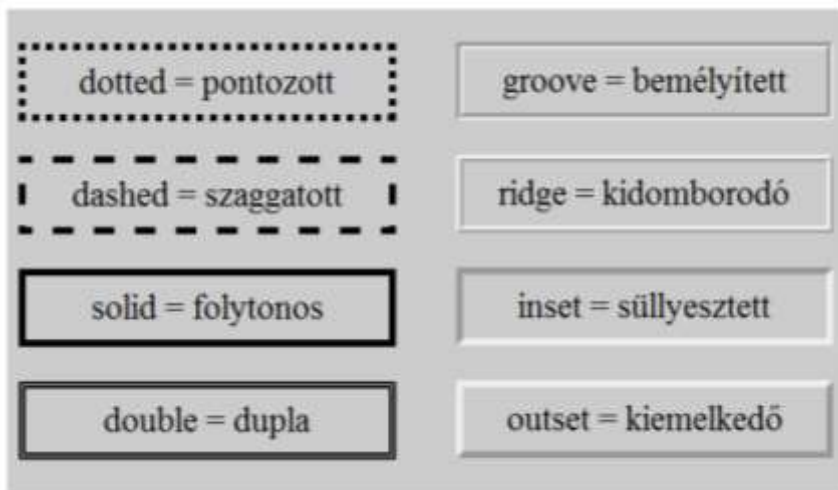
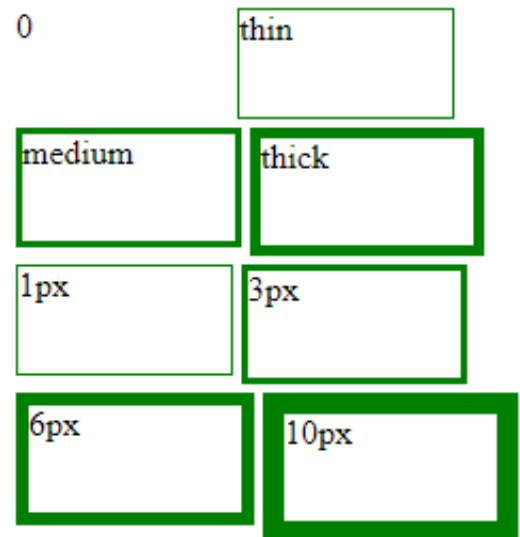
A szegély tulajdonsággal a megjelenített tartalom és a bélés együttesét alkotó doboz köré kívültre kerülnek. A szegélyt alkothatja előre meghatározott stílusú vonal vagy szabadon választott kép részlete is akár. (A szegély a háttér elé, de a tartalom mögé kerül.)

A dobozmodellel kapcsolatos tulajdonságok megadása – szegélyek		
CSS-tulajdonság	Funkció	Példa, magyarázat
border-...-width border-...-style border-...-color	szegélyek egyedi beállítása az egyes oldalakhoz	<pre>#alul { border-bottom-width: 10px; border-bottom-style: double; border-bottom-color: red; }</pre> <p>felső / jobb / alsó / bal szegély egyes tulajdonságainak (width = vastagság, style = stílus, color = szín) megadása oldalanként és jellemzőnként külön-külön definiálva</p>

A dobozmodellel kapcsolatos tulajdonságok megadása – szegélyek		
CSS-tulajdonság	Funkció	Példa, magyarázat
(folytatás)		<ul style="list-style-type: none"> width: a szegély vastagságát számmal (px, pt, cm, em, %) vagy szövegesen (<i>thin</i>: vékony, <i>medium</i>: átlagos, <i>thick</i>: vastag) adhatjuk meg style: a szegély stílusát szöveges megnevezéssel adhatjuk meg: lehet <i>none</i> (nincs), <i>hidden</i> (rejtett), <i>dotted</i> (pontosított), <i>dashed</i> (szaggatott), <i>solid</i> (folytonos vonal), <i>double</i> (dupla vonal), <i>groove</i> (barázdás, bemélyített), <i>ridge</i> (peremes, kidomborodó), <i>inset</i> (besüllyesztett), <i>outset</i> (kiemelkedő), <i>inherit</i> (öröklődő) color: a szegély színét meghatározhatjuk RGB-színkóddal, a szín angol nevével, illetve két szöveges értékkel: a <i>transparent</i> (átlátszó) vagy az <i>inherit</i> (öröklődő) nevekkkel
border-width border-style border-color	a szegélyek egyes jellemzőinek megadása mindegyik oldalhoz	<pre>#keret { border-width: 10px; border-style: double; border-color: green; }</pre> <p>a szegély egyes tulajdonságainak megadása jellemzőnként külön-külön definiálva úgy, hogy az az összes oldalra érvényes legyen ezek a tulajdonságok (hasonlóan a padding és a margin tulajdonságokhoz) a négy oldalra eltérő vagy részben azonos értékekkel is megadhatók:</p> <ul style="list-style-type: none"> négy egyedi érték megadásával: felső, jobb, alsó és bal sorrendben <pre>#keretes { border-width: 1px 2px 3px 4px; border-style: solid double ridge inset; border-color: red blue green silver; }</pre> három egyedi érték megadásával: felső, bal és jobb azonos, alsó érték sorrendben meghatározva <pre>#kozepra { border-width: 1px 2px 5px; border-style: dotted solid double; border-color: blue red green; }</pre> két egyedi érték megadásával: a felső-alsó és a bal-jobb azonosak <pre>#paros_szegely { border-width: 1px 5px; border-style: solid double; border-color: silver blue; }</pre> egyetlen egyedi érték megadásával: mindegyik szegély azonos értékű <pre>#azonos { border-width: 2px; border-style: groove; border-color: green; }</pre>
border-top border-right border-bottom border-left	szegélyek beállítására alkalmazott tulajdonságok	<pre>#kepszegely { border-top: 30px groove #FF0000; border-right: 10px inset #0000FF; border-bottom: 30px ridge #FF0000; border-left: 10px outset #0000FF; }</pre> <p>felső / jobb / alsó / bal szegély tulajdonságainak összevont módon történő meghatározása úgy, hogy a megadott tulajdonságok egy-egy oldalra legyenek érvényesek</p>
border	összevont tulajdonság a szegély értékeihez	<pre>#kepszegely2 { border: 10px dotted #00FF00; }</pre> <p>hasonlóan a margin és a padding tulajdonsághoz, itt is összevonható a beállítás egyetlen sorba, ha a border megnevezést használjuk</p>

Fontos!

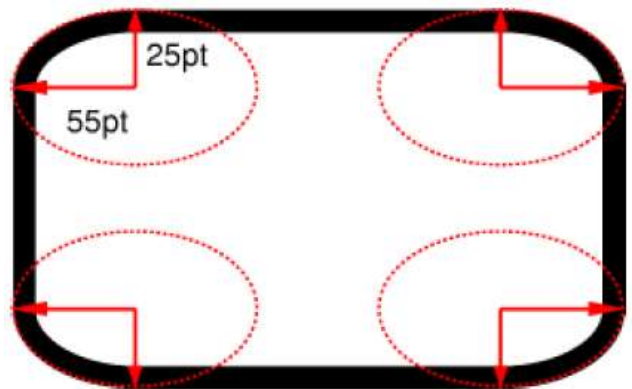
Ha összevont alakot (shorthand) használunk és kódolási hibát vétünk, akkor az összes érték érvénytelenítésre kerül, nemcsak a hibásan kódolt rész, részletező kódolásnál (longhand) viszont csak a helytelenül kódolt beállítás fogja a böngésző figyelmen kívül hagyni.

az alapvető szegélystílusok:**példák szegélyvastagságra:****Dobozokkal kapcsolatos egyéb tulajdonságok****A) Dobozsarkok lekerekítése**

A szegélyek lekerekítéséhez negyed-ellipsziseket használunk, amelynek során a sarkonként külön-külön megadott 2-2 sugárértékkel határozzuk meg a létrejövő sarokszegély alakját. Az egyes sarkoknak nem kell szükségszerűen azonos lekerekítéssel rendelkezniük, eltérő lekerekítésekkel aszimmetrikus alakzatok is kialakíthatók.

A lekerekítési sugarak pt-ben, px-ben, %-ban vagy em-ben egyaránt megadhatók, a két érték közül az első mindig a vízszintes sugarat, a második a függőleges sugarat jelöli.

Ha a második sugár értéke nincs megadva, akkor az egyenlőnek tekintendő az elsőével, így egy-egy saroknál azonos sugarak esetén az ellipszis speciális eseteként negyedkört kapunk. Ha bármelyik sugárérték 0, akkor szögletes (azaz nem lekerekített) a tekintett szegélysarok. Százalékos értékmegadás esetén a vízszintes sugárnál a szegélydoboz szélessége, a függőleges sugárnál a szegélydoboz magassága a 100%.



A saroklekerekítésekhez a **border-radius** tulajdonságot (mint összevont jellemzőt), illetve az egyes sarkok kerekítéséhez a **border-top-left-radius** (bal felső), **border-top-right-radius** (jobb felső), a **border-bottom-left-radius** (bal alsó) és a **border-bottom-right-radius** (jobb alsó) tulajdonságokat alkalmazhatjuk.

Dobozsarkok lekerekítése egyedi saroktulajdonságok megadásával		
CSS-tulajdonság	Funkció	Példa, magyarázat
border-top-left-radius	bal felső sarok lekerekítése	<pre>#balfelso { border: 5px solid black; border-top-left-radius: 45pt 30pt; }</pre> <p>a bal felső sarok szegélyének lekerekítési mértéke hosszúsági egységgel (px, pt, cm, em) adható meg vagy a doboz szélességi értékének arányában százalékértékben (%) is meghatározható – ha egy saroknál megadott két érték azonos, akkor negyedkörös, egyéb esetekben, azaz két különböző értékkel meghatározott lekerekítési értéknél ellipszis alakú lesz a szegély kerekítése</p>

Dobozsarkok lekerekítése egyedi saroktulajdonságok megadásával		
CSS-tulajdonság	Funkció	Példa, magyarázat
border-top-right-radius	jobb felső sarok lekerekítése	<pre>#jobbfelso { border: 5px solid black; border-top-right-radius: 25px 45px; }</pre> <p>a jobb felső sarok szegélyének lekerekítési mértéke hosszúsági egységgel (px, pt, cm, em) és százalékértékben (%) egyaránt meghatározható</p>
border-bottom-left-radius	bal alsó sarok lekerekítése	<pre>#balalso { border: 5px solid black; border-bottom-left-radius: 25% 35%; }</pre> <p>a bal alsó sarok szegélyének lekerekítési mértéke hosszúsági egységgel (px, pt, cm, em) és százalékértékben (%) egyaránt meghatározható</p>
border-bottom-right-radius	jobb alsó sarok lekerekítése	<pre>#jobbalsos { border: 5px solid black; border-bottom-right-radius: 8em 5em; }</pre> <p>a jobb alsó sarok szegélyének lekerekítési mértéke hosszúsági egységgel (px, pt, cm, em) és százalékértékben (%) egyaránt meghatározható</p>

Ha a sarkok lekerekítésénél a bal alsó érték nincs megadva, akkor annak értéke meg fog egyezni a jobb felső sarokra beállított értékkel. Ha a jobb alsó nincs definiálva, akkor annak nagysága a bal felső sarok lekerekítési értékével lesz azonos.

Dobozsarkok lekerekítése összevont tulajdonsággal		
CSS-tulajdonság	Funkció	Példa, magyarázat
border-radius	összevont tulajdonság a szegélyek lekerekítéséhez	<p>Ha összevont tulajdonságot használunk a sarkok lekerekítéséhez, akkor elsőként a vízszintes sugárértékeket kell felsorolnunk a bal felső saroktól indulva az óramutató járási sorrendjében (bal felső, jobb felső, jobb alsó, bal alsó) szóközzel elválasztva, majd / jelet követően ugyanilyen sorrendben a függőleges sugárértékeket kell megadni szóközzel tagolva:</p> <pre>#mindenkerekitve_1 { border: 5px solid black; border-radius: 45pt 25px 25% 8em / 30pt 45px 35% 5em; }</pre> <p>Ha csak abban van eltérés, hogy az egyes sarkoknál a vízszintes és a függőleges sugárértékek különbözőek, akkor elegendő egyszer kiírni azok mértékét és / jellel kell elválasztani azokat egymástól. Ebben az esetben az első érték szintén a vízszintes, a második pedig a függőleges lekerekítési értéket mutatja:</p> <pre>#mindenkerekitve_2 { border: 5px solid black; border-radius: 20px / 30px; }</pre> <p>Ha minden sarok minden lekerekítési értéke azonos nagyságú, akkor elegendő egyetlen értéket megadni:</p> <pre>#mindenkerekitve_3 { border: 5px solid black; border-radius: 15%; }</pre>

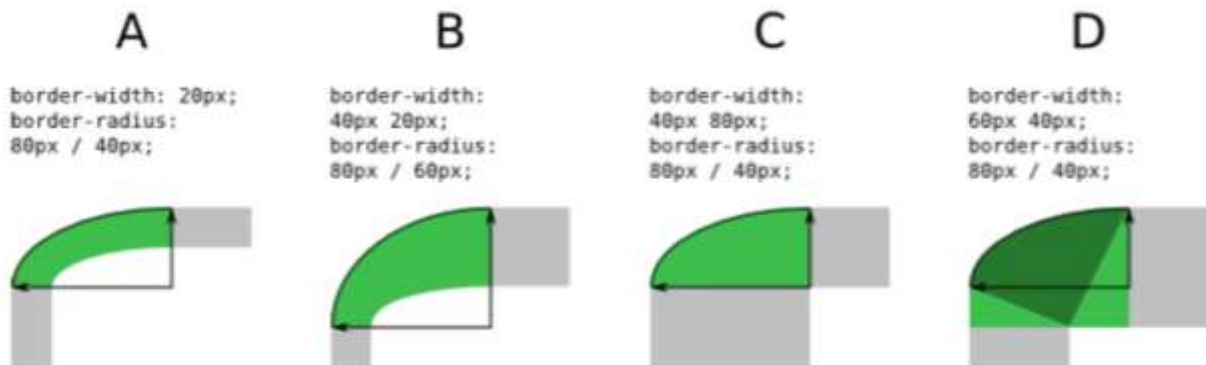
Néhány általános tudnivaló a sarkok lekerekítésével kapcsolatban:

- A lekerekítés nem függ a szegély stílusától (pl. egyszerű, dupla, pontozott, szaggatott, süllyesztett stb.), tehát valamennyi szegély-stílus követi a szegély lekerekítését.
- Ha nem azonos vastagságú szegélyek találkoznak egy lekerekített saroknál, akkor a lekerekítés során folyamatos átmenet lesz az egyik vastagságtól a másikba (ld. a köv. oldal bal képét).

- Ha a szegély vastagsága nagyobb, mint a lekerekítési sugár, akkor csak a szegély külső éle fog lekerekedni, a belső szegély szögletes marad (ld. az oldalsó képen a jobb oldali lekerekítést).



- Ha a lekerekítés sugara nagyobb, mint a szemközti saroktól mért távolság, akkor a kerekítés íve 90 fokosnál kisebb lesz (ld. az bal oldali alábbi ábrát).
- A sarokkerekítéseknek nem szabad átfedniük egymást, azaz két szomszédos szegélysarok sugarának összege nem lehet nagyobb a szegélydoboz méreténél. Ha az összeg mégis meghaladja ezt a mértéket, akkor a böngészők arányosan lecsökkentik a sugarak méretét az átfedés határhelyzetéig. Ez a megjelenítésben a kódoláshoz képest kisebb eltérést okoz.

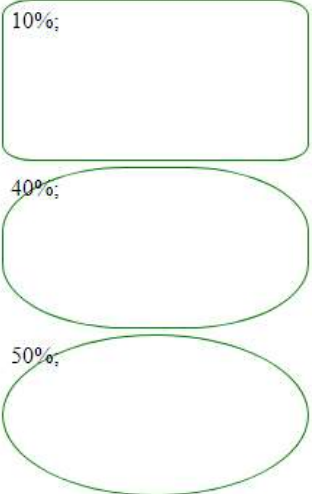
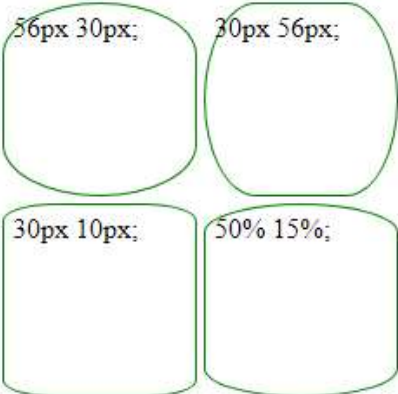



- Az érintkező szegélyek stílus- és színátmenetének is meg kell történnie a lekerekítés során. Ekor az átmenet középpontja a lekerekítési görbe azon pontja, amelynek a vízszinteshez viszonyított szöge arányos a szegélyvastagságok hányadosával. (Pl. ha a felső és a jobb szegély vastagsága egyenlő, akkor a hányados 1, tehát 45 fokos szögnél lesz az átmenet, mint az alábbi ábra A példáján. Ha a felső szegély kétszer vastagabb a bal oldalnál, akkor az átmenet a vízszinteshez képest a 30 foknál lesz, mint az alábbi ábra B példájában.) Ennek a szögnek a mentén, a szegély külső és belső ívét összekötő egyenes mentén lesz az éles szín- és stílusátmenet. Az alábbi ábrán látható lekerekítési adatoknál ezek az átmenetek ezért zöld területekre esnek. Azonban a D ábra esetében a lekerekítési sugarak által definiált négyszög nem foglalja magába a belső görbe középpontját (szögletes sarok), így az átmeneti területet ki kell úgy terjesztetni (sötétzöld terület), hogy a szögletes sarok is belekerüljön.

Néhány mintapélda a lekerekítésekre:

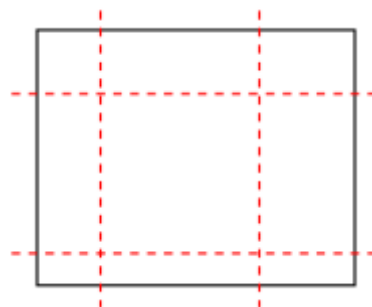
<p>50px szélességű és magasságú, 5px vastag szegélyű dobozok mindegyik sarkának ugyanolyan mértékű kerekítése képpont (px) értékkel meghatározva</p>	<p>100px szélességű és magasságú, 1px vastag szegélyű dobozok sarkainak eltérő mértékű kerekítése különböző mértékegységekkel és ellipszisekkel (két értékkel) definiálva</p>	<p>100px szélességű és magasságú, 5px vastag szegélyű dobozok sarkainak eltérő mértékű kerekítése különböző mértékegységekkel és negyedkörökkel kialakítva</p>
------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------

Néhány további mintapélda a lekerekítésekre:

 <p>10%;</p> <p>40%;</p> <p>50%;</p> <p>200px szélességű és 100px magasságú, 1px vastag szegélyű dobozok mindegyik sarkának ugyanolyan mértékű kerekítése %-értékkal meghatározva</p>	 <p>56px 30px;</p> <p>30px 56px;</p> <p>30px 10px;</p> <p>50% 15%;</p> <p>100px szélességű és magasságú, 1px vastag szegélyű dobozok sarkainak eltérő mértékű kerekítése különböző mértékegységekkel és ellipszisekkel (két értékkel) kialakítva</p>	 <p>Első;</p> <p>Második;</p> <p>Első;</p> <p>Második;</p> <p>200px szélességű és 100px magasságú, 3px vastag szegélyű dobozok sarkainak eltérő mértékű kerekítése különböző mértékegységekben meghatározva és ellipszisekkel (azaz két értékkel) definiálva</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Lehetséges gyakorlati / alkalmazási területek:**B) Képből készített szegélyek**

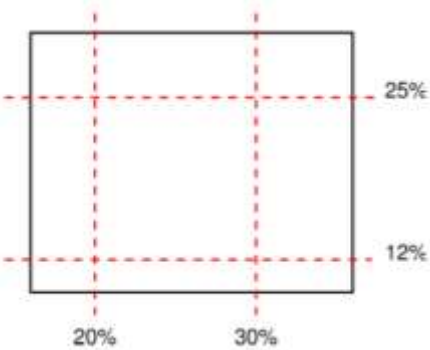
Egy dobozhoz nemcsak egy előre meghatározott vonalstílussal készíthető szegély, hanem **egy közötti algoritmus²³ alkalmazásával egy teljesen választott képet is felhasználhatunk szegélyként**. Ehhez a képet két vízszintes és két függőleges vonallal 9 darabra osztjuk és ezek a darabok alkotják majd a szegély oldalait, sarkait, illetve a középső rész kihagyásával vagy megtartásával a tartalomdoboz és a belső margó hátterét.



Egy képből készített szegély a `border-image` tulajdonsággal hozható létre, amely megadható összevont és részletezett alakban is.

- A szegély kialakításához használt képet a `border-image-source` tulajdonsággal definiáljuk, amelynek értéke vagy **none** (ekkor nem lesz képből szegély) vagy a kép elérési útja az **url()** függvénnyel meghatározva: pl. `border-image-source: url(border.png);`
- A `border-image-slice` jellemző a kép szélétől mérve a teljes képméret (ez a 100%) százalékában adja meg, hogy mekkora részeket használunk fel a képből a szegély céljára. Az értékeket szóközzel, vessző nélkül felsorolva kell megadni, a kötelező sorrend a már megszokott (fentről indul és az óramutató járásával megegyező irány: a kép felső, jobb oldali, alsó és bal oldali élei): pl. `border-image-slice: 25% 30% 12% 20%;` (ld. a következő ábrát).

²³ Annyiban kötött az algoritmus, hogy a szegély kialakítása csak egyetlen képből (nem pedig esetleg több kép kombinációjából) hozható létre, a kép csak a leírt módon darabolható (ferde egyenesek vagy kilencnél több rész nem lehetséges), és a szegélyrészek helyzete csak a kép darabolási sémája szerinti lehet (nem cserélhetők fel például az egyes oldalak).

- ha a bal oldali él értéke hiányzik, akkor az megegyezik a jobb oldalival
 - ha az alsó hiányzik, akkor az azonos a felsővel
 - ha a jobb oldali hiányzik, akkor az ugyanaz, mint a felső
 - nem értelmezhetők a negatív értékek, illetve a 100%-nál nagyobbak (ekkor a képnél nagyobbnak megadott részeket 100%-nak veszi a szabvány)
 - a kép középső része alapértelmezetten üresnek tekintett – ha a **fill** kulcsszót is definiáljuk a %-os értékek után, akkor az a rész látható marad és ha van háttér beállítva, akkor azt el fogja fedni
 - az egymással szemben lévő kijelölt képdarabok átfedhetik egymást:
 - ha a jobb és a bal oldali élek szélességének összege \geq mint a kép szélessége, akkor a szegély alsó és felső része, illetve a középső rész üres lesz
 - ha a felső és az alsó élek magasságának összege \geq mint a kép magassága, akkor a szegély jobb és bal oldali része, tovább a középső rész lesz üres
- 
- The diagram illustrates a rectangular image with dashed lines representing border widths. The top border is labeled 25%, the bottom border is labeled 12%, the left border is labeled 20%, and the right border is labeled 30%.
- A szegély oldalainak szélességét a **border-image-width** tulajdonság relatív (azaz %-ban megadott) vagy abszolút (azaz képpontban, px-ben definiált) nagyságban megadott értékei definiálják: pl. **border-image-width: 40px 30px 20px 30px;** .
 - ha a bal oldali érték hiányzik, akkor az megegyezik a jobb oldalival
 - ha az alsó hiányzik, akkor az azonos a felsővel
 - ha a jobb hiányzik, akkor annak értéke az ugyanaz, mint a felső
 - nem értelmezhetők a negatív értékek
 - ha két szemben lévő szegély él a megadott nagy szélességeik miatt átfedné egymást, a szabvány arányosan lecsökkenti a szélességeket az átfedés határhelyzetéig
 - további lehetséges érték **auto** – ekkor a szegély kialakításához felhasznált kép kijelölt darabjának eredeti, saját mérete lesz a szegély szélessége
 - A **border-image-outset** jellemzővel a szegély helyzete a szegély dobozától kifelé tolható: pl. **border-image-outset: 20px;** (ahol az értékmegadás a már ismert módon történik: a felső-jobb-alsó-bal sorrendet követi). Az érték megadható képpontban vagy a szegélyszélesség többszöröseként meghatározott számként: pl. a **border-image-outset: 1 2;** beállítás azt jelenti, hogy a felső és az alsó eltolás szegélyvastagságnyi, a bal és a jobb eltolás viszont a szegély vastagságának kétszerese.
 - A **border-image-repeat** tulajdonság a képdarabnak a szegélyben történő ismétlődését szabályozza, amelynek lehetséges értékei:
 - **stretch** (ez az alapértelmezett): a képdarab olyan mértékben van szét húzva, hogy kitöltse a szegélyszakaszt
 - **repeat**: a teljes kitöltendő szegélyszakaszt a képdarab méretének változatlanul tartásával, annak ismétlésével összefüggően tölti ki
 - **round**: ha a képdarab egész számú többszöröse nem pontosan tölti ki a szegélyszakaszt, akkor automatikusan úgy méreteződik át, hogy az egész számú többszöröse illeszkedjen a szegélyszakasz kitöltéséhez
 - **space**: az első és az utolsó képdarab a szegélyszakasz széleihez illeszkedik és a többi képdarab változatlan méretben olyan térközzel helyezkedik el közöttük, hogy egyenletesen kitöltsék a szegélyszakaszt
 - **az értékek megadása két kulcsszóval történik**: az első a vízszintes, a második a függőleges szegélyszakaszokra vonatkozik, pl. **border-image-repeat: round space;** (ha a második érték nincs megadva, akkor az azonosnak tekintendő az elsővel)
 - A **border-image** tulajdonságok értékei összevonva is megadhatók, ekkor az értékek sorrendje a következő: forráskép (**-source**), levágás (**-slice**), keretvastagság (**-width**), kifelé tolás (**-outset**) és ismétlés (**-repeat**).

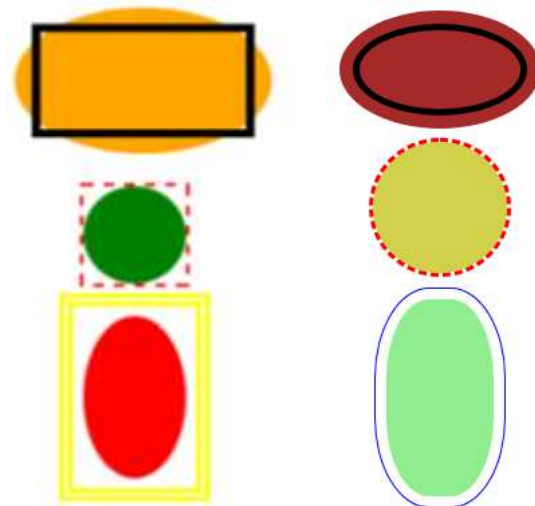
Néhány példa és minta képből készített szegélyre:

Ezt fogjuk látni	CSS-kód
	forráskép
	<pre>#borderimg1 { border: 10px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 10%; border-image-repeat: round; }</pre>
	<pre>#borderimg2 { border: 10px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 30; border-image-repeat: round; border-image-width: 25px; }</pre>
	<pre>#borderimg3 { border: 10px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 10% 20% 30% 40%; border-image-repeat: round; }</pre>
	<pre>#borderimg4 { border: 15px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 30; border-image-repeat: repeat; }</pre>
	<pre>#borderimg5 { border: 15px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 30; border-image-repeat: round; }</pre>
	<pre>#borderimg6 { border: 15px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 30; border-image-repeat: space; }</pre>
	<pre>#borderimg7 { border: 15px solid transparent; padding: 15px; border-image-source: url(border.png); border-image-slice: 30% fill; border-image-repeat: space; }</pre>

C) Körvonalak

A körvonalak (kontúrok) a szegélyekhez hasonlóan valamilyen elem (pl. gomb, képtérkép) vizuális kihangsúlyozására szolgálnak, amelyet az `outline` tulajdonság-névvel tudunk kódolni. Mind kódolásában, mind megjelenítésében hasonlít a standard vonaltípusú szegélyekre, de vannak attól lényeges eltérései:

- csak szögletesek lehetnek (egyes újabb böngészőknél már követik az alakzat formáját is),
- nem foglalnak el helyet, nem befolyásolják az elem méretét, mert mindig rajta és nem az elemen kívül jelennek meg,
- tetszőleges alakú elemhez mindig téglalapként illeszkednek,
- a körvonal egyes oldalait nem lehet külön-külön formázni.



A szegélyhez hasonlóan, három tulajdonságot lehet meghatározni: a körvonal szélességét (`-width`), a stílusát (`-style`) és a színét (`-color`), de összevontan is megadhatjuk az értékét a méret, stílus és szín sorrendben: pl. `outline: 3px solid blue;`

Alapértelmezetten mindig a szegély élén jelenik meg a körvonal (ha nincsen szegély, akkor is matematikailag létezik ez a hely 0 szegélyszélességgel), ezt a helyzetet lehet az `outline-offset` tulajdonsággal az elemhez közelebbre (ekkor az értéke negatív) vagy attól távolabbra (pozitív érték) eltolni.

Mivel a körvonal nem befolyásolja olyan értelemben a formázást, hogy a dobozmodellben nincsen a számára kihagyva hely, változatlanul a helyén maradó más szomszédos elem is rákerülhet.

D) Dobozok árnyékolása

A doboz-árnyék, azaz a `box-shadow` tulajdonság egy vagy több vetett árnyékot jelenít meg egy dobozhoz. A külső árnyék a szegélyen kívülre vetődik, mintha a szegélydoboz átlátszatlan lenne, a belső árnyék a belső margó szélétől befelé vetődik, mintha attól kifelé lenne minden átlátszatlan. Ha a doboznak lekerekített szegélyei vannak, akkor az árnyék alakja követi a lekerekítéseket.

A doboz-árnyék tulajdonság árnyékok vesszővel elválasztott listájából áll, amelyben mindegyik árnyék 2-4 db hosszúságértéket, színt és az opcionális *inset* kulcsszót tartalmazza. A kihagyott hosszúságok 0-nak értendők, ha nincsen szín megadva, a böngésző a szabvány szerintit rendeli hozzá (a gyakorlatban viszont nem mindig, ezért érdemes mindig megadni a színt is).

```
box-shadow: none|h-offset v-offset blur spread color
            |inset|initial|inherit;
```

Az egyes árnyékok komponensei:

- az 1. hosszúság az árnyék vízszintes kiterjedése: pozitív értéknél a doboztól jobbra, negatív értéknél a doboztól balra rajzolódik ki az árnyék
- a 2. hosszúság az árnyék függőleges kiterjedése: pozitív értéknél a doboztól lefelé, negatív értéknél a doboztól felfelé rajzolódik ki az árnyék
- a 3. hosszúság az életlenítési (blur) távolság: negatív érték nem megengedett; ha 0 az érték, akkor az árnyék széles éles, egyébként pedig minél nagyobb pozitív érték, annál inkább életlenedik az árnyék
- a 4. hosszúság a szórás/kiterjedés/nyílásszög (spread) távolság: pozitív értéknél az árnyék alakja minden irányban kiterjed a megadott sugárban, negatív értéknél összehúzódik
- az árnyék színe: a `color` tulajdonságnál megtanult módszerekkel
- az *inset* kulcsszó segítségével megváltoztatható a külső árnyék belső árnyékká

Példa: `#arnyekolt { box-shadow: 10px 0px 10px 2px #444444; }`

Megjegyzések:

- Több árnyék esetén azok előről hátrafelé haladva látszódnak, azaz az első van legfelül, a többi árnyék pedig alárétegződik.
- Az árnyék átnyúlhat más dobozokba vagy más dobozok árnyékaiba.
- Egy elem belső árnyékai az elem háttere fölé és a szegélye alá, a külső árnyékok az elem háttere alá kerülnek.

Néhány példa doboz-árnyék kialakítására:**box-shadow: 5px 10px:**

A div element with a shadow. The first value is the horizontal offset and the second value is the vertical offset. The shadow color will be inherited from the text color.

box-shadow: 5px 10px #888888:

You can also define the color of the shadow. Here the shadow color is grey.

box-shadow: 5px 10px red:

A red shadow.

*a külső árnyék a szegélyen kívül jelenik meg:***box-shadow: 5px 10px 18px #888888:**

More blurred.

*a belső árnyék a belső margó szélétől befelé vetődik:***box-shadow: 5px 10px 8px #888888 inset:**

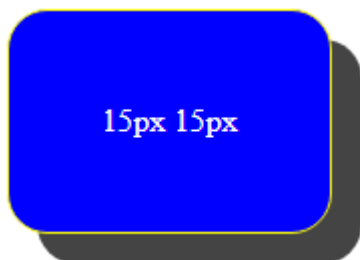
The optional third value adds a blur effect to the shadow.

ha a doboznak lekerekített szegélyei vannak, akkor az árnyék alakja is követi a lekerekítéseket:

More blurred and red.

További példák az értékek megadására:

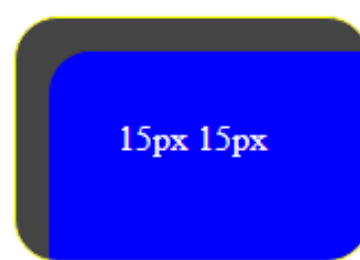
2 pozitív kiterjedési értékkel kifelé



15px 15px

```
#dob1_ki {
  box-shadow: 15px 15px #444; }
```

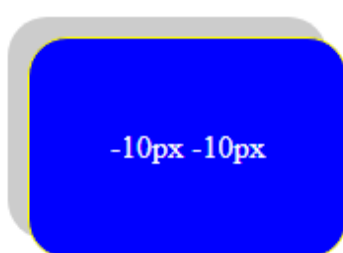
2 pozitív kiterjedési értékkel befelé



15px 15px

```
#dob1_be {
  box-shadow: 15px 15px #444 inset; }
```

2 negatív kiterjedési értékkel kifelé



-10px -10px

```
#dob2 {
  box-shadow: -10px -10px #ccc; }
```

3 pozitív kiterjedési értékkel kifelé



15px 15px 10px

```
#dob3 {
  box-shadow: 15px 15px 10px #0f0; }
```

4 pozitív kiterjedési értékkel kifelé



```
#dob4 {
  box-shadow: 15px 15px 10px 25px
             #123456; }
```

4 vegyes kiterjedési értékkel kifelé



```
#dob4_neg {
  box-shadow: -8px -8px 10px 5px
             #f0f; }
```

Lehetséges egyidejűleg több árnyékot is megadni, eltérő kiterjedési és életlenítési értékekkel és színekkel:



```
#dob_sokszinu {
  box-shadow:
    -15px -15px 8px red,
    10px 10px 3px yellow,
    20px -20px 6px green,
    -25px 20px 10px silver;
}
```

Ha egy elemre egyidejűleg több árnyékot is meghatározunk, akkor azok előlről hátrafelé haladva látszódnak majd: az első van legfelül és a többiek „alárétegződnek”.

Az árnyékok átnyúlhatnak más dobozokba vagy más dobozok árnyékaiba is. Egy elem belső árnyékai az elem háttere fölé és a szegélyes alá, a külső árnyékai az elem háttere alá kerülnek.

A doboz-árnyék legelterjedtebb alkalmazási területe a gombok formázása: a weboldalainkon szereplő űrlapjaink vagy más dobozaink egyedi alakítását is szolgálhatják (ld. a jobb oldali mintaképet).



VII. Blokkszintű elemek elhelyezése

A böngészők egy HTML-dokumentum olvasásakor balról jobbra és fentről lefelé haladnak, ez a normál szövegfolyam. A HTML-elemek alapértelmezetten a kódolási sorrendnek és a normál szövegfolyamnak megfelelően vagy sorban, vagy blokkszerűen egymás alatt jelennek meg – a dobozmodell alapján meghatározott részek figyelembevételével. Ebből a szerkezeti sémából egy adott elem az ebben a részben bemutatott tulajdonságokkal ragadható ki és pozicionálható más szabályok szerint (`position`, `z-index`, `opacity`, `float`, `overflow`, `visibility`, `display`).

A) Helyzetmegadás viszonyítási pontokhoz (*position* és kapcsolódó jellemzők)

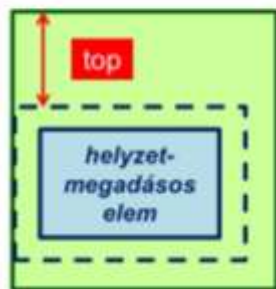
Helyzetmegadással az objektumok weboldalon elfoglalt pozícióját határozzuk meg úgy, hogy különböző viszonyítási pontokhoz képest adjuk meg az elem új helyzetét – ezt a viszonyítási pontot a `position` tulajdonsággal szabályozhatjuk. Az adott elem új helyét a `top` (felül), a `right` (jobbra), a `bottom` (alul) és a `left` (balra) tulajdonságok értékeivel definiáljuk, amelyek értékét képpontban (px), em-ben vagy %-os értékben adhatjuk meg. (Az egyes helymeghatározásra alkalmazott értékekre vonatkozó általános tudnivalókat a következő táblázatban foglaltuk össze.)

Az eltolás mértékére használt CSS-jellemzők áttekintése

felső eltolás

kijelölő { top: érték; }

a helyzetmegadási elemek *margójának* felső külső szélé és tárolótömbjének felső szélé közötti távolságot adhatjuk meg



jobb eltolás

kijelölő { right: érték; }

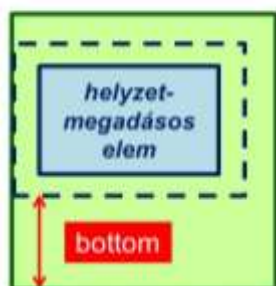
a helyzetmegadási elemek *margójának* jobb külső szélé és tárolótömbjének jobb szélé közötti távolságot adhatjuk meg



alsó eltolás

kijelölő { bottom: érték; }

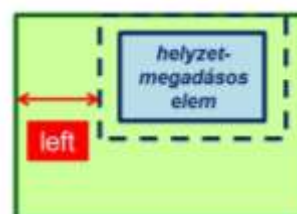
a helyzetmegadási elemek *margójának* alsó külső szélé és tárolótömbjének alsó szélé közötti távolságot adhatjuk meg



left eltolás

kijelölő { left: érték; }

a helyzetmegadási elemek *margójának* bal külső szélé és tárolótömbjének bal szélé közötti távolságot adhatjuk meg



A helyzetmegadás módja lehet abszolút, viszonyított, rögzített vagy statikus. (Tegyük fel, hogy van egy `id="nev"`-el azonosított objektumunk, amelyet szeretnénk elhelyezni – a példákban ezzel mutatjuk be a pozíciómeghatározást a különböző esetekben.)

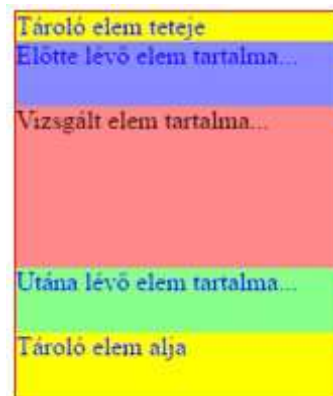
Elemek pozicionálásának lehetőségei a *position* tulajdonsággal

STATIKUS (alapértelmezett) pozíció

```
#nev { position: static; }
```

A statikus helyzetmegadás **a normál szövegfolyamban levő helyén hagyja az elemet** és akkor használjuk, ha az objektum más helyzetmegadást örökölt és vissza akarjuk helyezni a normál szövegfolyambeli helyére. Ebben az esetben a pozíciók (top, right, bottom és left) megadásának nincs hatása.

Az ily módon definiált elemek helyzetét a kódban definiált (eredeti) helye határozza meg.



RELATÍV (önmagához viszonyított) pozíció

```
#nev { position: relative;
      top: 30px;
      left: 20px; }
```

A relatív (vagy viszonyított) helyzetmegadás **az elemet a normál szövegfolyami helyéhez képest mozdítja el a kért mértékben és nem veszi ki azt a normál szövegfolyamból** (azaz a normál szövegfolyamban utána következő objektumok változatlan helyen fognak megjelenni – vagyis az eltoló elem eredeti helye üresen marad, tehát az eltolás az elem környezetére nincs hatással). Az így elmozdított HTML-elemek már kilóghatnak vagy akár teljes egészükben kikerülhetnek az őket tartalmazó tárolóelemük területéről és letakarhatnak más elemeket is akár.



Elemek pozicionálásának lehetőségei a position tulajdonsággal

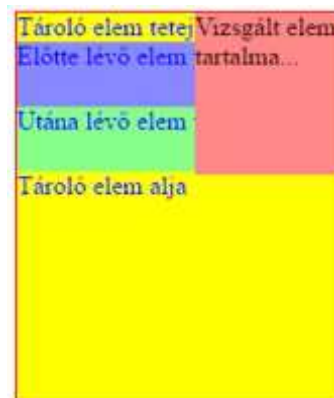
ABSZOLÚT (szülőelemhez viszonyított) pozíció

```
#nev { position: absolute;
      right: 0;
      bottom: 0; }
```

Abszolút helyzetmegadás esetén az elhelyezendő objektumot közvetlenül beágyazó objektum (azaz a szülőobjektum) bal felső sarkát tekintjük a koordináta-rendszer kiinduló pontjának, azaz (0,0) pozíciónak és azt adjuk meg, hogy az elhelyezendő objektum ahhoz képest lefelé, illetve jobbra milyen távolságra helyezkedjen el. Negatív érték esetén a beágyazó objektum bal felső sarkához képest balra, illetve felfele pozicionálunk.

Ha a szülőelem nem a kezdeti tárolóelem (azaz a body), akkor a szülőelemet is pozicionálni kell (lehetőség szerint relatívként).

Az abszolút helyzetmegadás esetén az elem pozíciója teljesen független a szövegfolyamtól (az eredeti helyétől), ezért az elem eredeti helyét felszabadítja a böngésző, „odacsúsztatva” a következő elemet.



FIXED (rögzített) pozíció

```
#nev { position: fixed;
      top: 10px;
      right: 10px; }
```

A rögzített (vagy nézetablakhoz viszonyított) pozíció esetén az elem mindig ugyanazon a helyen van, még akkor is, ha az oldal elgördítjük. Ezáltal az elem a böngészőablak látható részéhez rögzített és így mindig megjelenített vagyis látható marad. Ebben az esetben tehát maga a képernyő a viszonyítási pont.



STICKY (tapadós) pozíció

```
#nev { position: sticky;
      top: 0; }
```

A tapadós (vagy ragadós) pozicionálás esetén az adott elem a felhasználó görgetési pozíciója alapján kerül elhelyezésre.

A tapadós elem a görgetési pozíciótól függően vált a relatív és a rögzített között: relatív helyzetben van mindaddig, amíg az aktuális helyzete meg nem felel a meghatározott helyének, tehát ha elérte az, ott „megragad”.

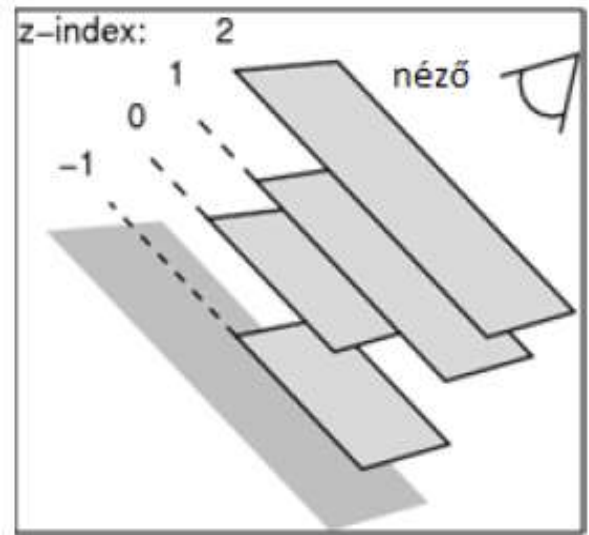


Megjegyzés: Csak az absolute elhelyezésű elemekre alkalmazható a **clip** (levágás) tulajdonság, amellyel az definiálható, hogy az adott elem szegélydobozának mekkora része legyen látható. Lehetőséges értékei: **auto** (nincs levágás), **shape** (alak – megadható a látható rész négyzet alakja az elem bal felső sarkától számítva px-ben, felsorolásszerűen, az óramutató járásával egyező irányban , -vel).

B) Átfedési sorrend (z-index)

Az elemek eredeti pozíciójának megváltoztatásakor azok olyan helyzetekbe kerülhetnek, hogy azok egymást részben vagy egészében átfedhetik. **Az egymást takaró elemek láthatóságát alapértelmezetten a HTML-dokumentumban történt definiálás sorrendje határozza meg, azaz a kódban később megadott elem takarja a kódban korábban definiált elem rétegét.**

Az z-index tulajdonság határozza meg az elemek halmozási sorrendjét, vagyis azt, hogy melyik elemet kell elhelyezni egy másik elem elé vagy mögé. (Nevét az elem síkjára merőleges z-tengelyről kapta a nevét. Az x-tengely a vízszintes, az y-tengely a függőleges pozíciót definiálja egy elem síkjában, az ezekre merőleges virtuális z-tengely a függőleges pozíciót, vagyis a térbeliséget szimbolizálja.)



Fontos tudnivaló: a z-index tulajdonság értékét csak akkor veszi figyelembe a böngésző, ha az elemre pozicionálási jellemző (azaz position tulajdonság) is megadásra került!

A z-index lehetséges értékei:

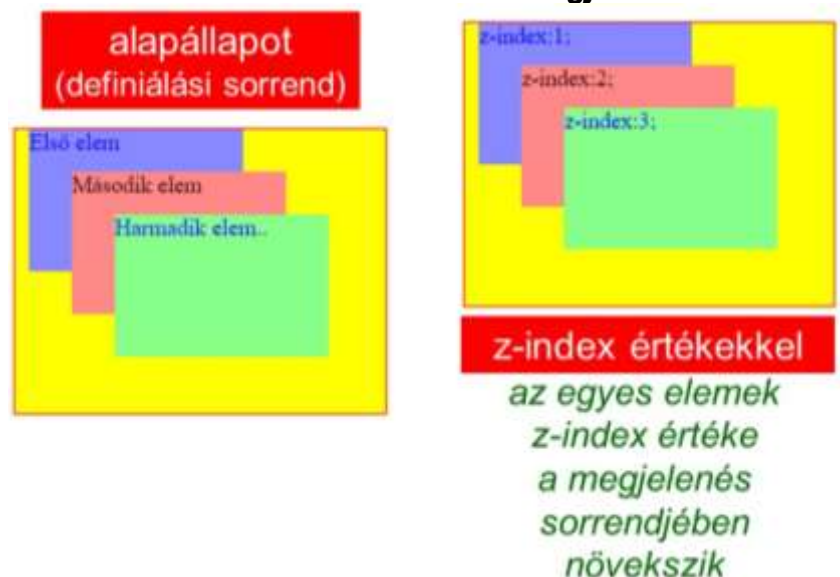
- tetszőleges előjelű egész szám (lehet 0 is és negatív is),
- bárhonnan indulhat a számozás,
- nem kell egymást követő számoknak lenniük.

A tartalom láthatóságának megjelenítése a számok növekvő sorrendjébe rendeződik át: a legkisebb értékű kerül legalulra (azaz a nézőtől a legtávolabbra), a legnagyobb indexű pedig legfelülre (azaz a nézőhöz közelebb).

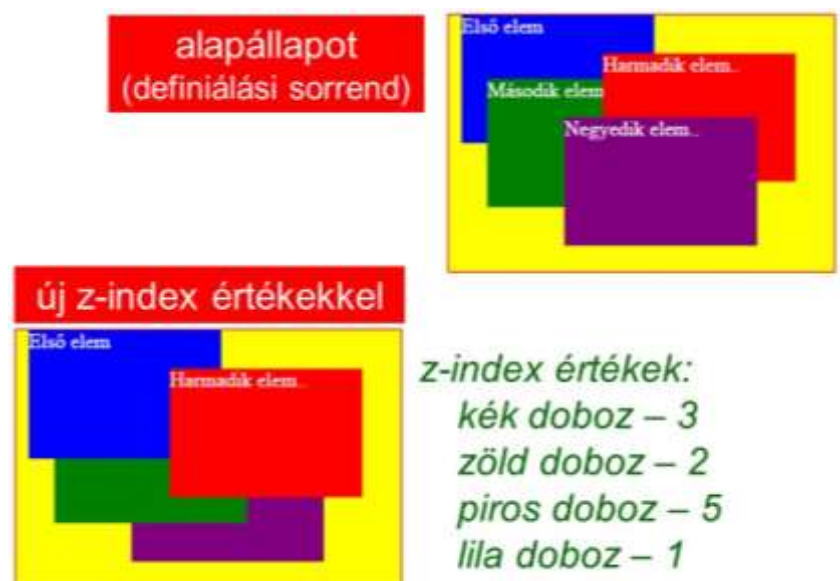
Hogyan lehet dinamikusan változtatni az elemek z-index értékét? Megoldás: ha az elemhez beállítjuk a :hover állományhoz egy nagyon magas vagy nagyon alacsony számértéket, akkor az elemre történő rámutatáskor (azaz a „hover” állapot elérésekor) az adott elemre érvényesíteni fogja a böngésző a beállítást. Így a nagyon nagy indexű elem legfelülre, a nagyon kicsi indexű elem pedig legalul fog megjeleníteni.

Mivel lehet biztosítani, hogy egy elem biztosan legalulra vagy legfelülre kerüljön? Megoldás: válasszunk az elemhez nagyon kicsi negatív értéket (pl. -1000) vagy nagyon nagy pozitív számot (pl. 5000), ez biztosítani fogja, hogy az elem legalul vagy legfelül fog megjeleníteni a többi elemhez képest.

definiális sorrend szerinti megjelenítés:



a definiált sorrendtől eltérő megjelenítés beállítása:



C) Átlátszóság (opacity)

Az oldalakon elhelyezett objektumokat (jellemzően képeket, háttereket) az általunk kívánt mértékben átlátszóvá is tudjuk tenni. Ennek akkor lehet jelentősége, ha az elemek részben vagy egészben takarják egymást. **Az átlátszóság tulajdonsággal azt határozzuk meg, hogy a felső elem alatt az alsó elem(ek) mennyire látszódik (látszódnak) át.**

Szabványosan az `opacity` stílustulajdonságnak adhatunk meg egy 0 és 1 közötti értéket azzal a jelentéssel, hogy minél nagyobb értéket adunk meg, az objektum annál kevésbé lesz átlátszó (tehát 0 esetén teljesen átlátszó, 1 esetén egyáltalán nem). (Nem szabványosan (IE-ben) az átlátszóságot a következőképpen is megadhatjuk: `filter: alpha(opacity=10)`; , ahol az `opacity` értéke 0 és 100 közötti egész szám és az érték növelése itt is csökkenti az átlátszóságot (tehát nem átlátszó itt a 100-as értéknél lesz).

Fontos tudni, hogy ha átlátszóságot adunk egy elem háttéréhez ezzel a tulajdonsággal, akkor minden gyermekelem átlátszóvá válik. Ha nem szeretne egy gyermekelemhez átlátszóságot adni, de mégis halványítani szeretné a háttérszín, akkor érdemes a háttérszín megadásához az `rgba()` függvényt alkalmazni. (Az RGBA-típusú színmegadás az RGB-típus kiterjesztett változata egy alfa csatornával, amely meghatározza a szín átlátszóságát: `rgba(piros,zöld,kék,alfa)`, amelyben alfa 0 értéke a teljesen átlátszó, 1 az egyáltalán nem átlátszó tulajdonságot jelöli. Az alfa értéke egész vagy törtszám lehet, 0 és 1 között mozoghat.

Felhasználása: a szöveget egy teljesen átlátszó elemen belül könnyen olvashatóvá lehet tenni vagy egy színes „filtert” helyezhetünk egy képre (ld. az alábbi minta első sorában lévő két példát).



`rgba(255, 99, 71, 0)`

`rgba(255, 99, 71, 0.2)`

`rgba(255, 99, 71, 0.4)`

`rgba(255, 99, 71, 0.6)`

`rgba(255, 99, 71, 0.8)`

`rgba(255, 99, 71, 1)`

D) Úsztatás és eltávolítás (float és clear)

További helyzetmegadási lehetőség az úsztatás, amellyel azt lehet előírni, hogy a többi tartalom körülfolyhassa az úsztatott objektumot.

A `float` tulajdonsággal vízszintes irányban a befoglaló elem széleihez tolhatók az elemek. Az inline elemeknek is blokk-dobozt hoz létre, a szomszédos elemekhez képesti távolságot pedig a margin megfelelő oldali beállításával szabályozhatjuk.

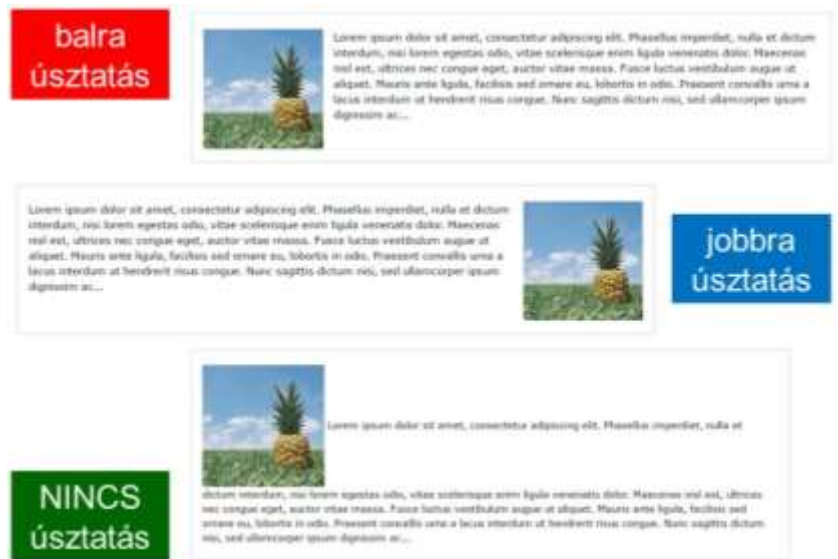
More Spectacular Yosemite Views

El Capitan is a 3,000-foot (910 m) vertical rock formation in Yosemite National Park, located on the north side of Yosemite Valley, near its western end. The granite monolith is one of the world's favorite challenges for rock climbers. The formation was named "El Capitan" by the Maripasa Battalion when it explored the valley in 1851.

Tunnel View is a viewpoint on State Route 41 located directly east of the Wawona Tunnel as one enters Yosemite Valley from the south. The view looks east into Yosemite Valley including the southwest face of El Capitan, Half Dome, and Bridalveil Falls. This is, to many, the first views of the popular attractions in Yosemite.

Az úsztatás a `float` jellemző **left**, **right**, illetve **none** és **inherit** értékével adható meg, amelyek esetén az objektum balra, jobbra, illetve egyáltalán nem lesz úsztatva, tehát azt jobbról, balról vagy nem folyja körbe a többi tartalom, további lehetőség még az **inherit** érték, vagyis a szülő elemtől örökölt beállítás alkalmazása). A következő ábrán a két jellemző úsztatási érték (left és right) beállításának eredményére láthat mintapéldát.

Megjegyzések: A `float` tulajdonság alapfunkciója a képek vagy táblázatok oldalra úsztatása, de bármilyen más elemre is alkalmazható. Így például `<div>` tagok segítségével lehetséges a weblapok elrendezésének definiálása is, viszont figyelniünk kell, hogy az úsztatás természetéből adódóan nagyon eltérő eredményeket kaphatunk a különböző szélességű monitorokon. További alkalmazási lehetőség az iniciálék kialakításánál lehet, amikor a `::first-letter` pszeudo-elemhez mint szelektorhoz állítunk be úsztatási jellemzőket (és betűtulajdonságokat, pl. nagyobb betűméretet, más -típust, vastagítást, döntést stb.).



Fontos tudni, hogy az úsztatott elemet követő elemek is „úszni” fognak mindaddig, amíg csak az lehetséges! Ezt a helyzetet a `clear` tulajdonsággal lehet megszüntetni.

Amíg a `float` tulajdonság oldalra úsztatja a képet (vagy más objektumot) a rendelkezésre álló helyen belül, addig **a `clear` tulajdonság azt adja meg, hogy a kép (vagy más blokk szintű doboz) melyik oldala nem kerülhet egy korábbi úsztatott elem mellé.** Lehetséges értékei:

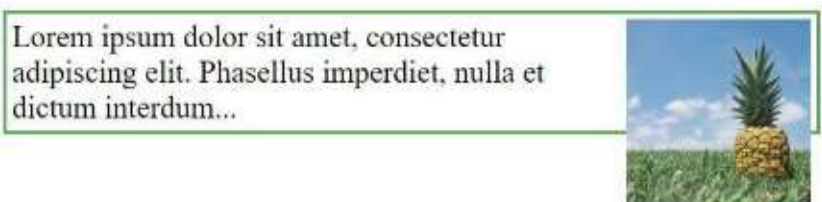
- **left:** az elemdoboz bal oldala nem kerülhet egy korábbi úsztatott elem mellé
- **right:** az elemdoboz jobb oldala nem kerülhet egy korábbi úsztatott elem mellé
- **both:** az elemdoboz egyik oldala sem kerülhet egy korábbi úsztatott elem mellé
- **none:** az elemdoboz nincs eltávolítva a korábbi úsztatott elemek mellől
- **inherit:** az érték a szülő elemtől örökölje

Több esetben előfordulhat, hogy egy úsztatott elem magasabb, mint az elemet tartalmazó elem és ekkor a beágyazott elem „túlcsordul” a szülő-elemen kívülre (ld. az oldalsó ábrát). Ebben az esetben a **clearfix** technikát alkalmazzuk, azaz a túlfolyás szabályozására az **overflow** tulajdonságot állítjuk be az **auto** értékkel. (További részleteket a következő pontban olvashat az **overflow** használatáról és az alkalmazható tulajdonságokról.)

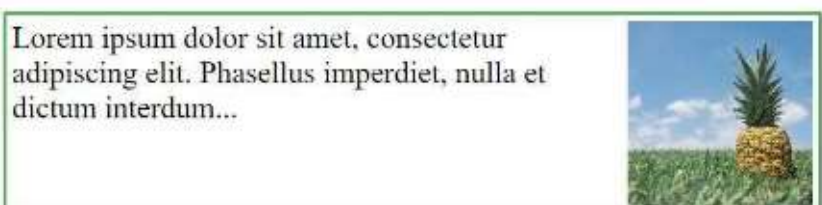
Az **overflow: auto;** clearfix mindaddig jól működik, amíg kézben tudja tartani a margókat és a kitöltést (egyébként görgetősávok jelenhetnek meg). Az új, modern **clearfix hack** használata azonban biztonságosabb, és a legtöbb weboldal a következő kódot használja:

```
azonosito::after { clear: both;
content: "";
display: ...; }
```

a beágyazott és úsztatott elem túlnyúlik a szülőelemen:



a túlcsordulás helyes kezelését követő állapot:



E) Túlcsordulások, túlnyúlások kezelése (overflow)

Az **overflow** (túlnyúlás vagy túlcsordulás) összevont tulajdonság specifikálja azt, hogy milyen legyen a megjelenítés, ha egy tartalom túlnyúlik a befoglaló doboznak a tartalom számára rendelkezésre álló területén. Mind blokkszintű, mint sorok belüli elemnél alkalmazható, de csak azoknál hoz érdemi változást, amelyeknél magassági értéke határoztunk meg!

A kilógó területek kezelésének módját az egyes irányokra is meghatározhatjuk külön-külön:

- **overflow-x**: vízszintes irányú túlnyúlás kezelése
- **overflow-y**: függőleges irányú túlnyúlás kezelése

Az **overflow** a kettő összevont alakja, amely egyszerre kezeli mindkét irányú túlnyúlást. Ha itt csak egy érték van megadva, akkor az mindkét irányra vonatkozik, egyébként az első érték az x, a második pedig az y irányú túlnyúlást szabályozza.

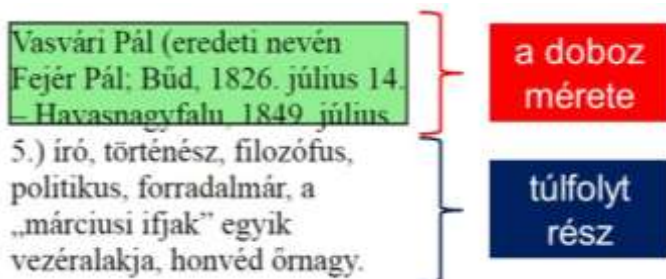
Ha gördítősáv helyeződik az elem széléhez, akkor az a belső szegély és a kitöltés külső éle közé illeszkedik be, így a gördítősáv által elfoglalt hely is befolyásolja a méretek kiszámítását, mert takarni fogja a tartalom egy részét!

Lehetséges értékei:

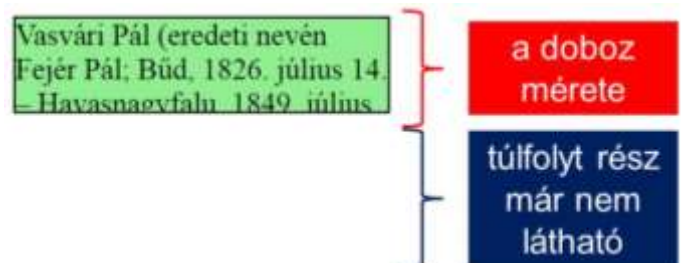
- **visible** (alapértelmezett): a beágyazott tartalom akár befér a befoglaló doboz content-box részébe, akár nem, mindig teljes egészében megjelenik – tehát az esetlegesen az el nem férő rész túlcsordul, de ez a túlcsordulás ne kerül levágásra, hanem a beágyazó elem dobozán kívül jelenik meg
- **hidden**: a beágyazott tartalomnak csak a befoglaló doboz content-box részébe eső darabja jelenik meg, az azon kívül eső rész nem látszik, azaz a túlcsordult rész le van vágva
- **scroll**: a tartalomnak a befoglaló doboz content-box részébe eső darabja és a böngésző által alkalmazott görgető mechanizmus jelenik meg, amellyel a tartalom kétoldali gördítősávval görgethetővé válik
- **auto**: viselkedése böngészőfüggő, de valamilyen görgetőmechanizmust hoz létre a túlcsordulásra, csak arra az oldalra helyez gördítősávot, ahová az szükséges a doboz mérete alapján

A túlcsordulás kezelése lehetséges eseteit bemutató példák:

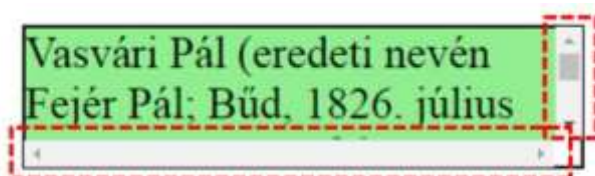
overflow: visible;



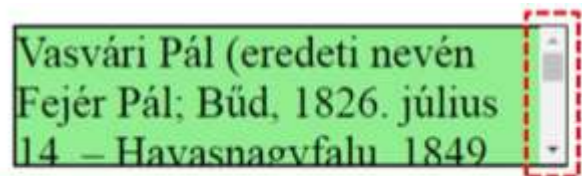
overflow: hidden;



overflow: scroll;



overflow: auto;



További két lehetséges érték is használható:

- **no-display**: ha a tartalom nem illeszkedik a content-box-ba, az egész doboz eltávolításra kerül, mintha a **display: none**; beállítást definiáltuk volna
- **no-content**: ha a tartalom nem illeszkedik a content-box-ba, az egész tartalom eltávolításra kerül, mintha a **visibility: hidden**; beállítást definiáltuk volna

Az **overflow-style** tulajdonsággal (amelyet nem támogatnak a nagyobb böngészők, ezért gyakran animációval (ld. később) helyettesítjük) még azt is befolyásolhatjuk, hogy egy vagy több preferált görgetőmechanizmus legyen-e. Értékeit preferencia-sorrendben, azaz listaként felsorolva adjuk meg, a böngésző pedig a listából az elsőt fogja alkalmazni, amelyet támogat. Ha egyiket sem támogatja, akkor úgy viselkedik mintha az **auto** értéket állítottuk volna be. A tulajdonság lehetséges értékei:

- **auto** (alapértelmezett beállítás): nincs preferencia, tehát a **scroll** érték hatása érvényesül
- **scrollbar**: keskeny csík, amelyet az elem egyik vagy mindkét oldalához illesztenek be, gyakran rákattintható nyilakkal és egy vonszoló csúszkával lehet az elem tartalmát fel-le vagy jobbra-balra vonszolni
- **panner**: az elem egyik sarkában látszódó négyszög, benne egy kisebb négyszöggel – a nagyobb négyszög képviseli az elem teljes tartalmát, a kisebb annak a látható részét, a kisebb négyszög mozgathatja a felhasználó, hogy ennek megfelelően mozgass az elem tartalmát
- **move**: a felhasználó közvetlenül tudja mozgatni a tartalmat, tipikusan az egérkurzor kézzé vagy négy-nyilas keresztte változik, ezzel jelzi, hogy a felhasználó az egérrel vonszolhatja a tartalmat
- **marquee**: a tartalom autonóm módon, a felhasználó közreműködése vagy kontrollja nélkül mozog, azaz ha a felhasználó elég sokat vár, az egész túlnyúló tartalmat látni fogja – a mozgás iránya, jellege, sebessége, az ismétlődések száma külön tulajdonságokkal definiálható:
 - **marquee-style**: a mozgás jellege (egy elem teljes tartalma egy darabban mozog, ha pl. egy elem két szövegsorból áll, akkor mindkettő sor blokkszerűen együttesen halad)
 - **scroll**: az egyik oldalról indul és eltűnik a másik oldalon
 - **slide**: a jobb oldalról elindul, a bal oldalon megáll, majd visszaugrik a jobb oldalra
 - **alternate**: oda-vissza mozog
 - **marquee-play-count**: a mozgatás száma
 - tetszőleges pozitív **egész** szám
 - **infinite**: végtelen (16 mozgás után leáll a mozgatás)
 - **marquee-direction**: a mozgatás iránya
 - **forward**: előre, azaz úgy mozog a szöveg, hogy az eredetileg elrejtett rész a normál olvasási iránynak megfelelően jelenjen meg
 - **reverse**: visszafelé (fordított irányban)
 - **marquee-speed**: a mozgatás sebessége (tényleges értékei az alkalmazott böngészőtől és a mozgatott tartalom típusától is függenek), értékei: **slow**: lassú, **normál**: sem gyors, sem lassú, **fast**: gyors

F) Láthatóság (visibility)

A **visibility** (láthatóság) tulajdonság egy adott elem két lehetséges állapotát jelöli, amely megmutatja vagy láthatatlanná teszi azt; a láthatatlan elem a weblap elhelyezésében megtartja a helyét, csak átlátszóvá válik.

látható elem

The visibility Property

This heading is visible

This heading is visible

Notice that the hidden heading still takes up space on the page.

elrejtett elem

The visibility Property

This heading is visible

This heading is visible

Notice that the hidden heading still takes up space on the page.

Általában ugyanazon a helyen egymásra helyezett elemek láthatóságának a látogató valamilyen tevékenysége, akciója révén kiváltott változtatásokra használják (pl. csak az egérkurzossal való rámutatáskor jelenik meg az adott elem), de a weblap kialakítása során egyébként is hasznos lehet egyes elemek időleges eltakarása.

Lehetséges értékei:

- **visible** (alapértelmezett érték): az elem látható
- **hidden**: az elem nem látható, de a helye megmarad
- **collapse** (csak táblázatoknál): sorok, oszlopok, sorcsoportok, oszlopcsoportok elrejtéséhez
- **inherit**: szülőtől örökölt érték

G) Megjelenítés (display)

A display tulajdonság határozza meg az elemhez generált doboz típusát és ezen keresztül előírhatjuk az adott elem megjelenítésének módját is. Nagyon sokféle lehetséges értéke van, amelyek közül a legjellemzőbbeket találja majd ebben a részben.

Minden HTML elemnek alapértelmezett megjelenítési értéke van attól függően, hogy milyen típusú elem van; a legtöbb elem alapértelmezett megjelenítési értéke blokk vagy inline. Amíg **blokk szintű** elemek mindig új sorban indulnak és alaphelyzetben a szülőelem teljes szélességét felveszik (pl. `<div>`, ``, `<h1>`, `<form>`, `<header>`), addig az inline elemek nem kezdőnek új sorban és csak akkor szélességet vesznek fel, amennyit a tartalmuk igényel (pl. `<a>`, ``, ``). **FONTOS** tudni azonban, hogy egy elem kijelzési tulajdonságának, azaz a display értékének beállítása csak az elem megjelenítésének módját változtatja meg, NEM pedig azt, hogy az milyen típusú elem!

A leggyakrabban használt értékek (hatásukat a lista utáni ábrában mutatjuk be):

- **inline** (alapértelmezett érték): szövegek közötti vagy folytonos elrendezésű
- **block** (blokk-doboz): blokk szintű elem
- **inline-block**: blokk-dobozként formázható, de inline dobozként kerül elhelyezésre az oldalon
- **none**: az elemhez nem rendel sorközi vagy blokk szintű dobozt, azaz az elemnek nincsen hatása a weboldal elrendezésére

The display Property**display: inline**

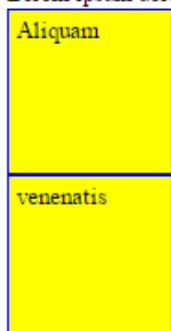
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. Aliquam venenatis gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

display: inline-block

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat. Aliquam venenatis gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

display: block

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat scelerisque elit sit amet consequat. Aliquam erat volutpat.



gravida nisl sit amet facilisis. Nullam cursus fermentum velit sed laoreet.

Alkalmazási területek:

- **inline:** lista elemeinek egymás melletti megjelenítése – tipikusan gombszerűen formázott vízszintes menüsorként
- **block:** nagy képek külön sorban történő megjelenítése
- **none:** elemre történő rámutatáskor vagy más feltétel teljesülésekor az elem tartalma és az általa foglalt hely is felszabadul (pl. egy gombra való rámutatáskor jelenik meg a segítség szövege, azaz tipikusan elemek elrejtéséhez használják azzal, hogy az elemet külön törölni kellene vagy újra létrehozni)

Blokkelemekből (pl. felsorolások) folytonos elrendezésű elem (pl. menü) kialakítása:

forrás

HTML-kód:

 kenyér
 tej
 vaj

cél

Scientific

| trial | trial |

541

```
li { display: inline;
      list-style-type: none;
      margin-right: 50px;
      padding: 1em;
      background-color: magenta;}
```

H) Hasznos tudnivalók, praktikák

Elemek és tartalmak vízszintes igazítása

1. Blokkelem vízszintes igazítása a szülőelemhez viszonyítva: `margin: auto;`

Az elem szélességének beállítása megakadályozza azt, hogy az elem a szülőelem széléig nyúljon, mert a rá meghatározott szélesség kisebb a szülőelemtől örökölt szélességnél, így a fennmaradó szélességi területet automatikusan szétosztja a böngésző a bal és a jobb oldalra.

[illegible]

A blokk (mint egység) szélessége meghatározott és a beágyazó elemben a margó beállításával középre igazított.

szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg
szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg
szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg
szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg szöveg

2. Blokkelem tartalmának vízszintes igazítása: `text-align: left|center|right|justify;`

3. Kép vízszintes igazítása:

```
display: block; margin-left: auto; margin-right: auto;
```

Elsőként a kép megjelenítését blokkszintűvé kell alakítani, majd ezt követően már értelmezhetővé válik a bal és a jobb margó automatikus kiszámítása.

To center an image, set left and right margin to auto, and make it into a block element.



Blokkelemek helyzetének meghatározása, struktúra kialakítása az oldalon

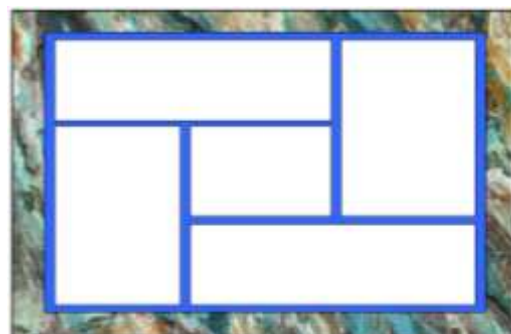
1. Pozíció meghatározásával:

```
position: ...;
left / right: ...;
top / bottom: ...;
```

Az abszolút pozícióval az elemeket kivehetjük a normál szövegfolyamból és a szülőhöz képest adhatjuk meg a helyzetét.

2. Úsztatás beállításával: `float: ...;`

Az elemeket egymás mellé csúsztathajtuk az úsztatással, de ekkor figyelni kell arra, hogy a beágyazó elem magassága haladja meg a beágyazott elem magasságát, különben a beágyazott elem ki fog lógni a szülőelem területéről. Ügyelni kell arra is, hogy az egymás mellé szánt blokkok szélességi értékeinek összege ne haladja meg a vízszintesen rendelkezésre álló szélességet, különben a struktúra szétesik, egyes (utolsó) blokkok a többi alá tördelődnek.



Elemek és tartalmak függőleges igazítása

1. Blokkelem tartalmának függőleges igazítása belső margókkal:

padding-top: ...; padding-bottom: ...;

Ha a dobozelem magassági méretét nem adjuk meg, akkor azonos alsó és felső margó alkalmazásával a doboz tartalma függőlegesen középre igazítható.

2. Blokkelem tartalmának függőleges igazítása sorközzel:

height: ...px; line-height: ...;

Ha a befoglaló elem magassága és a benne lévő szöveg sorközének értéke azonos, akkor a doboz tartalma (szövege) is függőlegesen középre igazodik. Csak rövid szövegeknél érdemes használni!

2. Blokkelem függőleges igazítása pozícióval és transzformációval:

position: absolute;;

top: 50%;

vagy bottom: 50%

transform: translateY(-50%);

transform: translateY(-50%);

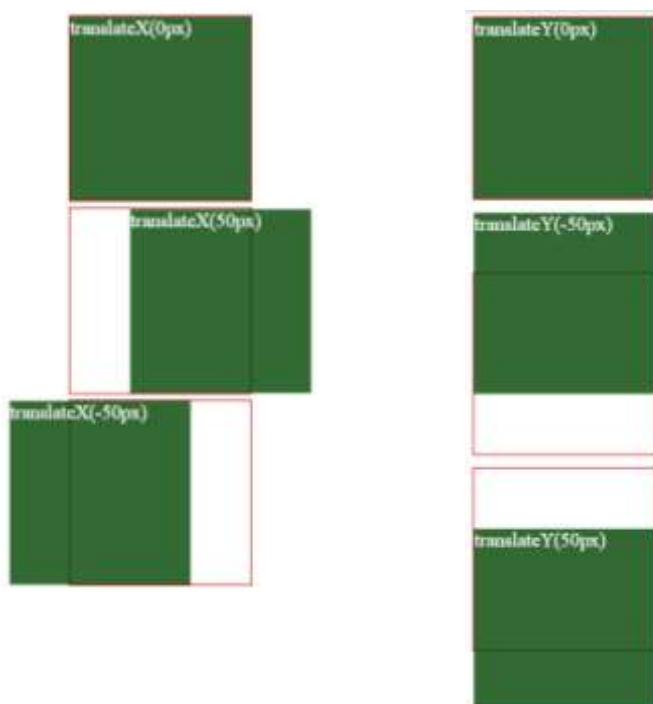
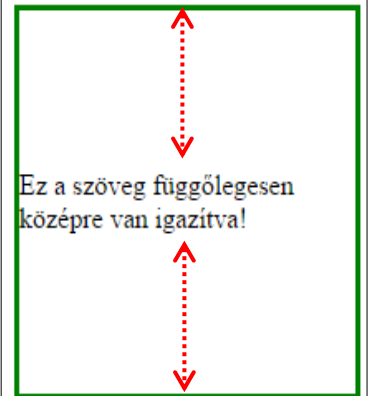
Ha blokkelemet szeretnénk a szülőelemen belül függőlegesen igazítani vagy az előző két módszer nem alkalmazható, akkor pozicionálással eltoljuk a beágyazó elem magasságának felével fentről lefelé (top) vagy alulról felfelé (bottom) a blokkot, majd a beágyazott elem magasság értékének felével korrigálunk (translateY).

A **transform** tulajdonság 2D vagy 3D átalakítást alkalmaz egy elemre, vagyis lehetővé teszi az elemek elforgatását, méretezését, mozgását, torzítását. A vízszintes és függőleges eltoláshoz alkalmazott alakjai:

- **transform: translateX(x);** - vízszintes (X tengely) irányú eltolást tesz lehetővé; segítségével egy blokkelemet pozitív érték esetén jobbra, negatív érték esetén balra tolhatunk
- **transform: translateY(y);** - függőleges (Y tengely) irányú eltolást tesz lehetővé; segítségével egy blokkelemet pozitív érték esetén lefelé, negatív érték esetén felfelé tolhatunk
- **transform: translate(x,y);** - egyszerre teszi lehetővé a vízszintes (X tengely) és a függőleges (Y) irányú eltolást (x értéke a vízszintes, y értéke pedig a függőleges eltolási értéket jelöli)

Az értékek megadásához alkalmazhatók az abszolút mértékegységek (pl. px), de gyakran relatív mértékegységeket (pl. %) használunk, mert ekkor a beágyazott (azaz a mozgatandó blokk) szélességi és/vagy magassági értéke százalékában viszonyított mértékben történik meg az eltolás.

In this example, we use the padding property to center the div element vertically:



In this example, we use positioning and the transform property to vertically and horizontally center the div element:



position: absolute;
top: 50%; left: 50%;
transform:
translate(-50%, -50%);

Blokkelemek vízszintes és függőleges igazítása meghatározott dobozméret esetén

Ha egy objektumot vízszintesen és függőlegesen is középre szeretnénk igazítani, akkor célszerű a doboznak konkrét méreteket adni, majd ezek értékét felhasználni a margók beállításához:

```
#fkozepre {
  width: szélesség;
  height: magasság;
  position: absolute;
  top: 50%;
  left: 50%;
  margin-top: -magasság/2;
  margin-left: -szélesség/2;
}
```

A tulajdonságok között azért szükséges negatív margókat megadni, mert az abszolút helyzetmegadás az objektum bal felső sarkát pozicionálja, de mi az objektum közepét akarjuk középre helyezni.

VIII. Képek formázása

A webkiszolgáló, a webböngésző és a végfelhasználó szempontjából **nincs jelentősége, hogy hol helyezzük el a képeket**, amennyiben mi magunk tudjuk, hol vannak és a HTML-kódban helyes elérési utakat adunk meg.



Ha képet szeretnénk elhelyezni egy weboldalon, akkor az első dolgunk az legyen, hogy a könnyebb átláthatóság végett a képet áthelyezzük a HTML-fájlt tartalmazó könyvtárba vagy egy külön mappába. A képek HTML-fájlba történő beszúrásával kapcsolatos ismeretokről és az tagról a HTML5 kapcsán már részletesen tanultunk.

A HTML5 arra is lehetőséget biztosít, hogy **a képeket és a hozzájuk kapcsolódó képfeliratokat összekapcsoljuk**; ehhez a <figure> és a <figcaption> páros HTML-tagokat használjuk. A <figure> használatával tárolóelembe foglalhatók a képek, diagramok, ábrák, listák és a hozzájuk tartozó címek, feliratok, magyarázatok, magyarázó szövegek, így együtt kezelhetők, pozicionálhatók, formázhatók az oldalunkon. Az alábbi példában egy formázott kép és képfelirat kódolása látható.

A weboldalon látható eredmény:



HTML-kód:

```
<figure>
  
  <figcaption>
    An elephant at sunset
  </figcaption>
</figure>
```

CSS-kód a formázáshoz:

```
figure {
  border: thin #c0c0c0 solid;
  display: flex;
  flex-flow: column;
  padding: 5px;
  max-width: 220px;
  margin: auto;
}

img {
  max-width: 220px;
  max-height: 150px;
}

figcaption {
  background-color: #222;
  color: #fff;
  font: italic smaller sans-serif;
  padding: 3px;
  text-align: center;
}
```

A képek formázásához nincsenek speciális CSS-tulajdonságok, hanem a már korábban megismert jellemzőket²⁴ alkalmazhatjuk:

- **reszponzívan viselkedő képekhez** (azaz amelyek mérete automatikusan a képernyő vagy böngészőablak méretéhez igazodik) **állítsunk be százalékos méretezést** – de ha nem akarjuk, hogy bármikor is nagyobb legyen az eredeti méreténél, akkor a szélességet kössük a szülő-elemhez és határozzuk meg a legnagyobb értékét (`max-width: 100%;`), illetve a magasságának kiszámítását bízzuk a böngészőre (`height: auto;`);
- **a képek vízszintesen középre igazításához** formázzuk a képet blokkszintű elemként (`display: block;`), illetve állítsuk be a bal és a jobb margó nagyságát is automatikusra (`margin-left: auto; margin-right: auto;`)
- **a képek sarkait a `border-radius` tulajdonsággal kerekíthetjük**, akár ellipszis vagy szabálytalan alakú területen is megjeleníthetjük a képeket;
- **létrehozhatunk polaroid megjelenést** (ld. az oldalsó ábrát) azzal, ha a képet és a feliratát egyetlen dobozban jelenítjük meg a megfelelő távolságbeállításokkal és még dobozárnyéket is meghatározunk hozzá;
- előfordulhat az is, hogy egy szöveget tartalmazó sor közepére szeretnénk egy kicsi képet beszúrni vagy egyetlen sor szöveget szeretnénk egy ábra mellett feliratként elhelyezni – ekkor alkalmazandó a `vertical-align` tulajdonság²⁵, amellyel a szöveg és a kép függőleges helyzetét szabhatjuk meg;
- a `padding` jellemző és a `:hover` álosztály segítségével ún. **indexképeket alakíthatunk ki**, amikor a képre történő rámutatáskor a beállított szegélytől eltérő színű dobozárnyék jelenik meg a kép körül;
- a képekhez meghatározott **átlátszósági érték** (`opacity`) **segítségével különleges hatást érhetünk el**, amelyet kombinálhatunk a `:hover` álosztállyal, hogy a képre történő rámutatáskor a kép átlátszósága megváltozzon;
- a **szűrő** tulajdonság (`filter`) alkalmazásával **elmosódást és telítettséget adhatunk képeinknek**;
- **a képekre szövegeket is helyezhetünk**, ha a képet és a szöveget egy közös szülődobozban definiáljuk, majd a szülőt relatív, a beágyazott elemeket abszolút módon formázzuk, majd a `top/bottom` és a `left/right` tulajdonságoknak értéket adunk;
- **dinamizmust is adhatunk a weboldalon elhelyezett képhez**, ha a képre történő rámutatáskor a képhez kapcsolódó másik, de alaphelyzetben elrejtett elemet (pl. szöveget vagy egy másik képet) **valamilyen effektus alkalmazásával teszünk láthatóvá**: pl. valamelyik irányból rányílik a képre a felirata – ehhez a `transition` tulajdonságot tudjuk felhasználni – vagy ha a képet tükrözzük a `transform` tulajdonság `scaleX(...)` függvényével;
- **több képből reszponzív képgalériát hozhatunk létre**, amelyet a **médialekérdezések** vagy valamilyen CSS-keretrendszer (pl. a Bootstrap) alkalmazásával kódolhatunk, hogy a képek a különböző méretű képernyőkön eltérő elrendezésben (oszlopszámban) jelenjenek meg;
- haladó lehetőség **az `image modal`, amely a CSS és a JavaScript együttműködésére** ad jó példát: nemcsak a képre történő rámutatáskor, hanem az arra való kattintáskor vagy egyéb művelet végzésekor különböző tevékenységeket végez (pl. kattintáskor kiemeli a képet).



²⁴ https://www.w3schools.com/css/css3_images.asp

²⁵ A `vertical-align` a soron (azaz a sorközi dobozoknak a sordobozon) belüli függőleges irányú elhelyezkedést definiálja. Lehetséges értékei: `top` (fent), `middle` (középen), `bottom` (lent), `super` (felső index), `sub` (alsó index), `text-top` (szöveg tetején), `text-bottom` (szöveg alján), `baseline` (betűvonalhoz).

IX. Hivatkozások formázása

A weboldalak legfontosabb elemei az olyan szövegek, képek vagy más objektumok, amelyekre kattintva átugorhatunk egy másik internetes helyre. Ezeket nevezzük élőkapcsoknak, linkeknek vagy hivatkozásoknak, amelyeket az `<a>` HTML-taggal hoztunk létre és tulajdonságaként a cél URL-jét a href értékeként kell megadnunk. Alapesetben a hivatkozásokat megtestesítő szövegrészeket a böngészők kék színnel és (solid) aláhúzással emelik ki, de ezt bármikor megváltoztathatjuk.



A hivatkozások többfélék lehetnek: mutathatnak egy mások által készített (külső) webes tartalomra, saját meghajtónkon elhelyezett oldalunkra, az oldalon belül, kezdeményezhetjük vele elektronikus levél írását, biztosíthatjuk segítségével a képek nagyobb méretű és jobb felbontású változatának megjelenítését (ld. a HTML5 jegyzet megfelelő fejezetét).

A) Állapot meghatározása

A hivatkozásoknak négy fő állapota lehet:

- a felhasználó még nem kereste fel a linket²⁶ (link állapot),
- a felhasználó már felkereste a linket²⁷ (visited állapot),
- az egér a link felett van, azaz a felhasználó rápozicionálta az egérkurzort (hover állapot)²⁸,
- a link használatban van, azaz a hivatkozott oldal betöltése folyamatban van (active állapot).

Ha csak az `a { ... }` alakú CSS-kódsort alkalmaznánk a formai tulajdonságok megadására, akkor a hivatkozások formai megjelenítése a linkek minden állapotában ugyanolyan lenne – ha viszont e 4 állapot egyikét (vagy többet) is megadjuk a kijelölés során, akkor pontosítható a linkek megjelenítése.

Az állapot szerinti formázásokra a CSS álosztály-kijelölőket használ (ld. jelen jegyzet pszeudo-osztályok pontja dinamikus pszeudo-osztályok és elemeik alpontját). **Az álosztály olyan „osztály”, amely az elemek stílusát bizonyos körülmények között írja le, amelyek megfelelhetnek például a felhasználó műveleteinek.**

Mivel hivatkozás állapota nem szerepel a HTML-kódban, hanem a felhasználó tevékenységétől függ, ezért a hivatkozások formázására ezeket az álosztály-kijelölőket használjuk, amelyeket a fenti állapotokra rendre a következőképpen kell megadni:

- a felhasználó még nem kereste fel a linket - `a:link`
- a felhasználó már felkereste a linket - `a:visited`
- az egér a link felett van, azaz a felhasználó rápozicionálta az egérkurzort - `a:hover`
- a link használatban van, azaz a hivatkozott oldal betöltése folyamatban van - `a:active`

Fontos a definiálási sorrend, azaz a `:link` és a `:visited` után lehet a `:hover`, és végül az `:active`!

B) Gyakran alkalmazott CSS-tulajdonságok

Ezeket az álosztály-kijelölőket más kijelölőkhöz hasonlóan használhatjuk továbbiakkal együtt és segítségükkel a hivatkozásokat a 4 állapotra vonatkozóan közösen, de külön-külön is formázhatjuk.

A leggyakrabban **gombszerűvé alakítjuk a linket**, ezért szükség lehet arra, hogy

- körbekeretezzük szegélyezéssel (`border`) és eltávolítsuk a szegélytől a feliratot a belső margó, azaz a bélés (`padding`) beállításával;
- változtassunk a megjelenítés színein: a gomb kitöltési színén (háttérszín: `background-color`) és a gombon megjelenő szöveg színén (betűszín: `color`);
- eltüntessük az aláhúzást (a `text-decoration` tulajdonságnak a `none` értéket adjuk).

²⁶ Ennek az állapotnak a teszteléséhez szükség lehet arra, hogy az oldal újból nem felkeresettnek minősüljön, ehhez azt törölni kell azt az adott böngészőben a böngészési előzmények közül.

²⁷ A böngésző azokat az oldalakat tekinti felkeresettnek, amelyek szerepelnek az adott böngésző böngészési előzményei között.

²⁸ A `:hover` álosztályt nemcsak élőkapcsok esetén használhatjuk, hanem minden olyan objektumnál, tagnál, amelyre rá tudunk az egérkurzossal mutatni.

Gyakori megoldás az is, hogy **háttérváltoztató hivatkozásokat hozunk létre**, ami azt jelenti, hogy csak akkor változtatjuk meg a hivatkozás megjelenítését (pl. az alapmegjelenés inverzére, vagyis a betű- és a háttérszín felcserélve), ha rámutatunk a hivatkozásra (pl. többoldalas webhelyünk esetén a gombszerű linkek biztosítják az oldalak közötti mozgást). Ha több oldalunkat kapcsoljuk össze ilyen módszerrel, akkor célszerű arra is ügyelnünk, hogy az aktuális oldal linkjét tegyük letiltottá, azaz az objektum szövege ne legyen kattintható (ne mutasson sehová sem) és csak a többi, más oldalakra mutató link esetén legyen a felirat háttérváltoztató hivatkozással formázott.

Előfordulhat az is, hogy **a kimenő linkeket, azaz a külső webhelyekre mutató élőkapcsokat megszeretnénk különböztetni a saját oldalainkra mutató hivatkozásoktól**. Ilyen esetben célszerű a kimenő linkekhez pl. más háttérszín, esetleg háttérképet, vagy az `::after` vagy a `::before` pszeudo-elem egyikének alkalmazásával egy kis ikont elhelyezni a hivatkozás szövege elé / mögé.

X. Listák formázása

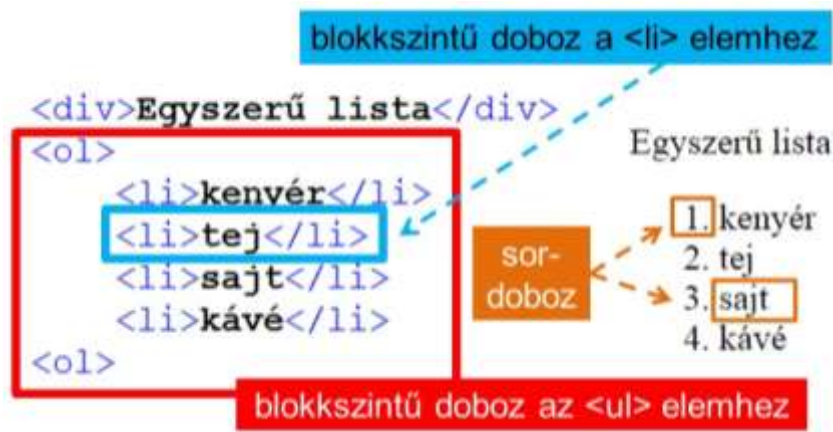
A weboldalak legelső olyan strukturális eleme, amely javította az oldalon elhelyezett információk megjelenítését, a lista volt. A HTML-ben a listáknak háromféle típusa létezik: a **számozás** (rendezett lista, `` és ``), a **felsorolás** (rendezetlen lista, `` és ``) és az **aszociáció** (definíciós lista, `<dl>`, `<dt>` és `<dd>`). A számozás alapértelmezésként (azaz külön beállítás nélkül) arab számokkal, a felsorolás a fekete korong (*disc*) jelölővel formázott. A listák egymásba ágyazásával hierarchikus szerkezetet is létrehozhatunk: ekkor arra érdemes figyelni, hogy a belső lista nem két `` elem között, hanem egy `` elem belsejében, azaz egy `` záró tag előtt szerepeljen. Számozott lista esetén a többszintű lista minden egyes eleme arab számokkal jelenik meg, felsorolás esetén a *disc* – *circle* – *square* a meghatározott sorrend és ennek alapján történik a megjelenítés.

számozott lista	felsorolás	definíciós lista
<ol style="list-style-type: none"> 1. általános iskola <ol style="list-style-type: none"> a. alsó tagozat b. felső tagozat 2. középiskola <ol style="list-style-type: none"> a. 4 évfolyamos b. 5 évfolyamos c. 6 évfolyamos d. 8 évfolyamos e. szakképzés 3. felsőoktatás <ol style="list-style-type: none"> a. főiskola b. egyetem 	<ul style="list-style-type: none"> • bölcsőde • óvoda • általános iskola • középiskola <ul style="list-style-type: none"> ○ szakképző iskola ○ szakgimnázium ○ technikum • felsőoktatás <ul style="list-style-type: none"> ○ főiskola ○ egyetem 	<p>HTML leírónyelv, amely definiálja a tartalom strukturáját</p> <p>CSS a weblapok megjelenésének leírását biztosító stíluslap</p> <p>objektum a weblapot alkotó strukturális egységek</p>

A) A list-style tulajdonság alkalmazásának lehetőségei

A listák formázása a CSS által generált doboz egy fő (tartalmi dobozból) és egy felsorolási jel-dobozból (azaz a jelölők dobozából) áll, ahogyan az alábbi ábrán is látható.

A listák esetén a már ismert betű-, szöveg- és dobozmodell-tulajdonságok mellett az is beállítható a CSS-ben, hogy milyen legyen a felsorolások és számozások listajelölője (azaz megváltoztathatjuk az alapértelmezett értéket) és akár képfájl is beállíthatunk az egyes listaelemek elé.



A szám vagy szimbólum módosításához a `list-style-type` tulajdonságnak kell értéket adni, ha pedig egy fájlban elhelyezett képet szeretnénk a jelölő helyén megjeleníteni, akkor a `list-style-image: url(...);` alakot kell használnunk.

A **list-style-type** leggyakrabban alkalmazott értékei:

- **disc** (teli kör, ●);
- **circle** (üres kör, ○);
- **square** (kitöltött négyzet, ■);
- **none** (nincs jelölő);
- **initial** (visszaállítás az alapértelmezett értékre);
- **inherit** (a szülőre beállított érték);
- **decimal** (10-es számrendszerbeli számok 1-től növekvő sorrendben: 1, 2, 3, ...);
- **decimal-leading-zero** (az előző kiegészített változata vezetőnullákkal: 01, 02, 03, ...);
- **lower-alpha** (a, b, c, ...), **lower-latin** (mint az előző, csak a latin abc-vel);
- **upper-alpha** (A, B, C, ...), **upper-latin** (mint az előző, csak a latin abc-vel);
- **lower-roman** (i, ii, iii, ...), **upper-roman** (I, II, III, ...);
- **upper-greek** (nagy görög betűk), **lower-greek** (kis görög betűk);
- mellett számos előre definiált (pl. grúz, örmény és héber karakterek, japán szótagjelek) további típus áll rendelkezésünkre.

A **list-style-type: none**; beállítás akkor lehet hasznos, ha a listajelölőket el szeretnénk távolítani, mert pl. a listából kattintható gombokat vagy menürendszert szeretnénk kialakítani. Ekkor érdemes a belső margókat (**padding**) és a külső margókat (**margin**) is nullára állítani.

A **list-style-image** esetén az **url()** függvény alkalmazásával adhatjuk meg (ha szükséges, akkor) a kép elérési útjával a kép pontos nevét kiterjesztéssel együtt. Ekkor arra is ügyelni kell, hogy a felsorolásjelnek választott kép saját magassága / szélessége megmarad a használat során, így kellően kicsire alakított képet érdemes használni (pl. az itt látható mintát).

A két beállítást akár a listaelemre (ul, ol), akár a listaitemekre (li) alkalmazhatjuk attól függően, hogy a teljes listát (vagyis az összes listaitemet) vagy csak egy-egy vagy néhány kiválasztott itemet szeretnénk formázni.

- Teljes nevén: Csontos Gyula József, Munkács, 1883. március 8. – Budapest, Erzsébetváros, 1945. augusztus 1.) magyar színész.
- Csontos sokféle szerepkörben, számtalan színházi stílushoz alkalmazkodva önálló hangú és arcú művész-egéniséggé vált.
- Drámai szerepeit mély emberábrázolással, hiteles szenvedélyességgel formálta meg.
- A komikusi szerepkörben volt elemében igazán: alakításait a tragikomikus tónus jellemezte.
- Fanyar humorával, különösképpen egyéniségével, színpadi tréfáival a magyar színművészet „rettenetes gyermeke” volt.

A **list-style-position** CSS-tulajdonsággal meghatározhatjuk azt is, hogy az egyes listaelem-jelölők (a felsoroláspontok) hol helyezkedjenek el, vagyis azt, hogy a jel kilógjon-e a szöveg vonalából. Ez a tulajdonság kétféle értéket vehet fel (ld. a következő képet):

- **outside** (alapértelmezett) = a felsoroláspontok a listaelemen kívül lesznek, azaz a jel kilóg a felsorolás szövegének vonalából („függő behúzás”)
- **inside** = a felsoroláspontok a listaelemen belül lesznek, azaz a felsorolásjel egy vonalban lesz a listaelem tartalmával (ez többsoros felsorolásnál látható jól)

HTML-kód

```
<div>Hogyan lehet három mozdulattal  
betenni egy elefántot a hűtőbe?</div>  
<ol>  
  <li>Kinyitod az ajtót.</li>  
  <li>Beteszted az elefántot a hűtőbe.</li>  
  <li>Becsukod a hűtő ajtaját.</li>  
</ol>
```

outside beállítással

- 1918-ban Kánn (Budapest, 1887. március 19. – New York, 1941. október 6.) színész, humorista.
- Pályáját mint táncoskomikus kezdte, majd a pesti kabaré utánozhatatlan egyénisége lett, a mórlandókat alkotott vígjátékokban, komikus és tragikomikus szerepekben is.
- Hadmér és dudógó beszédével, eszköztelen játékkal, félrevezető mozdulatokkal, groteszk mimikával teremtett figurákat számos 1931-1938 között készült filmben.
- 1939-ben az erősödő antiszemitizmus miatt családjával az Egyesült Államokba települt, ahol alkalmai fellépésekkel szórakoztatta a magyarságot és a közönséget.
- Filmvállalatok közül egyik sem alkalmazta, teljesen mellőzték, és utolsó évei keserűségben teltek.

inside beállítással

- 1918-ban Kánn (Budapest, 1887. március 19. – New York, 1941. október 6.) színész, humorista.
- Pályáját mint táncoskomikus kezdte, majd a pesti kabaré utánozhatatlan egyénisége lett, a mórlandókat alkotott vígjátékokban, komikus és tragikomikus szerepekben is.
- Hadmér és dudógó beszédével, eszköztelen játékkal, félrevezető mozdulatokkal, groteszk mimikával teremtett figurákat számos 1931-1938 között készült filmben.
- 1939-ben az erősödő antiszemitizmus miatt családjával az Egyesült Államokba települt, ahol alkalmai fellépésekkel szórakoztatta a magyarságot és a közönséget.
- Filmvállalatok közül egyik sem alkalmazta, teljesen mellőzték, és utolsó évei keserűségben teltek.

A felsorolások tulajdonságait összevontan, rövidített módon is megadhatjuk (hasonlóan a `border` vagy a `font` tulajdonsághoz). A `list-style` jellemzőnév mögött a `list-style-type`, a `list-style-position`, majd a `list-style-image` sorrendben, szóközzel elválasztva adjuk meg az egyes értékeket, például:

```
ul { list-style: circle inside url('jelolo.gif'); }
```

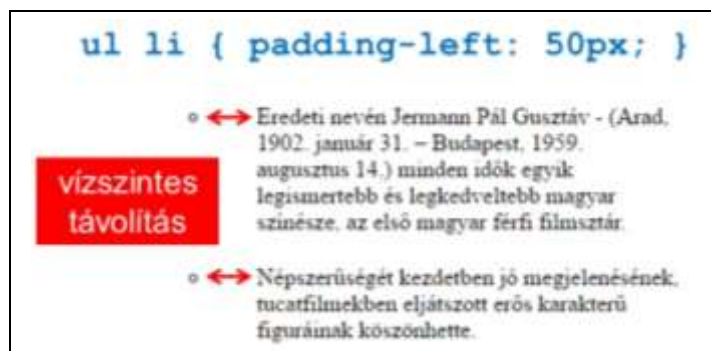
(Ha valamelyik tulajdonságérték hiányzik a felsorolásból, a böngésző a hiányzó tulajdonság alapértelmezett értékével dolgozik – ha van ilyen.)

A listákat színekkel is formázhatjuk, hogy jobban kitűnjenek a környezetükből (ld. az oldalsó ábrát), de figyelni kell arra, hogy míg az `ul` / `ol` címkékhez adott jellemzők a teljes listát, addig a `li` címkékhez adott beállítások csak az egyes listaelemeket fogják érinteni. (A példában a listaelemre és a listaitemekre eltérő háttérszínt állítottunk be.)

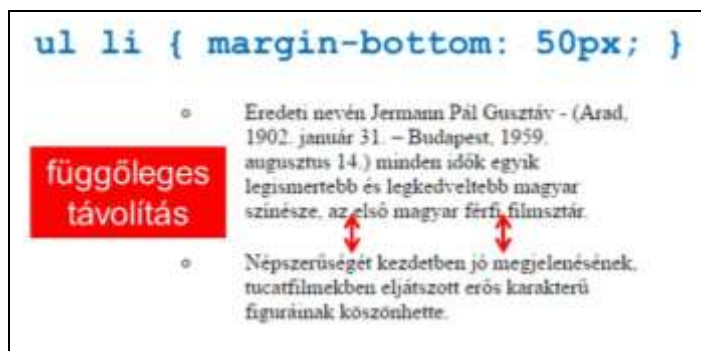
Ha egy lista minden egyes szövegét (tartalmát) beljebb akarjuk kezdeni a listajelölőtől, akkor alkalmazzuk a `padding` tulajdonságot (ld. következő ábra bal mintaképét); ha pedig egy lista egymást követő bejegyzéseit függőlegesen akarjuk eltávolítani egymástól, akkor `margin` jellemzőt kell beállítani (ld. a következő ábra jobb mintaképét).



vízszintes távolítás



függőleges távolítás



Ha egy lista minden elemét elválasztóvonallal is el akarunk szeparálni a többitől vagy táblázatszerű rúcsozatot (ld. az oldalsó ábrát) szeretnénk megjeleníteni, akkor az utolsó kivételével minden egyes listaelemhez alsó szegélyvonalat kell beállítanunk (vagy az első elem kivételével mindegyikhez felső szegélyvonalat), rácsozat esetén még a teljes lista köré is szükséges a szegély beállítása. Az alábbi kódsorban az itt látható minta kódolása tekinthető meg:

HTML-kód:

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Coca Cola</li>
</ul>
```

formázás nélkül:

- Coffee
- Tea
- Coca Cola

CSS-kód:

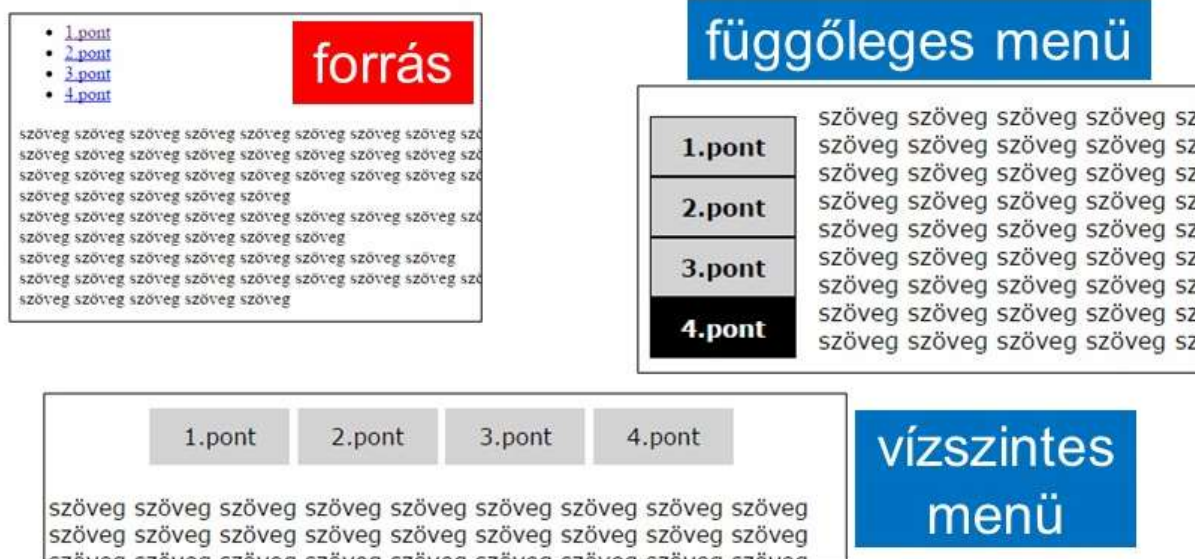
```
ul { list-style-type: none;
      border: 3px solid black; }
ul li { padding: 5px 15px;
        border-bottom: 3px solid black; }
ul li:last-child { border-bottom: none; }
```

formázott változat:

Coffee
Tea
Coca Cola

B) Listák használata menük kialakításához

A listákat gyakran arra is használják, hogy segítségükkel menüket alakítsanak ki, mert a listaelemek az egyes menüpontokként is tekinthetők.



Hogyan készíthető listából menü?

- a felsorolt elemeket hivatkozássá alakítjuk (` szöveg `) és megszüntetjük a link aláhúzását (`li { text-decoration: none; }`);
- ha nem akarjuk megtartani a listaelemek előtti jelölőket, akkor azokat is eltávolítjuk a kívánt helyekről (`ul { list-style-type: none; }`);
- ha háttérváltoztató menüpontokként akarjuk használni az egyes listaelemeket, akkor formázást rendelünk a listaelemekhez a `ul>li:hover` álosztály alkalmazásával;
- alkalmas formázások beállításával gombszerűvé alakítjuk az egyes listaelemeket (szegély, háttérszín, betűtulajdonságokat határozzunk meg);
- eltávolítjuk a gombon lévő szövege a szegélytől (`padding`);
- ha egymás mellé szeretnénk elhelyezni a gombokat, akkor megváltoztatjuk a megjelenítési módját (`display: inline-block;`);
- az egyes listaelemeket eltávolítjuk egymástól a `margin` tulajdonság megfelelő oldalra történő alkalmazásával.

C) Többszintű lenyitható listák készítése

Miként tudunk többszintű, egér fölé vitelek előbukkanó menüt készíteni?

Az egyértelműség kedvéért feltételezzük, hogy a menüket `ul` listaként valósítjuk meg, a megoldás könnyen alkalmazható lesz más esetben is. Többszintű menüt úgy kapunk, hogy a főmenü valamely eleme, a saját feliratán kívül, a `` és `` tagok között egy teljes újabb listát (pl. ` ... `) tartalmaz.

Kérdés: Hogyan érhetjük el azt, hogy ez a beágyazott lista csak akkor legyen látható, amikor az egeret az adott menüpontra pozicionáljuk (fölé visszük)?



Alapértelmezésképpen elrejtjük a belső listákat: `ul ul {display: none;}`, és ugyanakkor előírjuk, hogy amikor a felhasználó az egeret egy választott menüpont fölé viszi, az csak akkor jelenjen meg: `ul li:hover ul { display: block; }`. Ugyanígy kell eljárunk, ha ebbe a szintbe újabb alszint van beágyazva: az `ul li:hover ul ul { display: none; }` segítségével alap helyzetben elrejtjük a szinteket és az `ul ul li:hover ul { display: block; }` beállításával csak akkor jelenítjük meg, ha a megfelelő szintre mutatunk.

Az ilyen menük készítésekor azonban vegyük figyelembe, hogy a felhasználó nem memória-játékot játszik, tehát csak akkor használjuk, ha az eredetileg nem látszó menüpontok nagyon könnyen megjegyezhetők!

D) Többszintű listák többelemű (többszintű) számozása

A többszintű listák esetén a beágyazott listaelemek számozása újrakezdődik és abban nem jelenik meg az előző szint sorszáma (mintaábra bal oldali képe). A többszintű listák esetén igény jelentkezhet arra, hogy a beágyazott szintek esetén ne csak az aktuális elem száma, hanem a szülőelem száma is megjelenjen az aktuális elem előtt (mintaábra jobb oldali képe).

**többszintű lista
egyszintű számozással**

A szoftverek csoportjai	
1. Rendszerszoftverek	
1. Operációs rendszerek	
2. Meghajtóprogramok (driverok)	
3. Segédprogramok	
1. Fájlkezelők	
2. Szövegszerkesztők (editorok)	
3. Tömörítők	
4. Fejlesztési környezetek	
1. Fordítóprogramok (compilerok)	
2. Értelmezők (interpreterek) és futtatókörnyezetek	
3. Nyomkövetők és hibakeresők (debuggerek)	
4. Programszerkesztők (linkerek)	
2. Alkalmazói szoftverek	
1. Irodai szoftverek	
1. Szervezőprogramok	
2. Prezentációkészítők	
3. Kiadványszerkesztők	
4. Táblázatkezelők	
5. Adatbázis-kezelők	
2. Üzleti alkalmazások	
1. Számlázóprogramok	
2. Könyvelőprogramok	
3. Vállalatirányítási rendszerek	
3. Tervezőrendszerek	
4. Grafikai szoftverek	
1. Rajzprogramok	
2. Képszerkesztők	

**többszintű lista
többszintű számozással**

A szoftverek csoportjai	
1 Rendszerszoftverek	
1.1 Operációs rendszerek	
1.2 Meghajtóprogramok (driverok)	
1.3 Segédprogramok	
1.3.1 Fájlkezelők	
1.3.2 Szövegszerkesztők (editorok)	
1.3.3 Tömörítők	
1.4 Fejlesztési környezetek	
1.4.1 Fordítóprogramok (compilerok)	
1.4.2 Értelmezők (interpreterek) és futtatókörnyezetek	
1.4.3 Nyomkövetők és hibakeresők (debuggerek)	
1.4.4 Programszerkesztők (linkerek)	
2 Alkalmazói szoftverek	
2.1 Irodai szoftverek	
2.1.1 Szervezőprogramok	
2.1.2 Prezentációkészítők	
2.1.3 Kiadványszerkesztők	
2.1.4 Táblázatkezelők	
2.1.5 Adatbázis-kezelők	
2.2 Üzleti alkalmazások	
2.2.1 Számlázóprogramok	
2.2.2 Könyvelőprogramok	
2.2.3 Vállalatirányítási rendszerek	
2.3 Tervezőrendszerek	
2.4 Grafikai szoftverek	
2.4.1 Rajzprogramok	
2.4.2 Képszerkesztők	

A megoldás a számlálók alkalmazásában rejlik. A CSS-számlálók olyanok, mint a programozási nyelvekben a változók, nevük tetszőleges azonosítónév lehet, és értéküket módosítani (növelni) lehet – itt most CSS-szabályokkal – ezek nyomkövetik, hogy hányszor használják azokat. A többszintű számozásnál minden gyermekelem automatikusan egy újabb számlálót hoz létre, így nem kell az egymásba ágyazott listák minden egyes szintjéhez külön számlálónevet adni.

A számláló (**counter-reset**) tulajdonságai:

- **több számlálót definiál** vagy állítja vissza azok alapértékeit;
- a létrehozott számlálók **hatóköre** kiterjed a komponensre, amelyhez a számláló létrejön, és annak kapcsolataira, illetve az összes származtatott komponensre és azok kapcsolataira;
- kiegészülve a **counter-increment** jellemzővel valósítja meg az automatikus számozást és a **content** tulajdonsággal tudjuk megjeleníteni a számláló értékeit.

A többszintű számlálók létrehozásának menete:

- a felsorolás minden szintjéhez egy-egy számlálót rendelünk a **counter-reset** tulajdonsággal és eltüntetjük az alapértelmezetten megjelenő számozást:


```
ol { counter-reset: section; list-style-type: none; }
```
- a számláló értékét minden egyes listaelemnél növeljük a **counter-increment** értékének beállításával és elhelyezzük az elem elé:


```
li::before { counter-increment: section;
               content: counters(section, ".") " "; }
```

XI. A kurzor formájának meghatározása

A különböző elemekhez a `cursor` CSS-tulajdonsággal rájuk jellemző vagy róluk információt közvetítő kurzor-alakzatok kódolhatók, amelyeket általában a kurzort az elem fölé mozgatva, azaz a **`:hover`** dinamikus ál-osztály segítségével definiálva használhatunk.

A `cursor` tulajdonság lehetséges értékei:

- `auto`:** a böngésző határozza meg a kurzor alakját (ferde nyíl)
- `default`:** az operációs rendszer határozza meg a kurzor alakját (ferde nyíl)
- `help`:** segítség áll rendelkezésre (nyíl melletti kis kérdőjel)
- `pointer`:** linkre utaló kurzor (kézfej kinyújtott mutatóujjal)
- `wait`:** a program dolgozik, a felhasználónak várnia kell (homokóra, forgó kör)
- `progress`:** a program dolgozik, de a felhasználó közbeavatkozhat (homokóra, forgó kör)
- `text`:** választható szöveg (függőleges vagy ferde vékony vonal, mint a szövegbeírásnál)
- `move`:** mozgatható elem (négyágú nyilas kereszt)
- `not-allowed`:** a kívánt művelet nem hajtható végre (piros szegélyű fehér kör pirossal ferdén áthúzva)
- `not-drop`:** a vonszolt elem nem ejthető le (piros szegélyű fehér kör pirossal ferdén áthúzva)
- `crosshair`:** raszterképes kijelölési mód (vékony kereszt)
- `all-scroll`:** bármilyen irányban görgethető elem (négy irányú nyíl)
- `copy`:** másolható elem (nyíl mellett jobb alul kis kereszt)
- `cell`:** kiválasztható cella vagy cellacsoport (vastag fehér kereszt)
- `alias`:** létrejövő elem (nyíl mellett egy kis görbe nyíl)
- `none`:** eltűnik a kurzor
- `zoom-in`, `zoom-out`:** nagyítás, kicsinyítés (nagyító + jellel vagy – jellel a közepén)
- `context-menu`:** menü érhető el az elemhez (nyíl mellett vagy helyett kis menüábra)
- `col-resize`:** oszlop vízszintes átméretezése (2 vízszintes nyíl között 2 függőleges vonal)
- `row-resize`:** sor függőleges átméretezése (2 függőleges nyíl között 2 vízszintes vonal)

Az **oldalirányú átméretezésekhez** a tulajdonság értékét a **`...-resize`** alakú értékkel adhatjuk meg, amelyben a ... helyére kerül az irány jelölése az alábbiak szerint:

- `e`** (east) = keletre, jobbra (pl. `.jobbraMeretez { cursor: e-resize; }`)
- `w`** (west) = nyugatra, balra
- `n`** (north) = északra, felfelé
- `s`** (south) = délre, lefelé
- `se`** (south-east) = dél-keletre, jobbra le, illetve **`sw`** (south-west) = dél-nyugatra, balra le
- `ne`** (north-east) = észak-keletre, jobbra fel, illetve **`nw`** (north-west) = észak-nyugatra, balra fel

Ha kétirányú az átméretezhetőség, akkor az **`ew-resize`** (vízszintes), az **`ns-resize`** (függőleges), az **`nesw-resize`** és az **`nwse-resize`** (átlós) értékeket használhatjuk. (Az oldalirányú átméretezések beállításánál a kurzor vízszintes, függőleges vagy átlós irányú, kétágú nyílként jelenik meg.)

XII. Oldalstruktúra kialakítása

A weboldalak kialakítása során általános elvárásunk, hogy az elkészített weblapunk optimális módon jelenjen meg minden látogató számára, ami azt jelenti, hogy:

- a tartalom töltsse ki a rendelkezésre álló felületet;
- a weblap elemei oldal irányban ne lógnak le a megjelenítő felületről, vagyis egy kicsi monitoron se kelljen vízszintes irányban görgetni;
- az oldal felépítése egyezzen meg az általunk tervezett elrendezéssel (ez triviális elvárásnak tűnik, de az előző két követelmény teljesítésével együtt általában nehéz megvalósítani);
- bármilyen eszközön (asztali számítógépe, tableten vagy telefonon) is nézzük, alkalmazkodjon a tulajdonságaihoz úgy, hogy az a felhasználói élményt (pl. olvashatóság) ne befolyásolja.

Egy weblap elemeinek pozicionálása, illetve az oldalfelépítés kialakítása mindig is a weblapkészítés legproblémásabb területei közé tartozott. A felhasználók különböző felbontású és méretű monitorokon, különböző operációs rendszereken különböző böngészőkkel nézik meg a világhálóra feltöltött oldalakat.

Korábban két megoldás közül választhattunk:

- Minden elképzelhető összeállításra megpróbáltunk nagyon precíz beállításokat megadni. Egy ilyen weblap elkészítése nagyon sok munkával járt és egy nem tesztelt konfiguráció esetén „széteshetett” a struktúra, a weboldal részben „használatatlanná” vált.
- A másik lehetőség az volt, hogy csak a legszükségesebb tulajdonságokat definiáltuk. Ebben az esetben a felhasználók alapbeállításai jelentős hatást gyakoroltak a weblap megjelenésére.

A problémára többféle megoldás létezett, de egyik sem volt tökéletes:

- Egy **JavaScript program lekérdezte a képernyő méretét és átalakította a weblapot**, ha az szükséges volt. Komoly programozói tudás kellett hozzá, de biztos megoldást biztosított. (Feltéve, hogy a látogató gépén nincs kikapcsolva a JavaScript.)
- **Különböző stíluslapokat kellett definiálni** a különböző médiatípusokhoz, így a megjelenítési tulajdonságokat könnyű volt megváltoztatni, de a tartalom „átszabása” már nehézkesebbé vált.
- Olyan **oldalfelépítést** kellett kódolni, amit **rugalmasan (és automatikusan) tudott kezelni a böngésző**. Ekkor a weblap tartalmának elrendezése során ügyelni kellett arra, hogy a legfontosabb információk kisméretű megjelenítőn is a látható tartományban kerüljenek.

Ez az utolsó verzió jelenti az igazi megoldást.

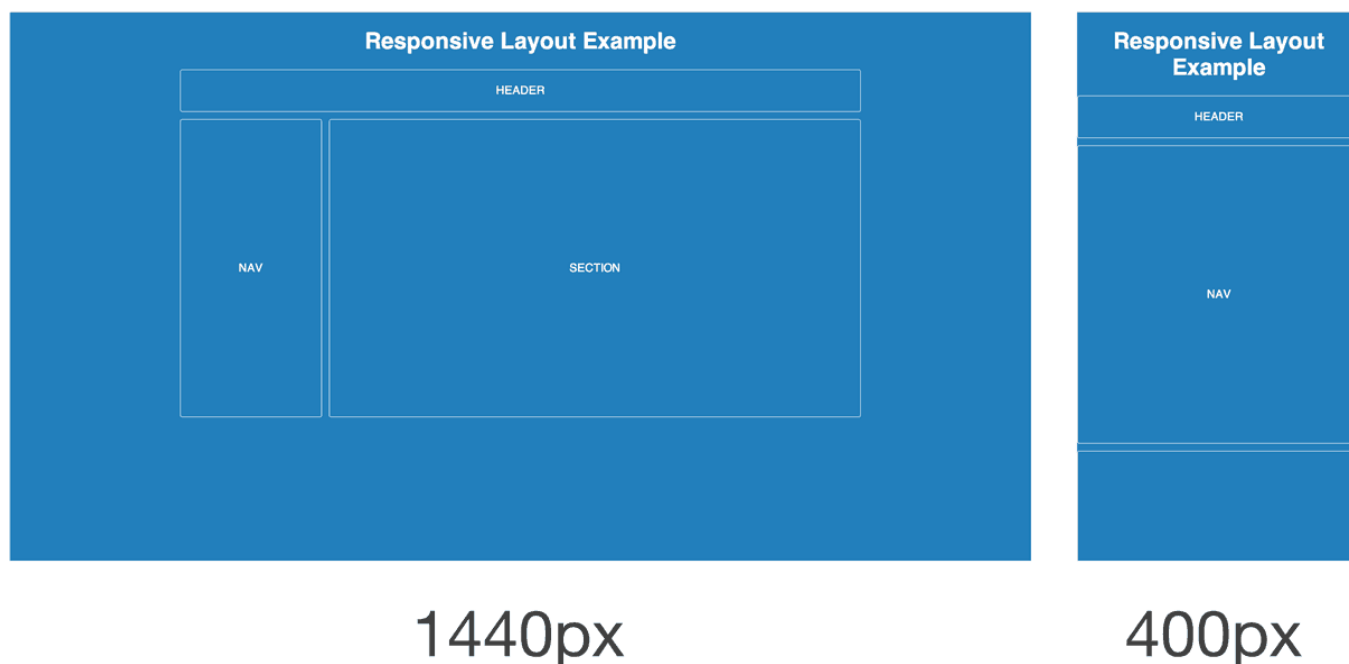
A **reszponzív dizájn (RWD) megjelenése és alkalmazása ma már olyan tervezési módszert jelent, amelynek segítségével optimális megjelenést (egyszerű olvashatóságot, könnyű navigáció és a legteljesebb felhasználói élményt) biztosíthatunk webhelyeinknek úgy, hogy bármilyen eszközt is használunk, a webhelyünk megjelenése automatikusan reagál az eszköz tulajdonságaira (felbontás, tájolás, böngésző típusa).** Ennek az az oka, hogy az RWD az **átméretezésen** és az **átrendezésen** alapul úgy, hogy az egyes elemek fizikailag nincsenek duplikálva, azaz tartalmilag ugyanazt a honlapot látjuk mindenhol, csak átrendezett és átméretezett külsőben. **A reszponzivitás teljesítéséhez jó segítséget nyújtanak a CSS-keretrendszerek is**, hiszen segítségével a fejlesztés meggyorsul, mert előre elkészített és használatra kész CSS stílusgyűjteménnyel dolgozhatunk.

Az oldalelrendezés (layout) határozza meg azt, hogy hogyan néz ki a weboldal felépítése, struktúrája. Többféle oldalelrendezési típus van (statikus, folyékony, adaptív, reszponzív) és mindegyik neve egyben leírja, **hogyan viselkedik az elrendezés, amikor az oldalt különböző böngészőszélességekben vagy különböző eszközökön tekintjük meg.**

Az oldalelrendezések (layout) kialakításakor ma már négyféle elvet követhetünk: statikus, folyékony, adaptív és reszponzív oldalelrendezés szerint építhetjük fel az oldalak struktúráját.

A 4 oldalelrendezés közül a reszpozív (rugalmas vagy illeszkedő) layout használata javasolt.

A reszponzív layout relatív egységeket és médialekérdezéseket használ, így látszólag ötvözi az adaptív és a folyékony elrendezés ötleteit. Ha a böngésző szélessége nő / csökken, akkor az oldal szerkezete automatikusan megváltozik (ld. az oldalon található ábrát). Ha a böngésző túllép bizonyos, a médialekérdezés töréspontja által meghatározott szélességeken, akkor az elrendezés drasztikusabban változik. Jó módszer a mobile first megközelítésű tervek megvalósításához.



Előnye, hogy jól alkalmazkodik a vízszintes méretváltozáshoz és a változó ablakmérethez, illetve eszközfüggetlen a megjelenítés, így követi a jó felhasználói élmény elveit.

Hátránya, hogy néha lehetetlen a felhasználói élmény maximális kielégítése minden egyes képernyőméretre vagy készülékre szabva, hosszabb fejlesztési időt (több dizájn és fejlesztői munkát) igényel, lassú oldalbetöltési sebességet ad.

Az egyes oldalelrendezések működésének kipróbálásához érdemes ellátogatni az alábbi oldalra: http://g-mops.net/epica_saitama/epica_layout/index_adaptive.html, amelynél a jobb felső sarokban választható a layout típusa, az ablak átméretezésével pedig megfigyelhető a layout struktúrájának változtatása.