

Villamosmérnöki szak
Alkalmazott informatika mellék szakirány
Laboratórium II. (VIAU5102)
World Wide Web lapok készítése (Silabusz)

HTML (Hyper Text Markup Language)	2
A HTML dokumentumokról.....	2
Néhány szóban a HTML alapvető szabályairól	3
A HTML dokumentum elvi felépítése	4
A HTML dokumentum fejléce.....	4
A HTML dokumentum szövegtestének felépítése	5
A HTML dokumentum címszintjeinek felépítése.....	6
Bekezdések a HTML dokumentumban	6
Kereszthivatkozások HTML dokumentumok között	7
Karakterformátumok a HTML dokumentumban	8
Képek elhelyezése a HTML dokumentumban.....	9
Különböző listafomátumok a HTML dokumentumban	11
A HTML formátum táblázatai	13
A HTML dokumentumablak felosztása	16
Kérdőívek a HTML dokumentumban	17
A HTML dokumentum egyéb elemei.....	19
A JavaScript használata a HTML dokumentumban.....	19
A HTML speciális karakterei.....	20
Karaktertáblák	23
Cascading Style Sheets (CSS)	23
Bevezetés	23
A stíluslap csatolása.....	24
Csportosítás	24
Öröklődés	25
A class szelektor.....	25
Megjegyzendő, hogy szelektoronként (HTML elemenként) csak egy osztály definiálható!	
.....	26
Az ID szelektor.....	26
Összekapcsolt szelektorok	26
Megjegyzések.....	26
Látszólagos osztályok és elemek	26
Látszólagos osztályok az előkapcsokban.....	27
Tipografikai látszólagos elemek.....	27
A 'first-line' látszólagos elem	27
A 'first-letter' látszólagos elem	28
Látszólagos elemek a szelektorokban.....	29
Látszólagos elemek többszörözése.....	29
Rangsor	29
'important'.....	30
A rangsor felállítása.....	30
Formázásmodell	31
Blokkszintű elemek	31

Függőleges formázás	33
Vízszintes formázás	34
Listaelemek	34
Lebegő elemek	35
Soron belüli elemek	36
Helyettesített elemek	36
Sorok magassága	37
A Vázon.....	37
CSS tulajdonságok	38
A tulajdonság-érték párok jelölési rendszere	38
Font tulajdonságok	38
Szöveg tulajdonságok	46
Doboz-tulajdonságok.....	48
Oszályozó tulajdonságok	55
Egységek.....	57
Hosszúság egységek	57
Százalékos egységek.....	58
Színjelölések	58
URL	59
Összhang	59
Előre-kompatibilis elemzés	60
JavaScript.....	62
Mi is az a JavaScript?	62
JavaScript és a böngészők.....	62
JavaScript beágyazása a HTML dokumentumba	63
Legfontosabb események.....	64
Használjuk amit tudunk, 2. példa	64
Link a Script-re	64
Csoportosítsunk, avagy a függvények	65
Objektumok, dokumentumok	66
Ha hibáztunk	67
Események	67
Változó képek	68
Előtöltött képek	68
Időzítések	69
Status sor.....	70
SWITCH használat.....	70
Ellenőrzések	73
"Üres szöveg"	74

HTML (Hyper Text Markup Language)

A HTML dokumentumokról

A HTML dokumentum-formátumot tekinthetjük az ún. hyper-text egyik megvalósítási formájának is. A HTML dokumentum egy olyan szövegfájl, amely a szövegen kívül tartalmaz ún. "HTML-tag"-eket - formázóutasításokat -, valamint megjelenítendő objektumokra történő hivatkozásokat is. Ezek a HTML formázóutasítások (más szóval: parancsok, elemek) befolyásolják a dokumentum megjelenítését, kapcsolatait. Ezeket az utasításokat a böngészőprogram értelmezi és végrehajtja. Ezen okból a formázóutasítás mindig megelőzi azt a részét a dokumentumnak, amelyre vonatkozik. A dokumentumkészítéshez használható

HTML utasítások köre állandóan bővül, a nyelv fejlődik. A szabványosítás csak lassan követi a fejlődést. Ezért nem minden böngészőprogram tudja a HTML utasítások mindegyikét értelmezni. Egy böngésző, ha számára értelmetlen utasítással találkozik, akkor kihagyja, így nem okoznak problémát az újabb keletű - még szabványosítatlan - utasítások a régebbi kiadású WWW-böngészőknek sem. Sajnos a fentiek miatt ugyanazt a dokumentumot két különböző program nem biztos, hogy azonos formában fogja megjeleníteni. Más oka is van ennek. A WWW-n kalandozónál kicsi a valószínűsége annak, hogy rendelkezésére áll ugyanaz a betűtípus, mint a WWW-oldalt fejlesztőnek. Vagy képek esetén semmi garancia nincs arra, hogy minden böngészőprogram ugyanazon felbontásban és színszámmal tudja megjeleníteni a képet. És így tovább ... A HTML-ben mégis az a nagyszerű, hogy nagymértékben megközelíti a platformfüggetlenséget. Egy HTML dokumentum - ha nem is azonos módon - mindenki számára megtekinthető.

Néhány szóban a HTML alapvető szabályairól

A HTML dokumentum normál szövegfájl. Bármely szövegszerkesztővel létrehozható, ill. módosítható, amely nem használ különleges fájlformátumot, vagy ha létezik TEXT formátumú mentési lehetőség benne. A HTML utasításokat a szövegben < és > jelek közé kell zárni. Egy-egy utasítás - HTML parancs, HTML elem - hatását általában a záró utasításpárja szünteti meg, amely megegyezik a nyitó utasítással, csak a / jel vezeti be (természetesen a < és a > jelek között). Az utasítások nagy része opcionális elemeket is tartalmazhat, melyek csak a nyitóutasításban szerepelhetnek, a záróban nem. Az opciók értékadásánál az idézőjel nem mindig kötelező, csak ajánlott. A HTML utasítás kulcsszavaiban nem különböztetjük meg kisbetűket és nagybetűket.

A HTML dokumentum elvi felépítése

Minden HTML formátumú szövegfájl a **<HTML>** utasítással kezdődik és a **</HTML>** záróutasítással végződik. Ezen elemek közé kell zárni a teljes dokumentumot - formázóutasításokkal és hivatkozásokkal együtt.

A HTML dokumentumot két részre lehet bontani a fejlécre és dokumentumtörzsre. (Egy harmadik rész lehet a keretek definíciója.)

A dokumentumot a fejlécelemek vezetik be, melyek kezdetét a **<HEAD>** utasítás jelzi. A fejlécelemek között szokás a dokumentumcímét megadni, mely címet a **<TITLE>** és a **</TITLE>** utasítások közé kell zárni. A fejlécet a **</HEAD>** utasítás zárja. Ezt a részét a dokumentumnak általában az ablak címsorában jelenítik meg a böngészőprogramok.

A dokumentumtörzs - amit voltaképpen a WEB böngésző meg fog jeleníteni - a fájl **<BODY>** és **</BODY>** utasítások közötti része. Ezen elemek között kell elhelyezni mindent: a szöveget, hivatkozásokat, képeket, stb. (A keretek és a JavaScript kódok kivételével!)

Példa (egy tipikus HTML dokumentum felépítése):

```
<HTML>  
<HEAD>  
<TITLE>A dokumentum neve</TITLE>  
Fejléc elemek ...  
</HEAD>  
<BODY>A tulajdonképpeni dokumentumtörzs következik ..  
<H1>Ez itt egy alcím</H1>  
<P>Ez itt egy normál bekezdés </p>  
<!-- HTML comment ez nem jelenik meg a dokumentumban -->  
</BODY>  
</HTML>
```

A HTML dokumentum fejléce

A dokumentumot a fejlécelemek vezetik be, melyek kezdetét a **<HEAD>**, végét a **</HEAD>** utasítás jelzi. A fejlécelemek között legfontosabb a dokumentumcím, mely címet a **<TITLE>** és a **</TITLE>** utasítások közé kell zárni. Ezt a részét a dokumentumnak általában az ablak címsorában jeleníti meg a legtöbb böngészőprogram. A **<BASE HREF="protokoll://gépnév/elérési_út">** utasításban szereplő URL határozza meg a báziscímet, melyből a relatív címeket értelmezni kell. Az intelligens kiszolgálók korábban nem kötelező megadni.

Az **<LINK>** utasításban szereplő opciók jelzik dokumentum kapcsolatait más dokumentumokkal, stíluslappal, címszalaggal, stb.

Az **<META NAME="mező" CONTENT="érték">** utasítás jelezheti a keresőrendszerek számára a dokumentum-adatbázisba kerülő adatokat, pl. a dokumentum alkotóját, a létrehozó programot, rövid tartalmat, stb.

A fejlécelemekre jellemző, hogy a böngészőablakban nem jelennek meg! Ezért a World Wide Weben szereplő dokumentumok fejlécének csak nagyon kis hányada tartalmaz a címen kívül egyéb információkat. (HTML fejléc részletesebb leírását lásd. ASP tárgyalásán belül)

Példa (HTML dokumentum fejléc):

```
<HTML>
  <HEAD>
    <TITLE>HTML leírás - 3. lap</TITLE>
    <META NAME="Author" CONTENT="Almási Pál">
    <LINK REL="stylesheet" HREF="pelda.css"><!--ez egy stylesheet
hozzárendelés lásd. CSS-rész -->
  </HEAD>
  <BODY> ... dokumentumtörzs ...
</BODY>
</HTML>
```

A HTML dokumentum szövegtestének felépítése

Minden HTML formátumú szövegfájl a **<BODY>** és a **</BODY>** utasításokkal közrezárt részében tartalmazza a megjelenítendő részét. (A dokumentum-kereteket kivéve!)

Ezen elemek között kell elhelyezni mindent: a szöveget, hivatkozásokat, képeket, stb.

A **<BODY BACKGROUND="fájlnév.kit" BGCOLOR="színkód" TEXT="színkód" LINK="színkód" VLINK="színkód" ALINK="színkód">** utasításban a dokumentumtörzshöz vonatkozó fenti előírások is szerepelhetnek opcióként.

A **BACKGROUND="elérési_út/fájlnév.kit"** opcióval a dokumentum háttéréül szolgáló fájlt jelölhetjük ki.

Háttérszín a **BGCOLOR="színkód"** opcióval kiegészített utasítással definiálhatunk. (Amennyiben háttérmintául szolgáló fájlt - lásd fent - is megadunk, akkor a háttérszín csak nagyon ritkán fog előtűnni a dokumentumban, pl. a keretek szegélyében.)

A dokumentumban a szöveg színét a **TEXT="színkód"** opcióval jelölhetjük ki.

A **LINK="színkód"** opció a hivatkozások megjelenési színét határozza meg. A **VLINK="színkód"** pedig, a már bejárt hivatkozásokat jelölő színt határozza meg.

Természetesen egyszerre több opció is szerepelhet - tehát nem kötelező egyik sem - a **<BODY>** utasításban.

Példa

```
<BODY
  BACKGROUND="pelda.gif"
  BGCOLOR="#FF3333"
  TEXT="#000099"
  LINK="#993399"
  VLINK="#009999">
  <P> A szemléltetés kedvéért az alábbi szöveg széles szegéllyel határolt! <P>
  <TABLE BORDER=12>
    <TH><TD><H1><A HREF="#pelda">Ez itt egy hivatkozás,</A> kattints
    rá!
  </H1></TD></TH></TD>
  </TABLE>
  <P> <IMG SRC="k1.gif" ALT="Görgetősáv"> <P>
  <A NAME="pelda"> Ide mutat a hivatkozás!</A>
</BODY>
```

A HTML dokumentum címszintjeinek felépítése

A HTML formátumú szövegfájlban definiálhatunk címeket, ill. alcímeket hat szint mélységig. A legfelső szintű címet a **<H1 ALIGN="hely">** és a **</H1>** utasításpárral kell közrezárni. A második szintet a **<H2 ALIGN="hely">** és a **</H2>** utasítások határolják, és így tovább a hatodik szintig.

Minden szint más-más betűformátumban jelenik meg a dokumentumban, a böngészőprogram beállításától függően. A címek igazítását szabályozza az **ALIGN** opció, melynek lehetséges értékei: *left*, *center*, *right*. Amennyiben túl hosszú a cím, de egy sorosnak kell maradnia, akkor a **NOWRAP** opció megakadályozza a cím betördelését több sorba.

Tulajdonsága, hogy nem szkrollozódik a dokumentum többi részével ellentétben.

A címek csak a szemléltető számára keltik a tagoltság érzetét, a valóságban nem tagolják fizikailag szakaszokra a dokumentumot. Ezt a tagolást a **<DIV CLASS="osztály" ALIGN="hely">**, **</DIV>** utasításokkal lehet meghatározni, ahol a **CLASS** opció sorolja a megfelelő SGML osztályba a szakaszt, az **ALIGN** pedig a szakasz igazítási formátumát írja elő. Az automatikus tördelést itt is megakadályozza a **NOWRAP** opció, ez esetben a szakasz tördelését a **<P>** vagy a **
** utasításokkal lehet szabályozni.

Példa a címszintekre:

<H1 ALIGN="left">Legfelső szintű címsor</H1>

<H2 ALIGN="center">Második szintű alcímsor</H2>

<H3 ALIGN="right">Harmadik szintű alcímsor</H3>

<H4 NOWRAP>Negyedik szintű alcímsor</H4>

<H5>Ötödik szintű alcímsor</H5>

<H6>Hatodik szintű alcímsor</H6>

<DIV ALIGN="center"> Ez egy szakasz, melyben a szöveg elvileg azonos módon - középre igazítva - kezelendő. </DIV>

Bekezdések a HTML dokumentumban

Minden dokumentum, így a HTML formátumú dokumentum is, alapvetően bekezdésekre tagolódik.

A HTML fájlban a bekezdések *kezdését* a **<P>** utasítás jelzi a böngészőprogram számára a végét **</P>**. A legtöbb böngészőprogram két bekezdés között egy üres sort szúr be megjelenítéskor!

A bekezdés elem hordozhat magában a bekezdés stílusát meghatározó opciókat. A bekezdés igazítását a **<P ALIGN="hely">** formájú utasítással szabályozhatjuk. Az automatikus tördelést a **NOWRAP** opció tiltja meg a böngészőprogram számára. Amennyiben tördelhetetlen szóközt igényel a szöveg, akkor az egyszerű szóköz helyett alkalmazzuk a ** ** különleges karaktert.

Amennyiben egy bekezdésen belül mindenképpen új sort szeretnénk kezdeni, akkor a **
** utasítást kell használni. (Nincs zárópárja!)

Példa bekezdésekre:

<P>Ez alapértelmezett (balra igazított) bekezdés.</p>

<P ALIGN=left>Ez balra igazított bekezdés.</p>

<P ALIGN=center>Ez középre igazított bekezdés.</p>

<P ALIGN=right>Ez jobbra igazított bekezdés.</p>

<P ALIGN=justify>Ez sorkizárt bekezdés lenne.</p>

<P>

Itt egy sortörés elem található,

melynek hatására új sorban folytatódik a szöveg,

és nem maradt ki üres sor.</p>

Kereszthivatkozások HTML dokumentumok között

A HTML formátum lényegét az egymásra és egymás tartalmára való hivatkozások jelentik (vagyis a hypertext lehetőség). A dokumentum bármely részéhez hivatkozást (linket) helyezhetünk el, amelyet aktivizálva, a hivatkozottal összefüggésben lévő szöveghez jutunk el. A hivatkozó utasítások megjelenési formája sokféle lehet, a célobjektumtól függően:

A legegyszerűbb esetben a hivatkozás az adott fájl egy távolabbi részére mozdtítja a böngészőablakot. A hivatkozás kezdetét a utasításnak a dokumentumban való elhelyezése jelzi. A hivatkozást a utasítás zárja le. Ez az elempár közrezárhat szövegrészt, képet, stb. A közrezárt részt a böngészőprogram a dokumentum többi részétől eltérően jeleníti meg (pl. aláhúzással, kerettel, ...), az egérkurzorral fölé érve a mutató alakja megváltozik. Azt a részt (praktikusan: könyvjelzőt), ahová a hivatkozás mutat a és a utasítások kell, hogy határolják.

A legtöbb esetben a egy hivatkozás egy másik fájlra/dokumentumra mutat. A hivatkozás kezdetét ekkor a utasítás jelzi, a hivatkozást ekkor is a utasításelem zárja le. Mind a protokoll, mind az elérési út elhagyható, amennyiben azonos URL-en van a kiindulási dokumentum és a hivatkozott. A hivatkozott fájlnek e példában nincs külön névvel (könyvjelzővel) jelölt része. Működés szempontjából a fentebb leírtak vonatkoznak erre a hivatkozási formára is.

A legbonyolultabb esetben a hivatkozás egy másik fájl valamely pontosan meghatározott részére mutat. A hivatkozás kezdetét a utasítás jelzi, és a hivatkozást szintén a elem zárja le. Ebben az esetben a hivatkozott fájl kell, hogy tartalmazzon egy olyan részt (könyvjelzőt), ahová a hivatkozás mutat. Ezt a részt a és a utasítások határolják.

Megjegyzés: Ha az , utasításpár képet fog közre, akkor a kép szegéllyel jelenik meg, amely szegély letiltható az utasításban elhelyezett BORDER=0 opció alkalmazásával. A képekkel kapcsolatos egyéb hivatkozási lehetőségeket lásd a képeknél.

Példa hivatkozásokra:

Ennek a fájlnak a végére vissz ez a hivatkozás.

A makói Almási Utcai Általános Iskola honlapjára vonatkozik e

hivatkozás.

helyére

repít e hivatkozás. Ezen leírással kapcsolatos véleményed

írd meg!

Itt az oldal vége!

Karakterformátumok a HTML dokumentumban

A HTML formátumú szövegfájlban is használhatjuk a szövegszerkesztőkben megszokott karakterformátumokat. Az alábbi táblázat a formázás kezdő és záróutasítása között a mintát is tartalmazza.

Kezdő elem	Ilyen betűformátumot eredményez	Záró elem
	Félkövér betűformátumot eredményez	
<I>	Dőltbetűs formátumot eredményez	</I>
<U>	Aláhúzott formátumot eredményez	</U>
<S>	Áthúzott formátumot eredményez	</S>
<TT>	Fixpontos betűket eredményez	</TT>
	Kiemeli a szöveget	
<CITE>	Idézetekre használható	</CITE>
<VAR>	Változónevet jelöl	</VAR>
	Ez is egy kiemelési lehetőség	
<CODE>	Kódoknál használjuk	</CODE>
<SAMPLE>	Minták jelzésére használjuk	</SAMPLE>
<KBD>	Billentyűfelirat jelzése	</KBD>
<BQ>	Idézet megjelenítése	</BQ>
<BIG>	Nagyméretű betűformátumot eredményez	</BIG>
<SMALL>	Kisméretű betűformátumot eredményez	</SMALL>
_{	Alsóindexet eredményez	}
^{	Felsőindexet eredményez	}
	A részleteket lásd lentebb	

A , utasításpárral direkt módon előírhatók a megjelenő szöveg betűinek a jellemzői. A **FACE** opciót nem szokás használni, mert nem valószínű, hogy minden rendszerben rendelkezésre áll pl. az **ARIAL CE FÉLKÖVÉR** betűtípus. A **COLOR** opció pontosan meghatározza a megjelenítendő szöveg színét. A **SIZE** opcióban egy számot megadva a betűméretet határozza meg direkt módon. (A **SIZE** opcióban előjeles szám is szerepelhet, ami az alapbetűtípushoz viszonyított méretet jelöl.)

Példa karakterformátumokra:

Vastag
<I>Dőlt</I>
<U>Aláhúzott</U>
<S>Áthúzott</S>
<TT>Fixpontos</TT>
Kiemelt
<CITE>Idézet</CITE>
<VAR>Változónév</VAR>
Kiemelt
<CODE>Kód</CODE>
<SAMPLE>Minta</SAMPLE>
<KBD>Billentyűfelirat</KBD>
<BQ>Idézet</BQ>
<BIG>Nagyméretű</BIG>
<SMALL>Kisméretű</SMALL>
_{Alsóindex}
^{Felsőindex}
Kicsi piros
N
ö
v
e
k
v
õ
ARIAL CE
SYMBOL

Képek elhelyezése a HTML dokumentumban

A HTML formátumú dokumentumban képeket - grafikákat - is elhelyezhetünk. Az utasítás a szöveg aktuális pozíciójába helyezi a megadott képet.

Ennél azért a legegyszerűbb szövegszerkesztő program is többet nyújt. A HTML dokumentum csinosítására is vannak a képek elhelyezésének finomabb lehetőségei is. Ha ezeket mind kihasználjuk, akkor az utasítás a következőképpen fog kinézni: .

Az **ALIGN** opció meghatározza a kép igazításának módját, lehetséges értékei: *top*, *middle*, *bottom*, *left*, *right*.

A **HSPACE** a kép melletti vízszintes térközt, a **VSPACE** pedig a függőleges térközt (ha úgy tetszik: margókat) határozza meg.

A **WIDTH** a szélességét, a **HEIGHT** pedig a magasságát adja a képnek, az **UNITS** által meghatározott egységben (*pixel* vagy *en*).

Az **ALT** azt a szöveget adja meg, amelyet nem grafikus böngészők használata esetén meg fog jelenni a kép helyett. A grafikus böngészőkben meg ha kurzort a kép fölé mozgatjuk akkor jelenik meg a kurzor alatt.

Az **USEMAP**, **ISMAP** összetartozó opciók a kép különböző területeihez különböző hipertext hivatkozásokat rendelhetnek. (Tehát csak akkor van értelmük, ha a kép egy hivatkozás része! Ekkor a hivatkozások **** és **** utasításpárját nem kell megadni.) Ezenkívül szorosan kapcsolódik ezen opciókhoz (az **** utasítást megelőzően) a következő utasításstruktúra:

```
<MAP NAME="jelző">
```

```
<AREA SHAPE="alak" COORDS="koordináták" HREF="hivatkozás">
```

...

```
</MAP>
```

amellyel egy hivatkozási térképet kell megadni. Az **<AREA>** utasításból természetesen több is szerepelhet. A **SHAPE** opció a *circle*, *rect*, *polygon* értékeket veheti fel, amikor *circle* (kör) esetén a **COORDS** három vesszővel elválasztott koordinátát tartalmaz (középx,középy,sugár), *rect* (téglalap) esetben négyet (balfelsőx,balfelsőy,jobbalsóx,jobbalsóy), a *polygon* (sokszög) esetén pedig minden csúcs koordinátáit meg kell adni. A **<MAP NAME="jelző">**, **</MAP>** utasításpárral körülhatárolt hivatkozási rész külön fájlban is elhelyezhető. Ekkor az **USEMAP** opció kimarad. - Vigyázat az **ISMAP** nem marad el! - Helyette a **** és az **** utasítások közé kell zárni az **** utasítást. (Ahol a *fájlnév.map* annak a fájlnek a neve, URL-je, amely a hivatkozásokat tartalmazza.)

Példa kép alkalmazására:

```
<MAP NAME="osztott">
```

```
<AREA SHAPE="circle" COORDS="130,130,50" HREF="#vege">
```

```
</MAP>
```

```
<P>
```

kép szövegbe beszúrva

```
<IMG SRC="k10.gif" ALT="valami kép" USEMAP="#osztott" ISMAP>
```

így jelenik meg

```
</P>
```

```
<P>
```

```
<IMG ALIGN="right" SRC="k10.gif" WIDTH="100" ALT="...">
```

Ugyanez jobbra igazítva és arányosan kicsinyítve...

```
</P>
```

```
<P>
```

```
<IMG ALIGN="left" SRC="k10.gif" VSPACE=20 HEIGHT="100" ALT="...">
```

Ugyanez balra igazítva és arányosan kicsinyítve...

```
</P>
```

```
<P>
```

```
<IMG ALIGN="top" SRC="k10.gif" HSPACE=40 WIDTH="100" ALT="...">
```

Ugyanez a szöveg felső széléhez igazítva...

```
</P>
```

```
<P>
```

```
<IMG ALIGN="middle" SRC="k10.gif" WIDTH="100" HEIGHT="100"
UNITS="en" ALT="...">
```

Ugyanez sor középre igazítva, valamint...

```
</P>
```

```
<P>
```

```
<IMG ALIGN="bottom" SRC="k.gif" WIDTH="200" HEIGHT="100"
UNITS="pixel" ALT="...">
```

Ugyanez a sor alsószéléhez igazítva, valamint...

```
<P>
```

```
<A NAME="vege">VÉGE</A>
```

Különböző listaformátumok a HTML dokumentumban

A HTML formátumú szövegfájlban használhatunk listákat is, amelyek szövegszerkesztőkbeli megfelelői a felsorolások és bajuszos bekezdések.

A számozott bekezdések (felsorolások) megfelelője a számozott lista, az ún. "bajuszos" bekezdések megfelelője pedig a számozatlan lista.

A harmadik lista típus pedig a leíró lista, ahol az egyes lista elemekhez tartozhat egy hosszabb leírás is.

A számozott listát az **** és az **** utasítások közé kell zárni. A számozatlan listát pedig az ****, **** utasításpár közé. Mindkét típusú listában használhatjuk a listafejléct, melyet az **<LH>** utasítás vezet be - az **</LH>** pedig zár! Mindkét listatípusban a listák sorai az **** utasítással kezdődnek és nem kötelező lezárni.

Számozott lista esetében a kezdő sorszám közvetlenül megadható az **<OL SEQNUM="szám">** formájú kezdőutasítással. Másik lehetőség, hogy egy előzőleg definiált lista számozása folytatható az **<OL CONTINUE>** kezdőutasítás használatával. Egyébként az **** utasítás 1-től kezdi a lista tagok számozását.

A számozatlan listák kezdőeleme is hordozhat formázóinformációkat. Az **<UL SRC="fájlnev.kit">** formájú utasítás például a listasort megelőző bajuszként a megadott képfájl használja. Az **<UL DINGBAT="karakter">** a megadott bajuszkaraktert alkalmazza. Az **<UL WRAP="irány">** pedig többoszlopos listák esetén az igazítás formáját határozza meg. (A **WRAP** opció a *horiz* és a *vert* értékeket veheti fel.

A számozatlan listák speciális - külön HTML utasításokkal létrehozható - fajtái a könyvtárlista és a menülista. A könyvtárlista típus a **<DIR>** utasítással kezdődik és a **</DIR>** utasításra végződik. A menülista pedig **<MENU>** és a **</MENU>** utasításokkal határolt. Ezek a listaformák a normál számozatlan listáktól mindössze annyiban különböznek, hogy a könyvtárlista tagjai 20 karakteresnél, a menülista tagjai pedig egy sorosnál nem lehetnek hosszabbak és nincs "bajuszuk".

A leíró listát a **<DL>** és a **</DL>** utasítások közé kell zárni. A lista fejléc megadása azonos az többi listatípussal látottal. A listák egyes alkotóelemeinek kezdetét a **<DT>** utasítás jelzi, az ehhez tartozó leírás kezdetét pedig a **<DD>** utasítás határozza meg. Nincs egyik utasításnak sem záró párja, ezért a lista tag a **<DT>** elemtől a **<DD>**-ig, a hozzá tartozó leírás pedig a **<DD>** elemtől a következő **<DT>**-ig tart.

Példa listákra:

<P>Normál szöveg</P>

<P>

<LH>A számozott lista fejléce</LH>

Első sor

<LH>A beágyazott lista fejléce</LH>

Első elem

Második elem

Harmadik elem

Második sor

Harmadik sor

Negyedik sor

</P>

<P>Normál szöveg</P>

<P>

<LH>A számozatlan lista fejléce</LH>

Első sor

Második sor

<UL wrap="horiz"><LH>A beágyazott lista fejléce</LH>

Első sor

Második sor

Harmadik sor

</P>

<P>Könyvtárlista:

<DIR>

Első tag

Második tag

Harmadik tag

Negyedik tag

</DIR>

</P>

<P>Menülista:

<MENU>

Első menü

Második menü

</MENU>

</P>

<P>Normál szöveg</P>

<P>

<DL><LH>A leíró lista fejléce</LH>

<DT>Első sor

<DD>Az első sorhoz tartozó leírás, lehet hosszabb

szöveg is.

A leírás tördelése automatikus. Szépen igazodnak a betördelt sorok az első sor kezdőpontjához.

<DT>Második sor

<DD>A második sorhoz tartozó leírás

</DL>

</P>

A HTML formátum táblázatai

A HTML formátumnak ez az utasításcsoportja képes a legváltozatosabb szöveg-, és képmegjelenítési formák előállítására. A **<TABLE>** és a **</TABLE>** utasítások közé zárt részt tekintjük egy táblázatnak.

A táblázatnak a címét a **<CAPTION>** és a **</CAPTION>** utasítások között kell megadni. (Figyelem! Az így megadott cím nem a táblázatban, hanem előtte fog megjelenni!) A cím **<CAPTION ALIGN="hely">** formájú megadással igazítható.

A táblázat minden sora a **<TR>** utasítással kezdődik és a következő **</TR>** vagy **<TR>**-ig, ill. a táblázat végéig tart. Egy sor tartalmazhat oszlopfejléceket és adatokat. Az oszlopfejléceket a **<TH>** **</TH>** utasítás vezeti be és választja el egymástól. A táblázat adatcellái pedig a **<TR>**-rel megkezdett sorban egy **<TD>** utasítással kezdődnek és minden cella a következő **<TD>**-ig - ill. **</TD>** ill. a következő sor elejét jelző elemig - tart, ahol értelemszerűen új cella kezdődik. Az oszlopfejléceknek és az adatcelláknak csak a kezdőutasítása használatos - habár van lezáró utasításuk is (**</TH>**, **</TD>**) -, mert a záróutasításuk elhagyható!

A táblázat nyitóutasítása tartalmazhat a teljes táblázatra vonatkozó beállításokat: **<TABLE BORDER="szám" ALIGN="hely" COLSPEC="oszlopjellemzők" UNITS="egység" NOWRAP CELLPADDING="pszám" CELLSPACING="kszám" BGCOLOR="színkód">**

Ahol a **BORDER** opció a rácsozat szélességét határozza meg. (0 esetén nincs rácsozat.) Az **ALIGN** a teljes tábla elhelyezkedését határozza meg (*left, right, center* lehet). A **COLSPEC** egy oszlop igazítását és szélességét adja meg. Egy oszlopra vonatkozóan egy betű és szám egybeírva (pl.: *L12 C24 R10*), melytől a következő oszlop értékeit egy köz választja el. Az **UNITS** a számokhoz tartozó mértékegységet jelöli ki (*en, relative* - oszlopszélességhez -, *pixel*). A **NOWRAP** opció a cellák szövegének tördelését tiltja le. Végül a **BGCOLOR** a táblázat háttérszínét határozza meg.

A táblázat oszlopfejlécei nem csak a legfelső oszlopban szerepelhetnek, hanem a táblázatban bárhol (pl. sorok címeiként is).

Mind az oszlopfejlécekben, mind az adatcellákban használhatók a következő formázásra való opciók:

COLSPAN="szám":

Egyesít több egymással szomszédos cellát - vízszintesen.

ROWSPAN="szám":

Egyesít több egymással alatti cellát - függőlegesen.

ALIGN="hely":

Igazítja a cellák tartalmát - vízszintesen. Lehetséges értékei: *left, center, right, justify, decimal*

VALIGN="hely":

Igazítja a cellák tartalmát - függőlegesen. Lehetséges értékei: *top, middle, bottom, baseline*

Példa táblázatok használatára:

```
<TABLE border="5" align="center">
  <CAPTION> A táblázat címe
</CAPTION>
  <TR>
    <TH colspan="2">      Az 1.-2. oszlop közös fejléce
    </TH>
    <TH colspan="3">      A 3.-4.-5. oszlop közös fejléce
    </TH>
    <TH rowspan="2">      A 6. (2 soros) oszlop fejléce
    </TH>
  </TR>
  <TR>
    <TH> Az 1. oszlop másodrendű fejléce
    </TH>
    <TH colspan="2">      A 2.-3. oszlop másodrendű fejléce
    </TH>
    <TH colspan="2">      A 4.-5. oszlop másodrendű fejléce
    </TH>
  </TR>
  <TR>
    <TH> Az első adatsor címe
    </TH>
    <TD> Az első adatcella
      <TD> Adat (indexe: C3)
      </TD>
      <TD> Adat (indexe: D3)
      </TD>
      <TD> Adat (indexe: E3)
      </TD>
      <TD> Adat (indexe: F3)
      </TD>
    </TD>
  </TR>
  <TR>
    <TH> A 2. adatsor címe
    </TH>
    <TD> Adat (indexe: B4)
    </TD>
    <TD> Adat (indexe: C4)
    </TD>
    <TD> Adat (indexe: D4)
    </TD>
    <TD> Adat (indexe: E4)
    </TD>
    <TD> Adat (indexe: F4)
    </TD>
  </TR>
  <TR>
    <TH> Függőleges igazítások
    </TH>
    <TD align="center" valign="bottom">      Le
      <TD align="center" valign="top">      Fel
      </TD>
      <TD align="center" valign="middle">      Középre
      </TD>
      <TD>
      </TD>
    </TD>
  </TR>
```

<TD> Adat (indexe: F5)
Ettől a cellától balra egy
üres cella,
alatta pedig két üres cella összevonva
</TD>

</TR>

<TR>

<TH> Vízszintes igazítások:

</TH>

<TD> Alapértelmezés

</TD>

<TD align="left"> Balra

</TD>

<TD align="center"> Középre

</TD>

<TD align="right"> Jobbra

</TD>

<TD rowspan="2">

</TD>

</TR>

<TR>

<TD align="center" colspan="5"><IMG src="k12.gif"
alt="Kép a cellában">

</TD>

</TR>

</TABLE>

A HTML dokumentumablak felosztása

Egyetlen böngészőablakban több HTML dokumentum is megjeleníthető a **<FRAMESET>** és a **</FRAMESET>** utasításpár, valamint a szorosan kapcsolódó **<FRAME>** utasítás együttes használatával.

A **<FRAMESET ROWS="oszlophatárok">** kezdőutasítással osztható fel a képernyő függőlegesen, a **<FRAMESET COLS="sorhatárok">** utasítással pedig vízszintesen. Ahol az oszlop- és sorhatárok megadhatók képernyőpontban ill. százalékosan - vesszővel elválasztva -, a maradék képernyőterületre pedig a * dzsókerekarakter használatával lehet hivatkozni. Mivel vagy csak vízszintesen, vagy csak függőlegesen osztható fel a képernyő, ezért ha mindkét irányban osztott böngészőablak létrehozásához a **<FRAMESET>** elemeket egymásba kell ágyazni! Tehát például egy függőleges felosztáson belül kell vízszintesen elválasztott részekre tagolni egy oszlopot.

A fenti módon definiált területekre a **<FRAME SRC="objektum">** utasítás tölti be a megadott objektumot, mely objektum lehet egy teljes HTML fájl, annak egy meghatározott része, ill. egy kép. Az így kitöltendő keretek viselkedését szabályozza az utasítás **<FRAME NAME="név" SRC="objektum" SCROLLING="érték" MARGINWIDTH="szám" MARGINHEIGHT="szám">** alakú megadása.

Az adott keretnek nevet ad a **NAME** opció, a szkrollozást letilthatja **SCROLLING="no"** kiegészítés (ezenkívül a *yes* és az *auto* értékeket veheti fel a **SCROLLING** opció), a **MARGINWIDTH** és a **MARGINHEIGHT** pedig a kereten belüli margók szélességét szabályozza.

Például a fejlécben megadott **<BASE TARGET="név">** utasítás a **NAME="név"** opcióval elnevezett keretbe irányítja a hivatkozásokat. Egyébként az **** utasítás is ismeri a **TARGET="név"** opciót. (A **TARGET="_top"** opcióval az egész böngészőablakot elfoglalja a hivatkozott dokumentum, tehát feloldja az ablak keretekre osztását!) Ha ezek egyike sem szerepel, akkor a hivatkozás a hivatkozó objektum keretében jelenik meg, felülírva azt!

A **<FRAMESET>**, **<FRAMESET>** utasításpárral határolt területnek meg kell előznie a **<BODY>** utasítással kijelölt dokumentumtörzset! Sőt a egy **<NOFRAMES>** utasítással kell jelezni a dokumentum azon részének kezdetét, amelyet akkor kell a böngészőnek megjelenítenie csak, ha nem ismeri a keretutasításokat. És csak ez a **<NOFRAMES>**-mel kezdődő rész tartalmazhatja a **<BODY>** és a **</BODY>** utasításpárt.

Példa keretek használatára:

```
<FRAMESET ROWS=185,*>
  <FRAMESET COLS=185,*>
    <FRAME SRC=k08.gif SCROLLING=NO NAME="cimer">
    <FRAME SRC=02.htm NAME="felepites">
  </FRAMESET>
  <FRAMESET COLS=25%,*>
    <FRAME SRC=index.htm NAME="tart">
    <FRAME SRC=13.htm NAME="keret" MARGINHEIGHT=10 MARGINWIDTH=10>
  </FRAMESET>
</FRAMESET>

<NOFRAMES>
<CENTER>
<BODY BGCOLOR="#FFFF00">
<FONT COLOR="#FF3333">
<H1>Sajnos ez a böngésző nem támogatja a keretek használatát!</H1>
</FONT>
</BODY>
```

Kérdőívek a HTML dokumentumban

A HTML formátumú dokumentumban kérdőíveket is közzétehetünk, melyek feldolgozásához külön programot kell írni. (Nem HTML-alapút! Általában valami server oldali program meghívása történik pl. ASP, CGI, PHP ...)

A **<FORM METHOD="mód" ACTION="elérési_út/fájlnév.kit">** és a **</FORM>** utasítások zárják közre a kitöltendő kérdőívet/űrlapot.

Az opciókat ajánlott mindig megadni, már csak azért is, mert az **ACTION** határozza meg a feldolgozást végző programot, a **METHOD** pedig a kitöltött űrlap továbbítási módját a feldolgozó programnak. Lehetséges értékei: *GET* - az URL-ben, *POST* - adatcsomagban. Az alapértelmezés a *GET*, ami bizonyos veszélyeket rejt magában, mivel túl hosszúra nyúlhat az URL. A *POST* a biztonságosabb mód.

Az **<INPUT NAME="név" TYPE="típus" ALIGN="hely">** utasítással határozható meg egy kitöltendő űrlapmező

A **NAME** természetesen a mezőnév, amely alapján a feldolgozóprogram azonosítja a bevitt adatot. A **TYPE** pedig az adattípus, melyet vár a beviteli mező. Lehetséges típusok: *TEXT* - szöveg, *PASSWORD* - jelszó (nem jelenik meg bevitelkor!), *HIDDEN* - rejtett (ez sem jelenik meg), *CHECKBOX* - kapcsoló (több is kiválasztható egyszerre), *RADIO* - kapcsoló (egyszerre csak egyet lehet kiválasztani), *RANGE* - numerikus adat, *FILE* - csatolandó fájl, *SUBMIT* - adattovábbító gomb, *RESET* - inicializáló gomb, *BUTTON* - egyéb nyomógomb. Az **<INPUT >** utasításban további opciók is szerepelhetnek, a fő opciók értékeitől függően:

- A **VALUE** kiegészítő opcióval megadott értéket veszi fel alapértelmezésként a szöveges vagy numerikus beviteli mező.
- *TEXT* típusú mező esetén egy további opció, a **SIZE="méret"** opció határozza meg a beviteli ablak szélességét, a **MAXLENGTH="érték"** pedig a bevihető maximális szöveghosszt.
- A *CHECKBOX* és a *RADIO* típusú mezők további paramétere lehet a **CHECKED** opció, mely bekapcsolja a kapcsolót - alapértelmezésként.
- *RANGE* típusú mező esetén megadható az a tartomány, melybe a bevitt értéknek bele kell esnie, a **MAX="maximum"** és a **MIN="minimum"** további opciókkal.
- A *FILE* típusú mezőben megadott fájl az **ACCEPT** kiegészítő opcióval megadott MIME módon csatolódik az elküldendő kérdőívhez. (Megjegyzés: Egy *Browse* nyomógommbal támogatott fájlkereső-ablakból lehet a fájl kiválasztani.)

- A *SUBMIT* és a *RESET* gombokhoz tartozó kiegészítő opció a **VALUE="felirat"**, amely a gombok feliratát jelöli ki. Egyébként a *SUBMIT* gomb lenyomásának hatására küldi el az űrlapadatokat a kérdőív a feldolgozó programnak, a *RESET* gomb lenyomása pedig az alapértékekkel tölti fel a beviteli mezőket.
- A *IMAGE* típussal készíthetünk egy képből gombot. A hozzá tartozó kiegészítő opció **SRC="elérési út/kép név"** segítségével adhatjuk meg a kép elérését.

Hosszabb szöveg bevitelére alkalmas a **<TEXTAREA NAME="név" ROWS="magasság" COLS="szélesség" VALUE="szöveg">**, **</TEXTAREA>** utasításpár, amely egy beviteli ablakot nyit a **COLS**-ban megadott szélességben és a **ROWS**-ban megadott sorban. A **VALUE** az alapértelmezésként megjelenítendő szöveget adja meg.

Egy kérdésre adandó válasz egyszerû - menüből történő - kiválasztását teszi lehetővé a kérdőíven a **<SELECT NAME="név" SIZE="sor">**, **</SELECT>** utasításokkal létrehozott kiválasztásos menü, melynek menüpontjait az **<OPTION>** utasítással adhatjuk meg.

A **SIZE** opció azt határozza meg, hogy hány sorban jelenjenek meg a választható menüpontok. Megadásával szkrollozható menüt kapunk. Elhagyása esetén, ún. legördülő menüből lehet választani. A **MULTIPLE** opció esetén több menüpont is kijelölhető egyszerre. Az **<OPTION SELECTED>** formájú utasítás adja meg az alapértelmezett választást!

Példa Űrlap kezelésre:

```
<CENTER><H1>Adatfelvételi lap:</H1>
  <FORM method="post" action="program.bin">
    <INPUT TYPE="reset" VALUE="Alapértelmezés">
<P>Vezetékeve:
  <INPUT NAME="vezeteknev" TYPE="text" VALUE="Kovács"
SIZE="25" MAXLENGTH="30">Keresztneve:
  <INPUT NAME="keresztnev" TYPE="text"
MAXLENGTH="50">Férfi:
  <INPUT NAME="neme" TYPE="radio" CHECKED>Nő:
  <INPUT NAME="neme" TYPE="radio">Kora:
  <INPUT NAME="kor" TYPE="range" SIZE="2" MIN="10"
MAX="60"></P>
<P>Érdeklődési köre:
  Windows:
    <INPUT NAME="erdek" TYPE="checkbox" CHECKED>Win95:

    <INPUT NAME="erdek" TYPE="checkbox">LINUX:
    <INPUT NAME="erdek" TYPE="checkbox">OS/2:
    <INPUT NAME="erdek" TYPE="checkbox"></P>
<P>
  <TEXTAREA name="egyeb" cols="40"
rows="4">Közlendők:</TEXTAREA></P>
<P>Foglalkozása:
  <SELECT name="foglakozas">
    <OPTION>diák
    <OPTION>tanár
    <OPTION selected>nyugdíjas
    <OPTION>egyik sem
  </SELECT></P>
<P>Csatolandó fájl(ok):<BR>
  <INPUT NAME="fajl" TYPE="file"></P>
<P>
  <INPUT TYPE="submit" VALUE="Elküldés"></CENTER>
</FORM></P>
```

A HTML dokumentum egyéb elemei

Egy HTML formátumú szövegfájl a tartalmazhat megjegyzéseket.

A megjegyzés egyik típusa a megjelenítendő megjegyzés, a **<NOTE>** és a **</NOTE>** utasításokkal közrezárva.

A HTML dokumentumban elhelyezhetők olyan megjegyzések is, melyek sehol sem jelennek meg a dokumentum WEB-böngészővel történő megjelenítésekor. Viszont a fájl átszerkesztéskor segítségül lehetnek a módosítást végzőnek. A megjegyzéseket a **<!--** és a **-->** utasítások között kell elhelyezni.

Egy HTML formátumú szövegfájl a tartalmazhat lábjegyzeteket. Az **<FN ID="azonosító">** és a **</FN>** utasítások között szerepel a lábjegyzet szövege. Az így definiált lábjegyzetszövegre hivatkozik a szövegnek az **** és az **** utasításokkal jelölt része.

Amennyiben a megjelenítendő szöveg formátuma pontosan olyan kell, hogy legyen, mint ahogy a HTML fájlban szerepel, akkor azt az előreformázott szöveget jelző utasítások közé kell zárni. Ezen utasítások a **<PRE>** és a **</PRE>**. A közéjük zárt szöveg pontosan annyi szóközzel, pontosan annyi sorban és olyan állapotban fog a dokumentumban megjelenni, mint ahogy azt a HTML fájl tartalmazza. E dokumentumsorozat példái is így módon kerültek rögzítésre ...

A **<PRE WIDTH="szám">**, **</PRE>** utasításpár használatával egy tördeletlen szöveg az adott szélességben betördelhető.

A szövegrészeket tagolás vagy esztétikai ok miatt vízszintes vonallal el lehet választani egymástól. Legegyszerűbb esetben a **<HR>** utasítás egy vízszintes elválasztó vonalat helyez el az adott ponton, a rendelkezésre álló szélességben. Ezt a durva megjelenítést lehet azért finomítani a **<HR ALIGN="hely" WIDTH="hszám" SIZE="vszám" NOSHADE>** alakú utasítással.

Az **ALIGN** az igazítás helyét adja meg (*left, right, middle*). A **WIDTH** a vonalhosszt definiálja, a **SIZE** pedig a vonal szélességét. Mindkettőt meg lehet adni képpontban, ill. a hosszt az ablak-szélesség százalékában. A **NOSHADE** pedig térhatást (árnyékoltságot) tiltja le.

Példa az egyéb elemekre:

```
<!-- Ez itt egy értelmezést segítő megjegyzés. -->
```

```
<NOTE>Ez itt egy jegyzet</NOTE>
```

```
Ehhez a sorhoz <A HREF="#az">lábjegyzet</A> tartozik.
```

```
<HR WIDTH=50 SIZE=50>
```

```
<PRE> Ez a sor sok közt tartalmaz. Ez pedig egy új sor, pedig nem előzte meg sem
```

```
<P>, sem <BR>
```

```
</PRE>
```

```
<HR ALIGN="left" WIDTH=50%> <P> <HR NOSHADE></P><P> <FN ID="az">Íme a fenti jelzéshez tartozó lábjegyzet</FN></P>
```

A JavaScript használata a HTML dokumentumban

Egy HTML formátumú szövegfájl a tartalmazhat JavaScript "programnyelven" megírt kódsorokat is **<SCRIPT LANGUAGE="JavaScript">** és a **</SCRIPT>** utasításokkal közrezárva. A JavaScript, mint programnyelv bemutatására nem kerül most sor. Ezért csak röviden:

A JavaScript változókat és függvényeket a dokumentum fejlécében szokás definiálni. (Vagyis a **<HEAD>** és a **</HEAD>** utasításokkal közrezárt részében a dokumentumnak. Az így definiált függvényeket lehet meghívni, a változókra lehet hivatkozni a szöveg HTML elemeiben.

Figyelem! Nem tévesztendő össze a JavaScript a JAVA programozási nyelvvel. A JAVA nyelven önálló programokat lehet írni, melyeket az **<APPLET CODE="osztály">** utasítással lehet meghívni.

JavaScriptről lásd. Külön fejezet.

Példa JavaScript alkalmazásra:

```
<HTML>
  <HEAD>
    <TITLE>HTML leírás - 17. lap</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
      function Ablak()
      {
        msg=open("", "DisplayWindow", "toolbar=no,directories=no,menubar=no,"
+ "resizable=no,width=300,height=200")
        msg.document.write("<BODY BGCOLOR=#009999>");
        msg.document.write("<CENTER><H2>Új böngészőablak</H2>");
        msg.document.write("<FORM>");
        msg.document.write("<INPUT TYPE='button' VALUE=' BEZÁR ÉS VISSZA '
'+ onclick='window.close()'>");
        msg.document.write("</FORM></CENTER>");
        msg.document.write("</BODY>");
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <CENTER>
      <H1>JavaScript példa</H1>
      <FORM>
        <INPUT TYPE="button" VALUE=' ÚJ ABLAK MEGNYITÁSA '
onclick="Ablak()">
      </FORM>
    </CENTER>
  </BODY>
</HTML>
```

A HTML speciális karakterei

Ha egy HTML formátumú szövegfájl nem csak az angol ABC alfanumerikus jeleit akarjuk használni, hanem ékezetes betűket vagy speciális jeleket is, akkor a HTML speciális jeleit kell használni. Lehetséges az ESCAPE szekvencia alapján történő jelölése és ISO-kód szerinti megadása is ezen speciális jeleknek.

Az ABC betűi és kódjaik							
Elnevezés	Jel	ESC	ISO	Elnevezés	Jel	ESC	ISO
Nagy A betű	A	A	A	Kis a betű	a	a	a
Nagy Á betű	Á	Á	Á	Kis á betű	á	á	á
Nagy B betű	B	B	B	Kis b betű	b	b	b
Nagy C betű	C	C	C	Kis c betű	c	c	c
Nagy D betű	D	D	D	Kis d betű	d	d	d
Nagy E betű	E	E	E	Kis e betű	e	e	e
Nagy É betű	É	É	É	Kis é betű	é	é	é
Nagy F betű	F	F	F	Kis f betű	f	f	f
Nagy G betű	G	G	G	Kis g betű	g	g	g
Nagy H betű	H	H	H	Kis h betű	h	h	h
Nagy I betű	I	I	I	Kis i betű	i	i	i
Nagy Í betű	Í	Í	Í	Kis í betű	í	í	í
Nagy J betű	J	J	J	Kis j betű	j	j	j
Nagy K betű	K	K	K	Kis k betű	k	k	k
Nagy L betű	L	L	L	Kis l betű	l	l	l
Nagy M betű	M	M	M	Kis m betű	m	m	m
Nagy N betű	N	N	N	Kis n betű	n	n	n
Nagy O betű	O	O	O	Kis o betű	o	o	o
Nagy Ó betű	Ó	Ó	Ó	Kis ó betű	ó	ó	ó
Nagy Ö betű	Ö	Ö	Ö	Kis ö betű	ö	ö	ö
Nagy O betű	O	Õ	Ô	Kis õ betű	o	õ	ô
Nagy P betű	P	P	P	Kis p betű	p	p	p
Nagy Q betű	Q	Q	Q	Kis q betű	q	q	q
Nagy R betű	R	R	R	Kis r betű	r	r	r
Nagy S betű	S	S	S	Kis s betű	s	s	s
Nagy T betű	T	T	T	Kis t betű	t	t	t
Nagy U betű	U	U	U	Kis u betű	u	u	u
Nagy Ú betű	Ú	Ú	Ú	Kis ú betű	ú	ú	ú
Nagy Ü betű	Ü	Ü	Ü	Kis ü betű	ü	ü	ü
Nagy Û betű	Û	Û	Û	Kis û betű	û	û	û
Nagy V betű	V	V	V	Kis v betű	v	v	v
Nagy W betű	W	W	W	Kis w betű	w	w	w
Nagy X betű	X	X	X	Kis x betű	x	x	x
Nagy Y betű	Y	Y	Y	Kis y betű	y	y	y
Nagy Z betű	Z	Z	Z	Kis z betű	z	z	z

Néhány speciális jel és kódjaik								
Elnevezés	Jel	ESC	ISO		Elnevezés	Jel	ESC	ISO
Tabulátor						Soremelés		
	
Szóköz		 			Felkiáltójel	!	!	!
Idézőjel	"	"	"		Számjel	#	#	#
Dollárjel	\$	$	\$		Százalékjel	%	%	%
Angol és jel	&	&	&		Aposztróf	'	'	&
Bal zárójel	(((Jobb zárójel)))
Aszteriszk	*	*	*		Pluszjel	+	+	+
Vessző	,	,	,		Kötőjel	-	-	-
Pont	.	.	.		Perjel	/	/	/
Nullás	0	0	0		Egyes	1	1	1
Kettes	2	2	2		Hármas	3	3	3
Négyes	4	4	4		Ötös	5	5	5
Hatos	6	6	6		Hetes	7	7	7
Nyolcas	8	8	8		Kilences	9	9	9
Kettőspont	:	:	:		Pontosvessző	;	;	;
Kisebb jel	<	<	<		Egyenlőségjel	=	=	=
Nagyobb jel	>	>	>		Kérdőjel	?	?	?
Kukac	@	@	@		Bal zárójel	[[[
Visszaper jel	\	\	\		Jobb zárójel]]]
Hatványjel	^	^	^		Aláhúzás	_	_	_
Vissza aposztróf	`	`	`		Bal kapocs	{	{	{
Függőleges		|			Jobb kapocs	}	}	}
Tilde jel	~	~	~		Alsó aposztróf	,	‚	
Alsó idézőjel	“	„			Kereszt	†	†	
Kettős kereszt	‡	‡			Ezrelék	‰	‰	
Felső vessző	‘	‘			Felső vessző	’	’	
Kettős vessző	“	“			Kettős vessző	”	”	
Szorzás jel	•	•			Mínusz előjel	–	–	
Kivonás jel	—	—			Trade Mark	™	™	
Cent jele	¢	¢	¢		Font jele	ℒ	£	£
Csővezeték jel	¡	¦	¦		Paragrafus jel	§	§	§
Umlaut	¨	¨	¨		Copyright	©	©	©
Bal tört idézet	«	«	«		Lágy kötőjel		­	­
Registered TM	®	®	®		Fok jele	°	°	°
Plusz-mínusz	±	±	±		Ékezet	´	´	´
Mikro	μ	µ	µ		Bekezdés vége	¶	¶	¶
Középen pont	·	·	·		Jobb tört idézet	»	»	»
Szorzás kereszt	×	×			Scharfes s	ß	ß	ß
Osztás jel	÷	÷						

Az alábbi kis táblázat tartalmazza a magyar ékezetes betűk Escape-szekvenciáit és ISO-kódját:

Jel	ESCAPE	ISO-kód	Jel	ESCAPE	ISO-kód
Á	Á	Á	á	á	á
Ê	É	É	é	é	é
Í	Í	Í	í	í	í
Ó	Ó	Ó	ó	ó	ó
Ô	Õ	Ô	ô	õ	ô
Ö	Ö	Ö	ö	ö	ö
Ú	Ú	Ú	ú	ú	ú
Û	Û	Û	û	û	û
Ü	Ü	Ü	ü	ü	ü

Karaktertáblák

Az Internet használatának első éveiben, az aktuális feladatok megoldására a 7 bites ASCII kódolás tökéletesen elegendő volt. A 7 bitbe (127 karakter) belefért a teljes angol ABC, a számok és még néhány jel is. Azután a hálózat kezdte átlépni a határokat, és egyre inkább szükség volt az angoltól különböző nyelvek betűinek megjelenítésére is. Eközben a hétbites rendszerekről lassan megkezdődött az áttérés a nyolc bitre (így lett az ASCII-ből ANSI), ezután kézenfekvő volt, hogy a speciális karaktereket a megjelenő 128 üres helyre lehet bekódolni. Ahány ház, annyi szokás: ki így, ki úgy töltötte ki ezt a felső tartományt. A különböző megoldásoknak köszönhetően megszülettek a kódtáblák (codepage), és elkezdődött a káosz. Még messze a DOS-os időkben járunk, amikor a táblázatrajzoló karakterek helyett néha ékezetes karakterek jelennek meg a rossz beállításoknak köszönhetően. A nyugat-európai (western) kódtáblák tartalmazznak ugyan ékezetes betűket, de a magyar hosszú ö és û már nem fért bele. Sebaj, van hasonló: (sajnos) valószínűleg mindannyian találkoztunk már a kalapos u[^] és hullámos o[~] betűkkel. Akkoriban ez a kis csúsztatás még elviselhetőnek tűnt, manapság viszont már inkább kínos hibának tűnik a megjelenésük. A világ ugyanis azóta sokat fejlődött. A szerteágazó kódtáblákat szabványokká fogták össze, a dokumentumokba beépítették a karaktértáblákat azonosító adatokat, így egy HTML oldal forrásából, vagy egy e-mailből azonnal kiderül, hogy azt a küldője milyen karaktértáblával írta. Bizony, így van ez akkor is, ha a (főleg nem Windows platformon futó) levelezőprogramok zöme erről nem hajlandó tudomást venni. Mindenekelőtt két fontos kódtáblára hívnám fel a figyelmet: az iso-8859-1, más néven Western kódtábla a nyugat-európai karaktereket (és a hullámos/ kalapos o[~]-t és u[^]-t) tartalmazza, míg az iso-8859-2 alias Central European amely tartalmazza a magyar változatot. Hasonló hatást lehet elérni a Windows-1250 nevű kódlappal, ez azonban, mint az a nevéből is kitalálható, nem kifejezetten elterjedt Un*x és Macintosh körökben :-). Magyar szövegben, amikor csak tehetjük, használjuk az iso-8859-2-t!

Cascading Style Sheets (CSS)

Bevezetés

A CSS egy olyan stíluslap megvalósítás, amely lehetővé teszi, hogy a HTML oldalak szerzői oldalaikhoz egyedi stílust rendeljenek hozzá. A CSS egyik alapvető tulajdonsága a folyamatos stíluslap - HTML lap kapcsolat. A lapok szerzői az általuk kedvelt stílust **egyszer** rögzítik, és hozzákapcsolhatják minden általuk készített HTML laphoz. Ez a leírás tartalmazza ennek

megoldási lehetőségeit. Stíluslappal (style sheet) tervezni nem nehéz, de szükséges hozzá némi alapvető HTML ismeret. Ezt szemlélteti az alábbi példa:

```
H1 {color: blue}
```

Fenti példa tartalmazza a CSS használatának alapszabályait - **egy css utasítás két részből áll:**

? a **szelektor** tartalmazza a formázandó HTML tag megnevezését (H1);

? a **deklaráció** végzi el a szelektorban meghatározott tag formázását.

Maga a deklaráció is két részre bontható: egy tulajdonságra és a hozzá tartozó értékre (szín: kék). A szelektor a tulajdonképpeni kapocs a HTML, mint leírónyelv és a stíluslap között; szinte minden HTML tag betöltheti a szelektor szerepét. Az előbb említett **szín (color)** tulajdonság csak egy a több, mint 50 közül, amelyek segítségével alakíthatjuk egy HTML oldal kinézetét. (A továbbiakban ahol **oldal, dokumentum** olvasható, értelemszerűen **HTML** oldalra, dokumentumra vonatkozik.)

A stíluslap csatolása

A következő példa mutatja be a kapcsolódás négy lehetséges módját:

```
<HTML>
  <HEAD>
    <TITLE>Stíluslapok</TITLE>
    <LINK REL=STYLESHEET TYPE="text/css" HREF="http://style.com/sajat"
TITLE="sajat">
    <STYLE TYPE="text/css">
      @import url(http://style.com/sajat)
    <!--
      H1 {color: blue}
    -->
    </STYLE>
  </HEAD>

  <BODY>
    <H1>A címsor kék</H1>
    <P STYLE="color: green">Az egész bekezdés zöld</P>
  </BODY>
</HTML>
```

A fenti példában látható első lehetőség a csatolásra a <LINK> tag használata; külső stíluslap behívására. Második lehetőségként a dokumentum HEAD szekciójában elhelyezett <STYLE> tag, ahol közvetlenül definiálhatók a használni kívánt stílusok, vagy az '@import' kulcsszóval külső stíluslap importálható. Az utolsóként bemutatott lehetőség pedig valamely HTML elem STYLE attribútumának használata, a dokumentum BODY szekciójában.

A böngészők általában figyelmen kívül hagyják az általuk ismeretlen elemeket. Ezért a régebbi böngészők jó esetben egyszerűen 'elmennek' a <STYLE> elem mellett. Kellemetlenebb eredménnyel jár, ha belezavarodnak tartalmába. Ez megelőzhető, ha egy standard SGML utasítással elrejtjük előlük:

```
<STYLE TYPE="text/css">
  <!--
    H1 {color: blue}
  -->
</STYLE>
```

Csoportosítás

A stíluslapok méretének csökkentése érdekében a szelektorok csoportosíthatók; vesszővel elválasztott listába szedve:

```
H1, H2, H3 {font-family: verdana}
```

Hasonló módon a deklarációk is csoportosíthatók:

```
H1
{
```



```
font-family: helvetica;
font-size: 12pt;
font-style: normal;
}
```

Néhány tulajdonság eltérő csoportosítási szintaktikát is megenged:

```
H1 {font: bold 12pt helvetica}
```

Öröklődés

Az első példában a H1 elem színbeállítását mutattuk be. A következőkben tartalmaz egy elemet is:

```
<H1>A címsor <EM>mindig</EM> fontos.</H1>
```

Ha az elemhez nincs külön szín rendelve, a kiemelt (emphasized) "mindig" szó *mindig* öröklí a tartalmazó (container) elem színét, jelen esetben a kéket. Hasonlóképpen más stílus-tulajdonságok is öröklődnek (pl.:font-family, font-size). Alapértelmezett stílustulajdonság beállításához az alkalmazni kívánt stílust olyan elemhez kell kötni, amely tartalmazza mindazokat az elemeket, amelyekre a stílust vonatkoztatni akarjuk. A HTML dokumentumokban erre a célra megfelelhet a <BODY> elem:

```
BODY
{
  color: black;
  background: url(hatter.gif) white;
}
```

Ez a stíluspélda a BODY szövegszínét feketére állítja be, háttérként képet rendel hozzá. Ha a kép nem érhető el, a háttérszín fehér lesz. Van azonban néhány stílustulajdonság, amely nem öröklődik (Ezek közé tartozik pl. a background), de a szülő elem háttértulajdonságát néhány *tartalmazott* (contained) elem láthatóan hagyja. (Ha egy táblázatnak nem adunk meg 'background' /háttér/ tulajdonságot, átlátszik rajta a BODY háttere.)

A tulajdonságok értéke százalékban is megadható:

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

Ebben a példában a 'line-height' tulajdonság értéke a 'font-size' tulajdonság értékének 120% -a lesz: 12 pont.

A class szelektor

A HTML elemek stílusbeállítási rugalmasságának növelése érdekében a W3Cúj attribútumot vett fel a HTML-be: ez a **CLASS** (osztály). A 'BODY' minden eleme osztályba sorolható, az egyes osztályok pedig stíluslapból megcímezhetők.

```
<HTML>
<HEAD>
<TITLE>Stíluslap példa</TITLE>
<STYLE TYPE="text/css">
H1.mezei {color: #00ff00}
</STYLE>
</HEAD>
<BODY>
<H1 CLASS=mezei>Zöld, mint a rét</H1>
</BODY>
</HTML>
```

Az osztályba sorolt elemekre az öröklődés általános szabályai vonatkoznak. Öröklí a stílusértékeket a dokumentum struktúrájában felettük álló ún. *szülő* elemektől. Minden azonos osztályba tartozó elem megcímezhető egyszerre is, elhagyva a hozzájuk tartozó tag nevét:

```
.mezei {color: green}
/*minden mezei osztályba tartozó elem (CLASS=mezei)*/
```

Megjegyzendő, hogy szelektoronként (HTML elemenként) csak egy osztály definiálható!

Az ID szelektor

A HTML -be felvételre került az 'ID' attribútum is, amely lehetővé teszi az egyedi azonosítók felvételét a dokumentumba. Ennek a lehetőségnek különleges jelentőséget ad, hogy felhasználható stíluslap szelektorként, és megcímezhető a '#' előtaggal.

```
#z98y {letter-spacing: 0,3em}
H1#z98y {letter-spacing: 0,5em}
<P ID=z98y>Széles szöveg</P>
```

Az fenti példa első esetében a kiválasztott formázást a 'P' elemhez feleltettük meg, az 'ID' attribútumérték segítségével. A második esetben kettős feltételt támasztottunk: a formázás akkor lép érvénybe, ha a H1 elemet a '#z98y' azonosítóval (ID) látjuk el. Ezért ez már nem vonatkozik a 'P' elemre. Az ID attribútum szelektorként való használatával HTML elemekre alapozott stílustulajdonságok állíthatók be.

Összekapcsolt szelektorok

Az öröklődés szabályai mentesítik a stíluslap tervezőjét egy csomó fölösleges gépelés alól. A tulajdonságok beállítása során elég egyszer elkészíteni az alapértelmezettet, utána felsorolni a kivételeket. Ha az 'EM' elemnek a 'H1' elemen belül más színt szeretnénk meghatározni, azt a következőképp tehetjük meg:

```
H1 {color: blue}
EM {color: red}
```

Mikor a stíluslap aktív, minden kiemelt () szövegrész, akár a H1 elemen belül, akár azon kívül található, vörösre változik. Abban az esetben, ha csak egyszer akarjuk a H1 -en belül az EM elemet vörösre színezni, a CSS kódot az alábbiak szerint kell megváltoztatni:

```
H1 EM {color: red}
```

Összekapcsolt szelektorok használata esetén azok az elemek lesznek megcímezve, amelyek a felsorolásban utoljára állnak. Tehát akár kettőnél több elem is 'egymásba ágyazható' ilyen módon.

Megjegyzések

A CSS kódban elhelyezett megjegyzések hasonlóak a C programnyelvben elhelyezett megjegyzésekhez: A megjegyzések nem ágyazhatók egymásba, illetve más CSS utasításba.

```
EM {color: red} /* A kiemelt szövegrész vörös!*/
```

Látszólagos osztályok és elemek

A CSS-ben a beállítandó stílus alapesetben egy HTML elemhez van kapcsolva; ez a kapcsolat az elemnek a dokumentum-struktúrában elfoglalt helyére alapozódik. Ez az egyszerű modell a stíluslapalkalmazás viszonylag széleskörű lehetőségét nyújtja, de nem nyújt lehetőséget az összes lehetséges megjelenítés alkalmazására.

A látszólagos osztályok és elemek a HTML leírásban nem szerepelnek (ezért látszólagosak), mégis köthetők szelektorokhoz. Tulajdonképpen a böngésző által átadott és a stíluslapon keresztül értelmezett címzési módról van szó. Ugyanúgy kell rájuk hivatkozni, mint bármely elemre, vagy osztályra; ez a szabványos eljárás viselkedésük leírására. Pontosabban: viselkedésük tagek elméleti sorozataként írható le.

A látszólagos elemek az elemek részlemeinek megcímezésére használhatók, a látszólagos osztályok pedig lehetővé teszik a stíluslapon keresztül történő elemtípus megkülönböztetést.

Látszólagos osztályok az élőkapcsokban

A böngészők közös tulajdonsága, hogy másképp jelenítik meg a látogatott linkeket, mint a még nem látogatottakat. Ezt a tulajdonságot a CSS az <A> elem látszólagos osztályain keresztül kezelni tudja:

```
A: link {color: red}
A: visited {color: blue}
A: active {color: lime}
```

Minden 'HREF' attribútummal rendelkező <A> elem a fenti csoportból egyet és egy időben csak egyet jelöl ki. A böngészők pedig kiválaszják, hogy az adott linket -állapotától függően- milyen színnel jelenítsék meg. Állapotukat a látszólagos osztály határozza meg:

? **link** - Nem látogatott hivatkozás;

? **visited** - Már látogatott hivatkozás;

? **active** - Amelyik hivatkozás éppen ki van választva (egérkattintással).

Egy élőkapocs látszólagos osztályának formázása ugyanúgy történik, mintha az osztály külön volna definiálva. A böngészők nem követelik meg az aktuálisan megjelenített dokumentum újrabetöltését, amikor egy élőkapocs látszólagos osztálya által meghatározott változtatás esedékessé válik. (Pl.: a CSS szabványos eljárása lehetővé teszi az 'active' link 'font-size' tulajdonságának futásidejű megváltoztatását úgy, hogy az aktív dokumentumot nem kell újra betöltenie a böngészőnek, mikor az olvasó kiválaszt egy 'visited' linket.). A látszólagos osztályok nem feleltethetők meg a normál osztályoknak és fordítva; ezért az alábbi példában bemutatott stílusszabályok nem befolyásolják egymást:

```
A: link {color: red}
<A CLASS=link NAME=target5>...</A>
```

Az élőkapocs látszólagos osztályoknak nincs hatásuk az 'A' -n kívül más elemre. Ezért az elemtípus el is hagyható a szelektorból:

```
A: link {color: red}
:link {color: red}
```

Fenti két deklarációban a szelektor ugyanazt az elemet fogja kiválasztani. A látszólagos osztályok nevei kis- és nagybetűérzékenyek. A látszólagos osztályok használhatóak a kapcsolódó szelektorokban is:

```
A: link IMG {border: solid blue;}
```

A látszólagos osztályok kombinálhatók a normál osztályokkal:

```
A.external: visited {color: blue}
<A CLASS=external HREF="http://valahol.mashol.com">Külső (external)
hivatkozás</A>
```

Ha a fenti példában levő hivatkozás látogatottá válik (visited), színe kékre változik. Megjegyzendő, hogy a normál osztályok neveinek a látszólagos osztályok neveit meg kell előznie a szelektorból.

Tipografikai látszólagos elemek

Néhány közös megjelenítési effektus nem strukturális elemhez kapcsolható, hanem inkább a képernyőn elemeket kirajzoló tipografikai tulajdonságokhoz. A CSS -ben két ilyen tipografikai tétel címezhető meg látszólagos elemen keresztül: egy elem tartalmának első sora és az első betű.

A 'first-line' látszólagos elem

A 'first-line' látszólagos elem az első sor különleges formázásához használható:

```
<STYLE TYPE="text/css">
P:first-line {font-variant: small-caps}
</STYLE>
```

A tagek elméleti sorozata a következőképp néz ki:

```
<P>
```

```
<P:first-line>
A szöveg első sora
</P:first-line>
kiskapitális betűkkel jelenik meg.
</P>
```

A 'first-line' látszólagos elem használata hasonló a soron belüli elemekhez, azonban figyelembe kell venni néhány megszorítást. Csak a következőkben felsorolt tulajdonságok alkalmazhatók hozzá:

- ? Betűtípus tulajdonságok
- ? Szín- és háttér tulajdonságok
- ? 'word-spacing',
- ? 'letter-spacing',
- ? 'text-decoration',
- ? 'vertical-align' (csak, ha a 'float' tulajdonság értéke 'none';),
- ? 'text-transform',
- ? 'line-height',
- ? 'clear'.

A 'first-letter' látszólagos elem

A 'first-letter' látszólagos elem gyakran előforduló használati lehetősége az iniciálé kialakítása, ami gyakran használt tipográfiai effektus. A következőkben felsorolt tulajdonságok alkalmazhatók hozzá:

- ? Betűtípus tulajdonságok,
- ? Szín- és háttér tulajdonságok,
- ? 'text-decoration',
- ? 'vertical-align' (csak, ha a 'float' tulajdonság értéke 'none';),
- ? 'text-transform',
- ? 'line-height',
- ? margó tulajdonságai,
- ? helykitöltő (padding) tulajdonságok,
- ? szegélytulajdonságok,
- ? 'float',
- ? 'clear'.

A következő példa bemutatja, hogyan készíthető kétsoros iniciálé:

```
<HTML>
<HEAD>
<TITLE>Lapcím</TITLE>
<STYLE TYPE="text/css">
P: {font-size: 12pt; line-height: 12pt}
P:first-letter: {font-size: 200%; float:left}
</STYLE>
</HEAD>
<BODY>
<P>
A sor első betűje kétszer akkora lesz, mint a többi.
</P>
</BODY>
</HTML>
```

A böngészőtől függ, mely karakterek tartoznak a 'first-letter' elemhez. Általában a bevezető idézőjelet is belefoglalják. A 'first-letter' látszólagos elem csak blokk szintű elemhez kapcsolható.

Látszólagos elemek a szelektorokban

Kapcsolódó szelektorok esetén a látszólagos elemeknek a szelektor utolsó elemeként kell szerepelniük:

```
BODY P:first-letter {color: purple}
```

A látszólagos elemek kombinálhatóak a szelektorban az osztályokkal is:

```
P.alap:first-letter {color: red}
```

```
<P CLASS=alap>Első bekezdés</P>
```

A fenti példában látható stílusmeghatározás az összes olyan P elem első betűjét bíborra színezi, amelynek osztálya 'alap' (class=alap). Ha látszólagos elemeket osztályokkal, vagy látszólagos osztályokkal kombinálunk, a látszólagos elemet a szelektor utolsó tagjaként kell elhelyezni. Egy szelektorban csak egy látszólagos elem lehet elhelyezve.

Látszólagos elemek többszörözése

Néhány látszólagos elem kombinálható:

```
P {color: red; font-size: 12pt}
```

```
P:first-letter {color: green; font-size: 200%}
```

```
P:first-line {color: blue}
```

A fenti példa minden 'P' elem első betűjét zöldre, a betűméretet pedig 24 pontosra állítja. Az első sor többi része kék lesz, a többi része pedig vörös.

Megjegyzendő, hogy a 'first-letter' elemet a 'first-line' elem tartalmazza. A 'first-line' elemre beállított tulajdonságokat a 'first-letter' elem örökli, de az öröklődés szabályánál a 'first-letter' elemre külön beállított tulajdonság-érték erősebb.

Rangsor

Ebben a fejezetben a terminológia egy új elemét kell bevezetnünk: a "súly". A stíluslap-szabályok súlya jelzi, hogy az adott szabály milyen prioritást élvez. Természetesen a nagyobb súlyú szabály erősebben érvényesül, mint az alacsonyabb súllyal rendelkező.

CSS használatával egyidejűleg egynél több stíluslap is befolyásolhatja a HTML oldal megjelenését. E tulajdonságnak két fő oka van: a modularitás és a szerző (tervező) / olvasó - egyensúly.

? **Modularitás** A stíluslap tervezője a redundancia csökkentése érdekében több stíluslap használatát kombinálhatja:

```
@import url(http://www.style.org/egyik);
```

```
@import url(http://www.style.org/masik);
```

```
H1 {color: red} /* Felülbírálja az importált stílust */
```

? **Szerző / olvasó egyensúly:** Stíluslappal a szerző és az olvasó is befolyásolhatja a dokumentum megjelenését. Ehhez mindkettejüknek ugyanazt a stílusnyelvet kell használniuk, így tükrözve a web egyik alapvető tulajdosságát: mindenki közzéteheti elképzelését. A böngészők a saját stíluslapokra hivatkozás kiválasztását szabadon lehetővé teszik.

Néha ellentét merül fel a dokumentum megjelenését meghatározó stíluslapok között. Az ellentét feloldását a stílusszabályok súlyozása teszi lehetővé. Alapértelmezésben az olvasó által felállított stílusszabályok súlya kisebb, mint a dokumentum szerzője által felállítottaké. Tehát, ha ellentét merül fel egy dokumentum szerzői és olvasói stíluslapja között, a szerzői stíluslap kerül alkalmazásra. Mind a szerzői, mind az olvasói stíluslap-szabályok felülbírálják a böngésző alapértelmezett beállításait.

Az importált stíluslapok szintén jól meghatározott rangsorban állnak egymással. A rangsor az alább meghatározott szabályok szerint dől el. Bármely szabály, amely a stíluslapban van leírva, erősebb, mint az importált stíluslap(ok)ban levő(k). Tehát, az importált stíluslapokban leírt szabályok súlya kisebb a stílusok rangsorában, mint a stíluslapban magában leírt

szabályoké. Az importált stíluslapok maguk is rekurzívan importálhatnak és ezáltal felülbírálnak más stíluslapokat.

A CSS minden `@import` utasításának a stíluslap legelején kell megjelennie, megelőzve minden más deklarációt. Ez megkönnyíti annak átláthatóságát, hogy melyik stíluslap melyik másikat bírálja felül.

'important'

A stíluslapok tervezői deklarációik súlyát megnövelhetik:

```
H1 {color: black ! important; background: white ! important }
P {font-size: 12pt ! important; font-style: italic }
```

Ebben a példában az első három deklaráció súlya a *fontos!* kiemeléssel meg van növelve, míg az utolsó deklaráció súlya normál. Egy olvasó által felállított *fontos* szabály prioritásban megelőzi a szerző normál súlyú szabályát. A szerző által felállított *fontos* szabály prioritásban megelőzi az olvasó által felállított *fontos* szabályt.

A rangsor felállítása

A deklaráció-rangsor felállításának szabályai lényegesek a CSS működésében. Egy elem/tulajdonság páros értékének meghatározásához az alábbi algoritmust kell követni:

1. A kérdéses elem/tulajdonság párosra alkalmazott összes deklaráció előkeresése. A deklaráció akkor 'alkalmazott', ha a kérdéses elem szelektorként szerepel. Ha nincs az elemhez alkalmazott deklaráció, az öröklés szabályai érvényesülnek. Ha nincs örökölt érték (pl.: a HTML elem esetében), a kezdeti értékek lesznek irányadók.
2. A deklarációk explicit súlya szerinti rendezés: az `!important` jelölésű deklarációk erősebbek, mint a jelöletlen (normál) deklarációk.
3. Eredet szerinti rendezés: A szerző stíluslapjában meghatározott szabályok mint az olvasó által meghatározottak; mindkettő erősebb a böngésző alapértelmezett beállításainál. Az importált stíluslapok eredete megegyezik annak a stíluslapnak az eredetével, ahonnan importálták.
4. Szelektor egyedisége szerinti rendezés: Az egyedileg meghatározott szelektorhoz tartozó szabály erősebb, mint az általános meghatározásban leírtak. Az egyediség megállapításához meg kell állapítani a szelektorban található ID attribútumokat (a), a CLASS attribútumokat (b), és a tag-nevek számát (c). A három szám 'összeláncolásával' (concatenating) állapítható meg az adott szelektor egyedisége. A könnyebb megértés kedvéért néhány példa:

LI	{...}	/*	a=0	b=0	c=1	=>	egyediség =	1	*/
UL LI	{...}	/*	a=0	b=0	c=2	=>	egyediség =	2	*/
OL UL LI	{...}	/*	a=0	b=0	c=3	=>	egyediség =	3	*/
LI.red	{...}	/*	a=0	b=1	c=1	=>	egyediség =	11	*/
OL UL LI.red	{...}	/*	a=0	b=1	c=3	=>	egyediség =	13	*/
#x34y	{...}	/*	a=1	b=0	c=0	=>	egyediség =	100	*/

A látszólagos elemeket és látszólagos osztályokat a számítás során normál elemekként, osztályokként kell figyelembe venni.

5. Rendezés a meghatározás sorrendje szerint: Ha két szabály ugyanakkora súllyal bír, a később meghatározott győz. Az importált stíluslapban leírt szabályok a saját lapban írtak után lesznek csak figyelembe véve.

A tulajdonság-értékek keresése megszakítható, ha az egyik kiértékelte szabály súlya egy elem/tulajdonság vonatkozásban egyértelműen nagyobb bármely más szabályénál.

Egy elem `STYLE` attribútumában történő deklarációnak ugyanolyan súlya van, mint egy - a stíluslap végén meghatározott - ID alapú szelektornak.

```
<STYLE TYPE="text/css">
```

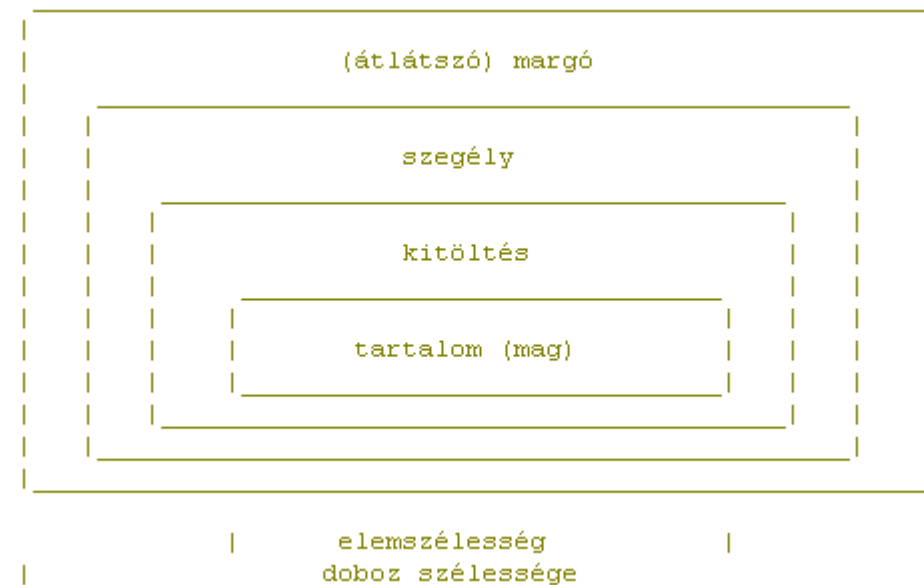
```
#x97z {color: blue}
</STYLE>
```

```
<P ID=x97z STYLE="color: red">
```

A fenti példában a P elem színe vörös (red) lesz. Bár mindkét deklarációs forma egyedisége megegyezik, mégis - a rangsor-meghatározás 5. pontjában írt szabály alkalmazása miatt - a STYLE attribútumban levő deklaráció erősebb lesz a STYLE elem által tartalmazott deklarációnál.

Formázásmodell

A CSS egy egyszerû, dobozszerû formázási modellt használ., ahol minden elemformázás eredménye egy, vagy több négyzetgletes dobozként képzelhetõ el. Minden doboznak van egy



'magja', az öt körülvevő 'kitöltéssel' (padding), szegéllyel (border) és margóval (margin).

A margó, a szegély és a kitöltés mérete egyenként meghatározható a **margin**, a **border** és a **padding** tulajdonságok értékeinek beállításával. A kitöltőterület háterszíne megegyezik az elemével, amelyet a **background** tulajdonság határoz meg. A szegély színe és stílusa szintén a **border** tulajdonság beállításával határozható meg, míg a margó *mindig* átlátszó, így a szülő elem állandóan látható marad. A doboz szélessége az *elem*, a *kitöltés*, a *szegély* és a *margó* területének összege. A formázások szempontjából két elemtípus különböztethető meg: a >blokk szintű elem és a soron belüli elem.

Blokk szintű elemek

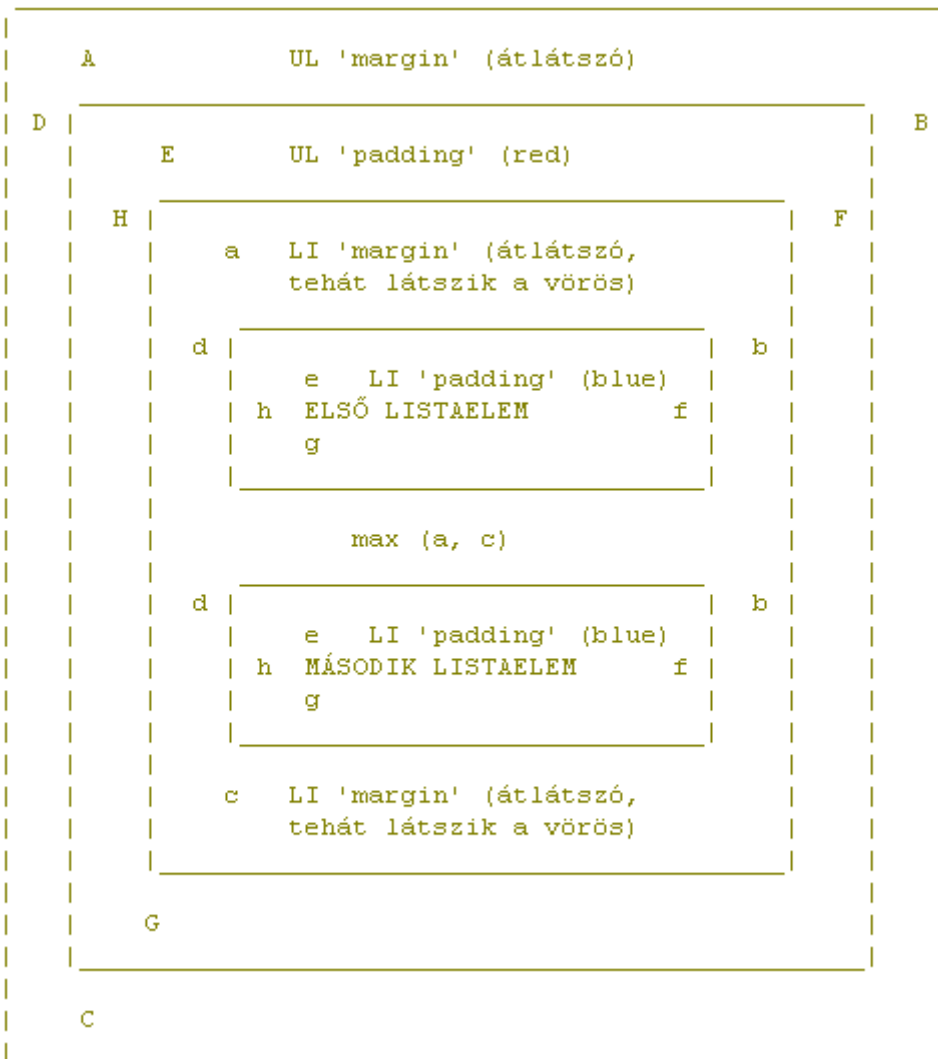
A következő példa bemutatja, hogyan formázható CSS használatával egy két gyermek-elemet tartalmazó **UL** elem. A példa egyszerűsítése okán szegélyeket nem definiálunk. Szintén ez okból használjuk értékeként az ABC kis- és nagybetűit, amelyek nem szabványos CSS meghatározások, de ebben az esetben használatuk megkönnyíti a táblázat és a példastílus összevetését.

```
<STYLE TYPE="text/css">
  UL {
    background: red;
    margin: A B C D;
```

```

padding: E F G H;
}
LI {
color: white;
background: blue;          /* fehér szöveg kék háttéren */
margin: a b c d;
padding: e f g h;
}
</STYLE>
...
<UL>
  <LI>Első elem
  <LI>Második elem
</UL>

```



Gyakorlatilag, a kitöltés és a margó tulajdonságai nem öröklődnek. De, ahogy a példa is mutatja, az elemek szülőkhöz és 'testvérekhez' (velük egyenrangúakhoz) viszonyított elhelyezkedése lényeges, mert ezeknek az elemeknek kitöltés- és margótulajdonságai hatást gyakorolnak a gyermek elemekre. Ha az előző példában lettek volna szegélyek is, azokat a kitöltés és a margó között kellett volna megjeleníteni.

A következő ábra a dobozmodell használatakor érvényes terminológiáról nyújt áttekintést:

Vízszintes formázás

A nem 'lebegő' blokkszintű elemek vízszintes pozícióját és méretét hét tulajdonság határozza meg: a 'margin-left', 'border-left', 'padding-left', 'width', 'padding-right', 'border-right', 'margin-right'. Ezen tulajdonságok értékeinek összege mindig megegyezik a szülő elem 'width' értékével. Alapértelmezésben egy elem szélességének értéke 'auto'. Ha az elem nem helyettesített elem, a szélesség értékét a böngésző számolja ki, a fent említett hét tulajdonság értékének összegzésével. Ha az elem helyettesített elem, 'width' tulajdonságának 'auto' értéke automatikusan kicserélődik az elem saját szélesség-értékére. A hét tulajdonság közül háromnak értéke állítható 'auto' -ra: 'margin-left', 'width', 'margin-right'. A 'width' tulajdonságnak van egy nemnegatív, böngésző által meghatározott minimum értéke (amely elemenként változik és mindig másik tulajdonságtól függ). Ha a 'width' értéke e határérték alá kerül, akár azért, mert így lett beállítva, vagy mert értéke 'auto' volt és az alábbi szabályok miatt vált értéke túl alacsonnyá: értéke kicserélődik a minimumként használható értékkel. Ha *pontosan egy* 'margin-left', 'width', vagy 'margin-right' értéke 'auto'; a böngésző ennek értékét úgy állítja be, hogy kiadja a fentebb felsorolt hét tulajdonsághoz tartozó maradék értéket; amennyi szükséges a szülő elem szélességének eléréséhez. Ha a tulajdonságok közül *egyik sem* 'auto'; a 'margin-right' tulajdonság értéke lesz 'auto'. Ha a háromból *egynél több* 'auto' és közülük egyik a 'width'; a többi tulajdonság ('margin-left' és/vagy 'margin-right') értéke nullára lesz állítva, és a 'width' kapja meg azt az értéket, amennyi szükséges a szülő elem szélességének eléréséhez. Abban az esetben, ha a 'margin-left', és 'margin-right' tulajdonságok vannak 'auto' -ra állítva, mindkettő egyenlő értéket kap. A függőleges margókkal ellentétben, a vízszintes margók nem összevonhatók.

Listaelemek

Azok az elemek, amelyek 'list-item' tulajdonságának értéke 'display', blokkszintű elemekként formáznak, és listajelölő előzi meg őket. A jelölő típusát a **list-style** tulajdonság határozza meg. A **list-style** tulajdonság értéke szerint a jelölő elhelyezkedése az alábbi lehet:

```
<STYLE TYPE="text/css">
  UL          {list-style: outside}
  UL.compact {list-style: inside}
</STYLE>

<UL>
  <LI>Első listaelem tartalma
  <LI>Második listaelem tartalma
</UL>

<UL CLASS="compact">
  <LI>Első listaelem tartalma
  <LI>Második listaelem tartalma
</UL>
```

A fenti példát a böngésző így jeleníti meg:

```
* Az első listaelem
  tartalma
* A második listaelem
  tartalma

* Az első listaelem
  tartalma
* A második listaelem
  tartalma
```

Jobbról-balra értelmezendő szövegek esetében értelemszerűen a listajelölő a doboz jobb oldalára kerül.

Lebegő elemek

A 'float' tulajdonság alkalmazásával egy elem elhelyezkedése meghatározható az elemek által általában elfoglalt helyek oldalsó vonalán kívül is; ez esetben formázásuk módja a blokkszintű elemekéhez hasonló. Példaként: ha egy kép 'float' tulajdonságát 'left' (bal) értékre állítjuk be, eredményképpen a kép addig tolódik balra, míg egy másik blokkszintű elem margóját, kitöltését, vagy szegélyét el nem éri. Jobb oldalán a lap többi tartalma fogja körbefutni. Az elem margói, szegélyei, kitöltései megmaradnak, de margói a szomszédos elem margóival *sosem* lesznek összevonva. A lebegő elemek pozicionálása a következő szabályok szerint történik

1. Egy balra igazított lebegő elem bal külső éle nem lehet balrább, mint szülő eleme bal belső éle. A jobbra igazított lebegő elemekre ugyanilyen szabály érvényes.
2. Egy balra igazított lebegő elem bal külső élének jobbra kell esnie bármelyik (a HTML forrásban) korábban leírt balra igazított lebegő elemtől, vagy tetejének kell lejjebb lennie, mint az előbb leírt aljának. A jobbra igazított lebegő elemekre ugyanilyen szabály érvényes.
3. Egy balra igazított lebegő elem jobb külső éle nem eshet jobbra bármely tőle jobbra eső, jobbra igazított lebegő elem bal külső élétől. A jobbra igazított lebegő elemekre ugyanilyen szabály érvényes.
4. A lebegő elem teteje nem lehet magasabban, mint szülő elemének belső magassága (inner top).
5. A lebegő elem teteje nem lehet magasabban, mint bármely korábbi lebegő, vagy blokkszintű elem teteje.
6. Egy lebegő elem teteje nem lehet magasabban, mint bármely 'sordoboz', amelynek tartalma a HTML forrásban megelőzi a lebegő elemet.
7. A lebegő elemet olyan magasra kell elhelyezni, amennyire csak lehet.
8. A balra igazított lebegő elemet annyira balra kell elhelyezni, amennyire csak lehet; a jobbra igazított lebegő elemet pedig annyira jobbra kell elhelyezni, amennyire csak lehet. Ez az igazítási mód előnyben részesül a többi (más) balra / jobbra igazítási eljárással szemben.

```
<STYLE TYPE="text/css">
  IMG {float: left}
  BODY, P, IMG {margin: 2em}
</STYLE>

<BODY>
  <IMG SRC="kep.gif">
  Példa szöveg, melynek nincs más szerepe, ...
</BODY>
```

A fenti példaszöveg formázása a következőképp alakul:

max(BODY margin, P margin)				
				Példaszöveg, melynek nincs
B	P	IMG margók		más szerepe, minthogy
O				bemutassa, hogy a lebegő
D	m			elem a szülő elem szélére
Y	a	IMG		igazodik, miközben meg-
	r			tartja annak margóját,
m	g			szegélyét és kitöltését.
a	ó			Megjegyzendő, hogy a szom-
r				szédos függőleges margók
g				összevonódnak a nem-lebegő
ó				blokkszintű elemek között.

Megjegyzés: a lebegő **IMG** elemet a **P** elem tartalmazza, tehát jelen esetben az a szülő elem. Van két olyan eset, amikor egy lebegő elemek margó-, szegély-, és kitöltő területei átfedhetik egymást:

- ? Amikor a lebegő elemnek negatív értékű margója van: a lebegő elem negatív margója úgy lesz figyelembe véve, mint más blokkszintű elemé.
- ? Amikor a lebegő elem szélesebb, vagy magasabb, mint az őt tartalmazó elem.

Soron belüli elemek

Azok az elemek, amelyek nem formázhatók blokkszintű elemként; a soron belüli elemek. Egy soron belül használható több soron belüli elem is. Figyeljük meg a példát:

```
<P>Néhány <EM>kiemelt</EM> szó jelenik meg a <STRONG>sorban</STRONG>
```

A **P** elem általánosan használt blokkszintű elem, míg az **EM** és a **STRONG** soron belüli elemek. Ha a **P** elem szélessége egy egész sort kitölt, akkor abban a sorban két soron belüli elem is található lesz.

Néhány *kiemelt* szó jelenik meg a **sorban**

Ha a **P** elem szélessége több, mint egy sort foglal el, a soron belüli elem 'doboz' felosztódik a két sor között:

```
<P>Néhány <EM>kiemelt szó jelenik</EM> meg; a sorban
```

Néhány *kiemelt* szó
jelenik meg a sorban.

Ha egy soron belüli elemhez margó, szegély, kitöltés, vagy szövegdekoráció van rendelve, ezek nem lesznek hatással a sortörésre.

Helyettesített elemek

A helyettesített elem olyan elem, amely megjelenítésekor avval a tartalommal van helyettesítve, amelyre hivatkozásként mutat. Például a egy HTML kódban szereplő **IMG** elem az **SRC** attribútumában szereplő képpel lesz megjelenésekor helyettesítve. Ilyenkor a helyettesített elemek magukkal hozzák saját méreteiket is. Ha a **width** értéke **auto**, az elem szélességét megjelenítésekor saját szélessége határozza meg. Ha stíluslapon keresztül más (nem **auto**) érték van meghatározva, a helyettesített elem azzal az értékkel kerül megjelenítésre. (A méretezési eljárás a média típusától függ.) A **height** tulajdonságra hasonló szabályok vonatkoznak. Helyettesített elemek lehetnek soron belüli, vagy blokkszintű elemek is.

Sorok magassága

Minden elemnek van **line-height**, azaz sormagasság tulajdonsága, amely egy szövegsor teljes magasságát adja meg. A szövegsor teljes magasságát a szöveg és az alá-fölé elhelyezett üres helyek összessége adja meg. Példaként: ha egy szöveg 12pt magas, és a **line-height** értéke 14pt; két képpontnyi üres hely adódik hozzá a betűmérethez: egy képpont a sor fölé, egy képpont alá.

A betűméret és a sormagasság közti különbséget *vezetésnek* nevezzük. A vezetés-érték fele a *fél-vezetés*. Formázáskor minden szövegsor egy-egy *sordoboz*ként értelmezhető.

Ha a szövegsor egyes részei különböző magasság-értékeket tartalmaznak (több különböző soron belüli elem), akkor minden egyes résznek megvan a maga saját vezetés-értéke is. A sordoboz magasságát a legmagasabb rész tetejétől a legmélyebbre érő rész legaljáig mérjük. Megjegyzendő, hogy a 'legmagasabb' és a 'legmélyebb' értékek nem szükségszerűen tartoznak ugyanazon elemhez, mivel az elemek függőleges pozicionálása a **vertical-align** tulajdonság beállításával történik. Egész bekezdés formázása esetén a minden sordoboz közvetlenül az előző alá kerül. A nem helyettesített elemekhez tartozó alsó és felső kitöltések, szegélyek és margók nem befolyásolják a sordoboz magasságát. Más szóval: ha a kiválasztott kitöltéshez, vagy szegélyhez tartozó **line-height** tulajdonság túl kicsi, a szomszédos sorok szövegei rá fognak takarni.

A sorban található helyettesített elemek (pl.: képek) megnövelhetik a sordoboz magasságát, ha a helyettesített elem teteje (beleértve a hozzá tartozó kitöltést, szegélyt, margót is) a legmagasabb szövegrész fölé, illetve alja a legalacsonyabbra érő szövegrész alá ér.

Általános esetben, amikor amikor az egész bekezdésnek egy **line-height** értéke van, és nincsenek nagyobb méretű képek, a fenti szabályleírás biztosan meghatározza, hogy a párhuzamos sorok alapvonalai pontosan a **line-height** tulajdonságban megadott értékre legyenek egymástól.

A Vászon

A Vászon (CANVAS) a böngészőfelület azon része, amelyre a dokumentum és beágyazott elemei kirajzolódnak. A vászon *nem* a dokumentum strukturáló elemeinek felel meg, és ez a dokumentum formázása során két fontos kérdést vet fel.

? A beállítás során honnan számítjuk a vászon méreteit?

? Amikor a dokumentum nem fedi le az egész vászont, az üres terület hogyan lesz kirajzolva?

Az első kérdésre adható válasz az, hogy a vászon kezdeti szélessége a böngészőablak méretén alapul, de a CSS a pontos méretbeállítást a böngészőre hagyja. Ésszerű a böngészőre hagyni a vászon méretének megváltoztatását az ablak átméretezésekor is, mert ez szintén kívül esik a CSS hatókörén.

A HTML kiterjesztései szolgáltatnak példát a második kérdés megválaszolásához: A **BODY** elem attribútumai állítják be a háttérét; az egész vászonra vonatkozóan. A laptervezők elvárásainak támogatására a CSS-ben bevezetésre került egy, a vászon háttérére vonatkozó szabály:

Ha a **HTML** elem **background** értéke nem **transparent**, akkor az kerül használatba, egyébként pedig a **BODY** elem **background** attribútuma.

Fenti szabály használat közben a következőképp jelenhet meg:

```
<HTML STYLE="background: url(http://style.com/hatter.jpg)">
```

```
<BODY STYLE="background: red">
```

A példában a vászont betéríti a 'hatter.jpg' nevű kép. A **BODY** elem háttérszíne pedig (amely nem biztos, hogy teljesen lefedi a vászont), vörös lesz.

CSS tulajdonságok

A stíluslapok a dokumentumok megjelenését a stílustulajdonságokhoz hozzárendelt értékekkel befolyásolják. Ez a fejezet bemutatja a definiált stílustulajdonságokat, és az azokhoz rendelhető értékeket.

A tulajdonság-érték párok jelölési rendszere

A következőkben a tulajdonságokhoz tartozó lehetséges értékeket az alábbi jelrendszer és szintaktika szerint fogjuk bemutatni:

Érték: N | NW | NE

Érték: [<hossz> | vastag | vékony]{1,4}

Érték: [<family-name>,]* <family-name>

Érték: <url>?<szín> [/<szín>]?

Érték: <url> || <szín>

A '<' és '>' jelek között szereplő szavak jelentik az érték *típusát*. A leggyakrabban használt közös típusok közé tartozik a <hossz>, a <százalék>, az <url>, a <szám> és a <szín> (length, percentage, url, number, color). A többi specializált típus (pl.: font-family, vagy border-style) leírása a hozzájuk tartozó tulajdonságnál található.

A kulcsszavak betű szerint szerepelnek, idézőjelek nélkül. A törtjelet (/) és a vesszőt (,) szintén ugyanoda és ugyanúgy kell elhelyezni, ahova és ahogyan a szabályok előírják.

Az egymás mellé írt kifejezések azt jelentik, hogy mindegyikük használható, a mutatott sorrendben. Függőleges vonal (A/B) jelzi az alternatívákat: a felsoroltak közül *egy* fordulhat elő. Kettős függőleges vonal (A//B) jelzi, hogy a felsoroltak közül vagy A, vagy B, vagy mindkettő előfordulhat, tetszőleges sorrendben. A szögletes zárójelek ([]) a csoportosításra utalnak. A tulajdonságok egymás mellé helyezése erősebb, mint a függőleges vonal; a kettős függőleges vonal erősebb, mint a függőleges vonal. Így az "a b/c//d e" kifejezés megegyezik az "[a b]//[c]//[d e]" kifejezéssel. Minden típust, kulcsszót, vagy zárójeles csoportosítást követhet egy, a következő módosítók közül:

- ? Csillag (*) jelzi, hogy az előzőekben írt típus, szó, vagy csoport 0 (nulla), vagy annál több esetben ismételhető.
- ? Plusz jel (+) jelzi, hogy az előzőekben írt típus, szó, vagy csoport 1, vagy több esetben.
- ? Kérdőjel (?) jelzi, hogy az előzőekben írt típus, szó, vagy csoport használata opcionális.
- ? Kapcsos zárójelbe írt számpár ({A,B}) jelzi, hogy az előzőekben írt típus, szó, vagy csoport legalább (A), de legfeljebb (B) számú esetben ismételhető.

Font tulajdonságok

A stíluslapok leggyakoribb használata a fonttulajdonságok beállítása. Balszerencsére, nem létezik általános érvényű és széles körűen elfogadott módszer a betűtípusok osztályozására, nincs terminológia, ugyanúgy alkalmazhatnánk a különböző betűtípus-családokra. (az 'italic' jelző általánosan használt a dőlt szöveg jelzésére, de a dőlt betűk jelezhetőek az *Oblique*, *Slanted*, *Incline*, *Cursive*, vagy *Kursiv* címkekkel is.) Ezért nem egyszerű probléma általánosan használható tulajdonságokat hozzárendelni egy meghatározott betűtípushoz. A CSS-ben a *font-family*, *font-style*, *font-variant*, *font-weight*, *font-size* és *font* tulajdonságok vannak leírva.

Betűtípusok megfeleltetése

Mivel nem létezik általános érvényű és széles körűen elfogadott módszer a betűtípusok osztályozására, a tulajdonságok és betűmegjelenítések összepárosítása fokozott óvatosságot kíván. A tulajdonságok megfeleltetése jól definiált sorrendben történik, hogy biztosítva legyen a konzisztencia a megfeleltetési művelet végeredménye és a böngésző képességei között.

1. A böngésző minden általa ismert betűtípushoz, készít (vagy megnyit) egy adatbázist, amely tartalmazza az alkalmazható CSS tulajdonságokat. A böngésző ismeri az elérhető betűtípusokat, mivel azok a helyi számítógépre vannak telepítve. Ha két betűtípus is megfelel ugyanannak a tulajdonságnak, a böngésző az egyiket figyelmen kívül hagyja.
2. Egy adott elemnél, és az elem minden karakterére vonatkozóan a böngésző összegyűjti az elemhez alkalmazható betűtípus-tulajdonságokat. A teljes tulajdonság-készlet használatához a böngésző a *font-family* tulajdonságot használja, hogy kiválasszon egy (ideiglenes) próba-fontcsaládot. A fennmaradó tulajdonságokat pedig a minden tulajdonságra leírt megfeleltetési kritérium szerint teszteli. Ha az összes fennmaradó tulajdonságot sikerült megfeleltetnie, akkor megtalálta a megfelelő font-kinézetet az adott elemhez.
3. Ha nincs a 2. lépésnek megfelelő font-kinézet a *font-family* tulajdonság értékei között, de van alternatív fontcsalád az elérhető betűkészletben, a második lépésben leírt megfeleltetési eljárás azzal folytatódik.
4. Ha van megfelelő kinézetű font, de a készlet nem tartalmazza az ábrázolandó karaktert, és van következő alternatív font-család; a második lépés ismétlődik.
5. Ha nincs megfelelő font a 2. lépésben kiválasztott betűkészlet-családban, a böngészőtől függő alapértelmezett betűkészlet-család kerül használatba, és a böngésző azon belül ismétli meg a 2. lépést, amíg végül megtalálja az előírtak legjobban megfelelőt.

A tulajdonságonkénti megfeleltetés szabályai (2. lépés) a következők:

1. Az első keresési feltétel a *font-style*. (Az '*italic*' keresési feltétel akkor megfelelő, ha vagy van a böngésző font-adatbázisában, amely megfelel a CSS '*italic*', vagy '*oblique*' kulcsszava által meghatározottaknak.)
2. A következő megfeleltetési feltétel a '*font-variant*'. '*normal*' az a font, amely nincs '*small-caps*'-ként megjelölve. '*small-caps*' az a font, amely (1) így van megjelölve, (2) az a font, amelyből a '*small-caps*'-et előállítják, vagy (3) az a font, amely esetében kisbetűk nagybetűkkel vannak helyettesítve. Kiskapitális betűtípust elő lehet állítani elektronikus úton is, normál fontból, a nagybetűk átméretezésével.
3. A következő megfeleltetési feltétel a '*font-weight*'; ez sosem okozhat hibát (lásd a '*font-weight*' tulajdonságot, lejjebb).
4. A '*font-size*' tulajdonság megfeleltetése a böngésző tűréshatárán történik. (Általában, a méretezhető fontok méretei a legközelebbi egész számú pixel értékére vannak kerekítve; a bittérképes fontok megjelenítésének tűrése 20% körül van.)

'font-family'

Értéke lehet: `[(<family-name>|<generic family>),]*(<family-name>|<generic family>)`

Alapértelmezés: Böngészőfüggő.

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

Értéke elsőbbségi listája a fontcsalád neveknek, és/vagy az általános fontcsalád-neveknek. Eltérően a legtöbb más CSS tulajdonságtól, az értékeket vesszővel kell elválasztani, így jelezve, hogy azok alternatívák.

BODY {font-family: gill, helvetica, sans-serif}

A felsorolásban használható típusok:

<family-name>

A választott fontcsalád neve. Fenti példában fontcsalád a 'gill' és a 'helvetica'.

<generic-family>

Fenti példában az utolsó érték az általános fontcsalád-név. A következő általános fontcsalád-nevek vannak definiálva:

- ? serif; (pl: Times)
- ? sans-serif (pl: Helvetica)
- ? cursive (pl: Zapf-Chancery)
- ? fantasy (pl: Western)
- ? monospace (pl: Courier)

Az általános fontcsaládok, mint végső eset jöhetnek számításba. A szóközöket is tartalmazó fontneveket idézőjelbe kell tenni:

```
BODY {font-family: "new century schoolbook", serif}
```

'font-style'

Értéke lehet:

normal | italic | oblique

Alapértelmezés: normal

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A 'font-style' tulajdonság használatával lehet egy fontcsaládból kiválasztani a normál, vagy dőlt betűs megjelenést (az 'italic' és az 'oblique' is dőlt betűt eredményez).

Az értékek azokat a betűtípusokat választják ki a böngésző fontadatbázisából, amelyek az értéknek megfelelően vannak megjelölve. Ha 'italic' jelölésű font nem elérhető, helyette az 'oblique' kerül használatba. Az 'oblique' jelölésű fontokat a böngésző futásidőben is generálhatja; egy normál (álló) font megdöntésével.

```
H1, H2, H3 {font-style: italic}
```

```
H1 EM {font-style: normal}
```

Fenti példában a H1 elemen belül szereplő kiemelt (EM) szöveg normál betűkkel jelenik meg.

'font-variant'

Értéke lehet:

normal | small-caps

Alapértelmezés: normal

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A fontcsaládon belüli megjelenítésváltozatok másik típusa a kiskapitális betűtípus. A kiskapitális megjelenítésnél a kibetűk hasonlítanak a nagybetűkhöz, csak méretük kisebb; arányaikban jelentéktelen a különbözőség.

A 'normal' érték a betűtípus normál alakját, a 'small-caps' a kiskapitális alakot választja ki. A CSS-ben elfogadható (de nem megkövetelt), hogy a kiskapitális betűk kisbetűi a normál fontkészlet nagybetűinek futásidejű átméretezésével legyenek kialakítva.

A következő példában található utasítások eredményeképpen a H3 elembe levő szöveg kiskapitális betűkkel lesz megjelenítve:

```
H3 {font-variant: small-caps}
```

'font-weight'

Értéke lehet:

normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700
| 800 | 900

Alapértelmezés: normal

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A 'font-weight' tulajdonság választja ki a betűtípus súlyát (megjelenését).

Az angol eredetiben a "weight" szó szerepel, a magyarban nincs igazán megfelelő **egy** szó az itt alkalmazott jelentésre: a betűtípus vastagsága, színintenzitása. Ezért megmaradtam a szó szerinti fordításnál: súly.

Alkalmazható értékei 100 - 900 -ig terjedhetnek, ahol minden egyes szám egy súlyt jelez, amelyek mindegyike sötétebb az előzőnél. A `normal` kulcsszó a '400'-as, a `bold` a 700 -as értéknek felel meg.

```
P {font-weight: normal}      /* 400 */
H1 {font-weight: 700}       /* bold */
```

A `bolder` és `lighter` értékek a szülő elemtől örökölt súlyhoz viszonyítva választják ki az alkalmazandó súlyt.

```
STRONG {font-weight: bolder}
```

A további gyermek-elemek az így eredményként kapott értéket öröklik, nem az eredetit.

A fontoknak (font adatoknak) jellemzően van egy, vagy több tulajdonságuk, amelyeknek értékei a font "súlyát" leíró nevek. Ezeknek a neveknek nincs széleskörűen elfogadott, általános érvényű jelentésük. Feladatuk elsősorban az, hogy súly szerint megkülönböztessék a fontcsaládon belül a különböző kinézetű fontokat. Használatuk különböző fontcsaládok között meglehetősen változó. Az a font, amit **bold**nak (félkövérnek) gondolhatnánk, az leírása szerint lehet, hogy *Regular*, *Roman*, *Book*, *Medium*, *Semi-*, vagy *Demi-bold*, *Bold*, vagy *Black*, attól függően, mennyire sötét a 'normal' font az adott fontcsaládban. Mivel a neveknek nincsenek mértékadó használati előírásaik, a CSS `weight` tulajdonság-értékei egy numerikus skálán vannak megadva, ahol a 400-as, vagy `normal` érték jelenti a fontcsalád "normal" értékét. Ez a súlynév van hozzárendelve ahhoz a kinézetű fonthoz, amelynek neve általában *Book*, *Regular*, *Roman*, illetve ritkábban *Medium*.

A fontcsaládon belül más súlyok hozzárendelése a numerikus értékekhez az árnyékolási (sötétítési) sorrend megtartásával történik. A következő műveleti sorrend megmutatja, hogy egy tipikus esetben hogyan történik a súlyok számértékhez rendelése.

- ? Ha a fontcsalád már használ kilencértékű numerikus skálát, a hozzárendelés közvetlenül történik.
- ? Ha a fontcsaládban a `medium` érték és a *Book*, *Regular*, *Roman*, *Medium* értékek más súlyt jelentenek, a `medium` értékhez az 500 lesz hozzárendelve.
- ? A `bold`ként jelölt font leggyakrabban a 700-as értéket kapja.
- ? Ha a fontcsaládban kevesebb, mint kilenc súlyérték van, a "lyukak" kitöltésére a következő algoritmus használatos: Ha az '500'-as érték hozzárendelés nélküli, ahhoz a fonthoz lesz hozzárendelve, amelyikhez a '400'. Ha a '600', '700', '800', vagy '900'-as értékek valamelyike marad hozzárendelés nélkül, ahhoz a fonthoz lesznek hozzárendelve, amelyiket a következő sötétebbnek megfelelő kulcsszó kijelöl. Ha a '300', '200', vagy '100'-as értékek valamelyike marad hozzárendelés nélkül, ahhoz a fonthoz lesznek hozzárendelve, amelyiket a következő világosabbnak megfelelő kulcsszó kijelöl.

A következő példa mutatja ennek végrehajtását. A 'Család 1' fontcsaládban négy súlyt feltételezünk, a világosabbtól a sötétebb felé: *Regular*, *Medium*, *Bold*, *Heavy*. A 'Család 2' fontcsaládban 6 súly van: *Book*, *Medium*, *Bold*, *Heavy*, *Black*, *ExtraBlack*. Figyeljük meg, hogy a 'Család 2 ExtraBlack' fonthoz nem rendelünk értéket.

Elérhető kinézetek	Hozzárendelt érték	Kitöltött "lyukak"
Család 1 Regular	400	100, 200, 300
Család 1 Medium	500	
Család 1 Bold	700	600
Család 1 Heavy	800	
Család 2 Book	400	100, 200, 300

Család 2 Medium	500	
Család 2 Bold	700	600
Család 2 Heavy	800	
Család 2 Black	900	
Család 2 ExtraBlack	(nincs)	

Mivel a **bolder** és **lighter** relatív kulcsszavak célja a *családon belül* a fontok sötétebben, vagy világosabban történő ábrázolása, és mert a fontcsaládoknak nincs valamennyi súlyértékhez fontja hozzárendelve, a **bolder** kulcsszó hatására a fontcsalád következő sötétebb, a **lighter** kulcsszó hatására a fontcsalád következő világosabb tagja kerül használatba. A pontosság kedvéért a **bolder** és **lighter** relatív kulcsszavak jelentése a következő:

- ? A **bolder** azt a súlyt választja ki, amelyik sötétebb, mint az eredeti örökölt érték. Ha nincs megfelelő, ez egyszerűen a következő numerikus értéket jelenti (és a font változatlan marad). Ha az örökölt érték '900' volt, az eredményül kapott súly is '900' lesz.
- ? A **lighter** hasonlóképpen jár el, csak ellenkező irányban. A következő világosabb súlyértékhez tartozó kulcsszót választja ki; ha nincs megfelelő, a következő világosabb számértéket (és a megjelenített font változatlan marad).

Nincs garantálva, hogy minden font-weight érték hatására az alkalmazott font sötétebb lesz; a fontcsaládok egy részében csak normál és félkövér típus van, míg más fontcsaládok nyolc különböző súlyértékkel rendelkeznek. Nincs garantálva az sem, hogy a böngészők helyesen rendelik egymáshoz az ugyanahhoz a fontcsaládhoz tartozó fontokat a megfelelő súlyértékekkel. Az egyetlen garantálható következmény az, hogy egy adott értéknél a font nem lesz kevésbé sötét, mint egy világosabb értéknél.

'font-size'

Értéke lehet: <abszolút méret> | <relatív méret> | <hossz> | <százalék>

Alapértelmezés: medium

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Viszonyítási alap a szülő elem fontmérete

<abszolút méret>

Az abszolút méret egy indexet jelent a böngésző által nyilvántartott fontméret-táblázaton.

Lehetséges értékei:

[xx-small | x-small | small | medium | large | x-large | xx-large].

A szomszédos indexek közötti különbség megjelenítéskori javasolt értéke 1,5-szörös; ha a 'medium' 10 pontos méretet jelent, a 'large' 15 pontos lesz. A különböző médiatípusokhoz különböző méretezési faktorok szükségesek. A fontméret-táblázat a különböző fontcsaládok esetén különbözhet egymástól.

<relatív méret>

A relatív méret a fontméret-táblázat és a szülő elem valós fontmérete közötti arányt veszi figyelembe. Lehetséges értékei:

[larger | smaller]

Ha a szülő elem fontmérete 'medium' volt, a larger érték az aktuális elem fontméretének értékét large -ra állítja. Ha a szülő elem fontméretét a böngésző fontméret-táblázata nem tartalmazza, a böngésző helyettesítheti a kívánt méretet a sorban következővel. A hossz és százalékos értékek nem férhetnek hozzá a fontméret-táblázathoz az elem aktuális fontméretének kiszámításakor. Negatív értékek használata nem megengedett. A többi tulajdonságnál, az 'em' és 'ex' méretértékek az aktuális elem fontméretére hivatkoznak. A font-size tulajdonság esetén azonban ezek a mértékegységek a szülő elem fontméretét veszik alapul. Példák:

```
P {font-size: 12pt;}
```

```
BLOCKQUOTE {font-size: larger}
```

```
EM {font-size: 150%}
```

EM {font-size: 1,5em}

Ha a javasolt 1,5-szörös méretezési faktor van használatban, az utolsó három deklaráció - eredményét tekintve- megegyezik.

'font'

Értéke lehet:

[<font-style> || <font-variant> || <font-weight>]?<font-size> [/<line-height>]?<font-family>

Alapértelmezés: Nincs definiálva

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Értelmezhető a font-size és a line-height tulajdonságokhoz.

A font tulajdonság használata gyors elérési lehetőséget biztosít afont-style, font-variant, font-weight, font-size, line-height, font-family tulajdonságok beállításához. Használata a hagyományos rövidutas tipográfiai jelrendszer szabályain alapul.

A lehetséges és alapértékek beállításához lásd az előző bekezdésekben leírt szabályokat. Azon tulajdonságok esetében, ahol érték nincs beállítva, azok alapértelmezett értékei kerülnek használatba.

P { font: 12pt/14pt sans-serif }

P { font: 80% sans-serif }

P { font: x-large/110% "new century schoolbook", serif }

P { font: bold italic large Palatino, serif }

P { font: normal small-caps 120%/120% fantasy }

A második szabályban a százalékos méretérték a szülő elem fontméretét veszi alapul. A harmadik szabály sormagasságra vonatkozó százalékos értéke magát az aktuális elemet veszi viszonyítási alapul. Az első három szabályban a 'font-style', 'font-variant', 'font-weight' tulajdonságok nincsenek explicit módon említve, ezért mindhárom tulajdonság alapértelmezett értékét ('normal') veszi fel. A negyedik szabály a 'font-weight' tulajdonságot 'bold'-ra, a 'font-style' tulajdonságot 'italic'-ra állítja, a 'font-variant' pedig implicit módon kapja meg a 'normal' értéket. Az ötödik szabály állítja be a 'font-variant' (small-caps), a font-size (a szülő elem betűméretének 120%-a), a 'line-height' (a fontméret 120%-a) és a 'font-family' (fantasy) tulajdonságokat.

Szín- és háttértulajdonságok

Ezek a tulajdonságok írják le egy elem (gyakran előtérzínként [foreground color] említett) szín és háttértulajdonságait. A háttér az a felület, amelyre a tartalom kirajzolásra kerül. Ez lehet háttér szín, vagy háttér kép. Szintén ezek a tulajdonságok használhatóak néhány más fontos megjelenítési lehetőség beállításához: háttér-kép helyzete, legyen-e és hányszor ismételve, fix legyen-e, vagy gördíthető a vászonhoz képest, stb...

A color tulajdonság normál módon öröklődik. A háttér tulajdonságai nem öröklődnek, de az egyes szülő elemek hátterei áttűnnek az alapértelmezett háttérértéken, mivel a background-color alapértelmezett értéke a transparent.

'color'

Értéke lehet: szín

Alapértelmezés: Böngészőfüggő.

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság írja le egy elemhez tartozó szöveg színét; gyakran előtérzínként (foreground color) történik róla említés. Egy szín értékének meghatározása többféle módon történhet:

EM { color: red }

EM { color: rgb(255,0,0) }

EM { color: #FF0000 }

Fenti három deklaráció mindegyike az `EM` elem szövegéhez a vörös színt rendeli hozzá. Az első esetben a szín *nevét* írtuk le, a második és harmadik esetben a szín *RGB kódját*; először *decimálisan*, az utolsó esetben *hexadecimálisan*.

'background-color'

Értéke lehet: <szín> | transparent

Alapértelmezés: transparent

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be egy elem háttérszínét.

```
H1 { background-color: red }
```

```
H1 { background-color: rgb(255,0,0) }
```

```
H1 { background-color: #FF0000 }
```

'background-image'

Értéke lehet: <url> | none

Alapértelmezés: none

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be egy elem hátterként látható képet. Egy háttér-kép beállításakor beállítható háttér-szín is, amely akkor látható, ha a kép nem elérhető. Ha a kép megjeleníthető, a háttérszínt el fogja takarni.

```
BODY { background-image: url(hatter.gif) }
```

```
P { background-image: none }
```

'background-repeat'

Értéke lehet: repeat |

repeat-x | repeat-y |

no-repeat

Alapértelmezés: repeat

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ha van háttér-kép beállítva, a 'background-repeat' tulajdonsággal állítható be, hogy legyen-e és hányszor legyen ismételve a kiválasztott kép.

A repeat érték azt jelenti, hogy a kép vízszintesen és függőlegesen egyaránt ismétlődni fog. A repeat-x (repeat-y) érték felelős a kép vízszintes (függőleges) ismétlődéséért; így egy kisebb méretű háttérképből a képernyő egyik oldalától a másikig érő sáv állítható elő.

```
BODY
```

```
{  
  background: red url(hatter.gif);  
  background-repeat: repeat-y;  
}
```

Fenti példában a háttérkép függőlegesen ismétlődik.

'background-attachment'

Értéke lehet: scroll | fixed

Alapértelmezés: scroll

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ha van beállított háttérkép, a 'background-attachment' értéke határozza meg, hogy fixen marad-e, vagy a tartalommal együtt gördíthető.

```
BODY
```

```
{
background: red url(hatter.gif);
background-repeat: repeat-y;
background-attachment: fixed;
}
```

A böngészők néha a `fixed` értéket is `scroll`-nak értelmezik. A W3C ajánlása azt tartalmazza, hogy a böngészők a `fixed` értéket értelmezzék helyesen, legalább a HTML és a BODY elemek vonatkozásában, mivel a weblapok szerzőinek nincs arra lehetőségük, hogy külön háttérrel válasszanak csak a gyengébb képességű böngészők számára.

'background-position'

Értéke lehet: [<százalékos> | <hossz>]{1,2}[<top | center | bottom>][<left | center | right>]

Alapértelmezés: 0% 0%

Alkalmazható: Blokkszintű- és helyettesített elemekhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a formázott elem

Ha van beállított háttér-kép, a `'background-position'` értéke határozza meg annak betöltődéskor elfoglalt helyzetét. A 0% 0% értékpár használatával a kép az elemet körbefogó doboz bal felső sarkába kerül. (Nem a margót, szegélyt, kitöltést tartalmazó dobozról van szó.) A 100% 100% értékpár a kép jobb alsó sarkát azelem jobb alsó sarkába helyezi. A 14% 84% értékpár a képet az elemen vízszintesen a 14%-ot jelentő ponttól függőlegesen a 84%-ot jelentő pontig helyezi el.

A 2cm 2cm értékpár a képet az elem bal felső sarkától 2 cm-re balra és 2 cm-re lefelé helyezi el.

Ha csak egy százalékos, vagy hosszúságérték van megadva, az csak a vízszintes pozíciót állítja be, a függőleges pozíció értéke 50% lesz. Két érték megadása esetén a vízszintes pozícióra vonatkozó értéket kell először megadni. A hosszúság- és százalékos értékek kombinációja megengedett (50% 2cm). Megengedett a negatív értékek használata is.

A háttérkép helyzetének beállításához használhatók kulcsszó-értékek is. A kulcsszavak **nem kombinálhatók** a hosszúság- és százalékos értékekkel. A használható kulcsszó-kombinációk és értelmezésük:

Kulcsszavak	Értelmezésük
'top left' és 'left top'	0% 0%
'top', 'top center', 'center top'	50% 50%
'right top', 'top right'	100% 0%
'left', 'left center', 'center left'	0% 50%
'center', 'center center'	50% 50%
'right', 'right center', 'center right'	100% 50%
'bottom left', 'left bottom'	0% 100%
'bottom', 'bottom center', 'center bottom'	50% 100%
'bottom right', 'right bottom'	100% 100%

Példák:

```
BODY {background: url(hatter.gif) right top}
BODY {background: url(hatter.gif) top center}
BODY {background: url(hatter.gif) center}
BODY {background: url(hatter.gif) bottom}
```

Ha a háttér-kép rögzítve van a vászonhoz, az elem elhelyezéséhez a méretek viszonyítása nem az elemhez, hanem a vászonhoz történik.

'background'

Értéke lehet: <background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>

Alapértelmezés: Nincs definiálva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Használható a <background-position> tulajdonsághoz.

A *background* egy ún. gyorstulajdonság, amellyel egy helyről állítható be az összes háttérre vonatkozó tulajdonság. Használható értékei megegyeznek az egyedi háttér-tulajdonságok **összes** értékeivel.

```
BODY {background: red}
```

```
P {background: url(hatter.jpg) gray 50% repeat fixed}
```

A *background* tulajdonság mindig beállítja az összes egyedi tulajdonságot. Az első példában csak a háttér-szín lett beállítva, a többi tulajdonság alapértelmezett értékeit kapja. A második példában az összes egyedi tulajdonság kapott kezdőértéket.

Szöveg tulajdonságok

'word-spacing'

Értéke lehet: normal | hossz

Alapértelmezés: normal

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A megadott hosszúságérték jelzi, hogy mennyivel növelendő meg a szavak közötti alapértelmezett szóköz értéke. Értéke lehet negatív, de lehetnek megvalósítás-függő korlátozások. A böngésző szabadon választhatja meg a megfelelő szóközérték-kiszámítási algoritmust. A szóköz mértéke függhet a szöveg igazításától is (ez a text-align tulajdonság értékeivel állítható be).

```
H1 { word-spacing: 1em }
```

Fenti példában a H1 elemen belül a szóköz értéke 1em értékkel megnövelt. A böngészők a word-spacing bármely értékét értelmezhetik normal-ként.

'letter-spacing'

Értéke lehet: normal | hossz

Alapértelmezés: normal

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A megadott hosszúságérték jelzi, hogy mennyivel növelendő meg a betűk közötti alapértelmezett köz értéke. Értéke lehet negatív, de lehetnek megvalósítás-függő korlátozások. A böngésző szabadon választhatja meg a megfelelő betűközérték-kiszámítási algoritmust. A betűköz mértéke függhet a szöveg igazításától is (ez a text-align tulajdonság értékeivel állítható be).

```
BLOCKQUOTE { letter-spacing: 0,1em }
```

Fenti példában a BLOCKQUOTE elemen belül a betűköz értéke 0,1em értékkel megnövelt. Ha a beállított érték a normal, a böngésző szabadon állíthatja be a betűközt, a szöveg igazításának megfelelően. Ez nem történik meg, ha a letter-spacing tulajdonság értékét explicit módon adjuk meg:

```
BLOCKQUOTE { letter-spacing: 0 }
```

```
BLOCKQUOTE { letter-spacing: 0cm }
```

A böngészők a letter-spacing bármely értékét értelmezhetik normal-ként.

'text-decoration'

Értéke lehet: none | [underline | overline | line-through | blink]

Alapértelmezés: none

Alkalmazható: Minden elemhez

Öröklődik: Nem, de lásd a részletezést alább

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság írja le egy elem szövegének megjelenítésekor alkalmazható dekorációkat (nincs, aláhúzott, fölhúzott, áthúzott, villogó). Ha az elem üres (), vagy nem tartalmaz szöveget (IMG), a tulajdonság nem eredményez látható hatást. A blink érték hatására a szöveg villog.

A szövegdekoráció színeit a color tulajdonság értékeivel kell beállítani.

Ez a tulajdonság nem öröklődik, de az elemek összeillenek szülő elemeikkel. Tehát; ha egy elem aláhúzott, az aláhúzás meg fog jelenni a gyermek elemnél is. Az aláhúzás színe ugyanaz marad akkor is, ha a leszármazott elem color tulajdonságának értéke más.

A:link, A:active, A:visited { text-decoration: underline }

A fenti példa az összes link szövegét (link minden A elem, amelynek HREF attribútuma van). A böngészőknek fel **kell** ismerniük a blink kulcsszót, de nem kötelesek támogatni a villogtatást.

'vertical-align'

Értéke lehet: baseline|sub|super|top|text-top|middle|bottom|text-bottom|<százalékos>

Alapértelmezés: baseline

Alkalmazható: Soron belüli elemekhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap az elem line-height tulajdonsága

Ez a tulajdonság az elem függőleges pozicionálását befolyásolja. Értékkészletének egy csoportja a szülő elemhez viszonylik:

- ? baseline - az elem alapvonalára igazít (vagy az aljára, ha a szülő elemnek nincs alapvonala);
- ? middle - az elem függőleges középpontjára igazít (általában kép esetén);
- ? sub - az elem alsó indexe;
- ? super - az elem felső indexe;
- ? text-top - az elem tetejéhez igazít, a szülő elem betűinek tetejéhez;
- ? text-bottom - az elem aljához igazít, a szülő elem betűinek aljához.

Az értékkészlet másik csoportja a formázott sorhoz viszonylik:

- ? top - az elem tetejéhez igazít, a sor legmagasabb elemével egy vonalba;
- ? bottom - az elem aljához igazít, a sor legmélyebben levő elemével egy vonalba;

A százalékos értékek viszonyítási alapja az elem line-height tulajdonsága. Ez a tulajdonság felel az elem alapvonalának (baseline) hollétéért (vagy az elem aljának elhelyezkedéséért, ha az elemnek nincs alapvonala). Negatív értékek megengedettek (a -100% lesüllyeszti az elemet annyira, hogy az elem alapvonala a következő sor alapvonalához ér. Ez az olyan elemek függőleges igazításának is különösen pontos beállítását teszi lehetővé, amelyeknek nincs alapvonaluk [képek]).

'text-transform'

Értéke lehet: capitalize|uppercase|lowercase|none

Alapértelmezés: none

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

- ? capitalize - Minden szó első karakterét nagybetűssé alakítja;
- ? uppercase - Az elem összes betűjét nagybetűssé alakítja;
- ? lowercase - Az elem összes betűjét kisbetűssé alakítja;
- ? none - Hatástalanítja az örökölt értéket.

Az aktuális transzfomáció minden esetben nyelvfüggő.

```
H1 { text-transform: uppercase }
```

A H1 elemben levő szöveget nagybetűssé alakítja.

A böngészők figyelmen kívül hagyhatják a text-transform tulajdonságot, ha azok a karakterek, amelyikre alkalmazni kellene, nem a *Latin-1* karakterkészletből valók.

'text-align'

Értéke lehet: left | right | center | justify

Alapértelmezés: Böngészőfüggő

Alkalmazható: Blokkszintű elemekhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság írja le a szöveg igazítását az elemen belül. Az aktuális igazítási algoritmust a böngésző nyelvfüggően alkalmazza.

```
DIV.center { text-align: center }
```

Mivel a text-align öröklődik, a DIV elemen belüli összes CLASS="center" attribútummal ellátott blokkszintű elem középre lesz igazítva. Megjegyzendő, hogy az igazítás viszonyítása nem a vászonhoz, hanem az elemhez történik. Ha a justify értéket a böngésző nem támogatja, általában a left értékkel helyettesíti.

'text-indent'

Értéke lehet: hossz | százalékos

Alapértelmezés: 0 (nulla)

Alkalmazható: Blokkszintű elemekhez

Öröklődik: Igen

Százalékos értékek: Viszonyítási alap a szülő elem szélessége

Ez a tulajdonság határozza meg a formázott sor behúzását. Értéke negatív is lehet, de lehetnek megvalósításbeli korlátozások. Behúzás nem helyezhető el olyan elem belsejébe, ahol valamilyen törés (pl.: BR) már van.

```
P {text-indent: 3em }
```

'line-height'

Értéke lehet: normal | szám | hossz | százalékos

Alapértelmezés: normal

Alkalmazható: Minden elemhez

Öröklődik: Igen

Százalékos értékek: Viszonyítási alap az elem fontmérete

Ez a tulajdonság állítja be a két szomszédos sor alapvonalának távolságát. Ha numerikus érték van meghatározva, a sormagasságot a fontméret és a szám szorzata adja meg. Ez a megoldás a százalékos értékmegadáستól az öröklődésben különbözik: számérték megadásakor a leszármazott elemek magát a **számot** öröklöik, nem az eredményt. Negatív értékek nem alkalmazhatóak. A következő példában szereplő három deklaráció eredménye ugyanaz a sormagasság:

```
DIV { line-height: 1,2; font-size: 10pt }          /* számérték */
DIV { line-height: 1,2em; font-size: 10pt }        /* hossz      */
DIV { line-height: 120%; font-size: 10pt }         /* százalékos */
```

A normal érték a line-height tulajdonság értékét az alkalmazott fonthoz képest ésszerű méretre állítja be.

Doboz-tulajdonságok

A doboz-tulajdonságok az elemeket reprezentáló dobozok méretét, területét és helyzetét állítják be. A margó-tulajdonságok állítják be az elemhez tartozó margókat. A margin tulajdonság minden oldalra vonatkozik, míg a többi margó-tulajdonság csak a saját margóikra van hatással. A kitöltés-tulajdonságok írják le, mekkora hely legyen a szegély és az

elem tartalma között. A `padding` tulajdonság minden oldalra vonatkozik, míg a többi kitöltés-tulajdonság csak csak a saját oldalon lévő kitöltésre van hatással. A szegély-tulajdonságok állítják be az elemhez tartozó szegélyeket. Minden elemnek négy szegélye van, minden oldalon egy-egy, amelyek szélességükkel, színükkel, stílusukkal vannak leírva. A `width` és `height` tulajdonságok állítják be a doboz méretét; a `float` és `clear` tulajdonságok az elem pozícióját változtathatják.

'margin-top'

Értéke lehet: hossz | százalékos | auto

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be az elem felső margóját:

```
H1 { margin-top: 2em }
```

Negatív érték használata engedélyezett, de lehetnek megvalósításbeli korlátozások.

'margin-right'

Értéke lehet: hossz | százalékos | auto

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be az elem jobb oldali margóját:

```
H1 { margin-right: 2em }
```

Negatív érték használata engedélyezett, de lehetnek megvalósításbeli korlátozások.

'margin-bottom'

Értéke lehet: hossz | százalékos | auto

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be az elem alsó margóját:

```
H1 { margin-bottom: 2em }
```

Negatív érték használata engedélyezett, de lehetnek megvalósításbeli korlátozások.

'margin-left'

Értéke lehet: hossz | százalékos | auto

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be az bal oldali margóját:

```
H1 { margin-left: 2em }
```

Negatív érték használata engedélyezett, de lehetnek megvalósításbeli korlátozások.

'margin'

Értéke lehet: [hossz | százalékos | auto]{1,4}

Alapértelmezés: Nincs definiálva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

A `margin` tulajdonság egy 'gyorstulajdonság', amelynek segítségével egy deklaráción belül állítható be a `margin-top`, `margin-right`, `margin-bottom` és a `margin-left` tulajdonság értéke. Ha értékként négy számértéket adunk meg, annak értelemszerű sorrendje mindig a

felső => jobb alsó => bal. Ha csak egy érték van megadva, az mind a négy oldalra vonatkozik, ha kettő, vagy három: a szemben lévő értékek meg fognak egyezni.

```
BODY { margin: 2em }
```

```
BODY { margin: 1 em 2em }
```

```
BODY { margin: 1em 2em 3em }
```

Fenti példák közül a legalsó -hatását tekintve- a következővel megegyezik:

```
BODY {  
    margin-top: 1em;  
    margin-right: 2em;  
    margin-bottom: 3em;  
    margin-left: 2 em;    /* megegyezik a szembeni oldal értékével */  
}
```

Negatív margó-értékek engedélyezettek, de lehetnek megvalósításbeli korlátozások.

'padding-top'

Értéke lehet: hossz | százalékos

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be egy elem felső kitöltését.

```
BLOCKQUOTE { padding-top: 0,3em }
```

Negatív kitöltés-érték nem megengedett.

'padding-right'

Értéke lehet: hossz | százalékos

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem.

Ez a tulajdonság állítja be egy elem jobb oldali kitöltését

```
BLOCKQUOTE { padding-right: 0,3em }
```

Negatív kitöltés-érték nem megengedett.

'padding-bottom'

Értéke lehet: hossz | százalékos

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be egy elem alsó kitöltését

```
BLOCKQUOTE { padding-bottom: 0,3em }
```

Negatív kitöltés-érték nem megengedett.

'padding-left'

Értéke lehet: hossz | százalékos

Alapértelmezés: 0

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokk szintű szülőelem

Ez a tulajdonság állítja be egy elem bal oldali kitöltését

```
BLOCKQUOTE { padding-left: 0,3em }
```

Negatív kitöltés-érték nem megengedett.

'padding'

Értéke lehet: [hossz | százalékos]{1,4}

Alapértelmezés: Nincs definiálva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a legközelebbi blokkszintû szülőelem

A padding tulajdonság egy 'gyorstulajdonság', amelynek segítségével egy deklaráción belül állítható be a padding-top, padding-right, padding-bottom és a padding-left tulajdonság értéke. Ha értéként négy számértéket adunk meg, annak érteélemzési sorrendje mindig a felső => jobb alsó => bal. Ha csak egy érték van megadva, az mind a négy oldalra vonatkozik, ha kettő, vagy három: a szemben lévő értékek meg fognak egyezni.

A kitöltőfelület megadása a background tulajdonsággal:

```
H1 {  
    background: white;  
    padding: 1em 2em;  
}
```

Fenti példa 1em értékû kitöltést állít be függőlegesen (fent és lent), és 2em értékû kitöltést állít be vízszintesen (jobb- és baloldalt). Az em érték magyarul a -szorosnak felel meg. Viszonyítási alapja az elem fontmérete. Az 1em (egyszeres) megegyezik a használt font magasságával.

A kitöltési értékek nem lehetnek negatívak.

'border-top-width'

Értéke lehet: thin | medium | thick | hossz

Alapértelmezés: medium

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be egy elem felső szegélyét. A kulcsszavakhoz tartozó tényleges értékek böngészőfüggők, de a következő sorrend mindenképpen érvényben marad: thin (vékony) « medium (közepes) « thick (vastag)

```
H1 { border: solid thick red }  
P { border: solid thick blue }
```

A fenti példa szerint a H1 és a P elemek ugyanolyan szegélyt kapnak, függetlenül a fontmérettől. Relatív szélesség eléréséhez az em használható:

```
H1 { border: solid 0,5em }
```

A szegélyszélességek nem kaphatnak negatív értéket.

'border-right-width'

Értéke lehet: thin | medium | thick | hossz

Alapértelmezés: medium

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be egy elem jobb oldali szegélyét. Használata megegyezik a border-top-width tulajdonság használatával.

'border-bottom-width'

Értéke lehet: thin | medium | thick | hossz

Alapértelmezés: medium

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be egy elem alsó szegélyét. Használata megegyezik a border-top-width tulajdonság használatával.

'border-left-width'

Értéke lehet: thin | medium | thick | hossz

Alapértelmezés: medium

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be egy elem bal oldali szegélyét. Használata megegyezik a border-top-width tulajdonság használatával.

'border-width'

Értéke lehet: [thin | medium | thick | hossz]{1,4}

Alapértelmezés: Nincs definiálva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

A border-width tulajdonság egy 'gyorstulajdonság', amelynek segítségével egy deklaráción belül állítható be a border-width-top, border-width-right, border-width-bottom és a border-width-left tulajdonság értéke.

Négy értéke lehet, a következő értelmezésben:

- ? *Egy megadott érték:* - Mind a négy szegély a megadott értéket kapja;
- ? *Két megadott érték:* - Az első érték a felső és alsó szegélyekre, a második érték a bal és jobb szegélyre vonatkozik;
- ? *Három megadott érték:* - Az első érték a felső, a második a jobb és bal, a harmadik az alsó szegélyre vonatkozik;
- ? *Négy megadott érték:* - Felső, jobb, alsó és bal oldali szegélyekre vonatkozik, a felsorolás sorrendjében.

A következő példában a megjegyzésben írt értékek jelzik a szegélyek vastagságát:

```
H1 {border-width: thin } /* thin thin thin thin */
H1 {border-width: thin thick } /* thin thick thin thick */
H1 {border-width: thin thick medium } /* thin thick medium thin */
H1 {border-width: thin thick medium thin } /* thin thick medium thin */
```

A szegélyek szélessége nem vehet fel negatív értéket.

'border-color'

Értéke lehet: szín{1,4}

Alapértelmezés: A color tulajdonság értéke

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

A border-color tulajdonság a négy szegély színét állítja be. Négy értéke lehet, az értékek a különböző oldalak színeit állítják be, ahogyan azt a border-width tulajdonságnál leírtuk.

Ha nincs színérték meghatározva, helyét az elem color tulajdonsága veszi át.

```
P {
  color: black;
  background: white;
  border: solid
}
```

Fenti példában az elem szegélye vékony, fekete vonal lesz.

'border-style'

Értéke lehet: none | dotted | dashed | solid | double | groove | ridge | inset | outset

Alapértelmezés: none

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

A border-style tulajdonság állítja be a négy szegély stílusát. Négy értéke lehet, az értékek a különböző oldalak színeit állítják be, ahogyan azt a border-width tulajdonságnál leírtuk.

```
#b14 { border-style: solid dotted }
```

Fenti példa szerint a vízszintes szegélyek stílusa `solid`, a függőleges szegélyeké pedig `dotted` (pontvonal) lesz. Mivel a szegélystílusok alapértelmezés szerinti beállítása `none`, külön beállított értékek nélkül az elemeknek nem lesz látható szegélye.

A szegélystílusok a következőket jelentik:

- ? `none` - Nincs kirajzolt szegély (tekintet nélkül a `border-width` tulajdosság értékére);
- ? `dotted` - A szegély pontozott vonallal lesz kirajzolva az elem hátterére;
- ? `dashed` - A szegély szaggatott vonallal lesz kirajzolva az elem hátterére;
- ? `solid` - A szegély folytonos vonallal lesz kirajzolva az elem hátterére;
- ? `double` - A szegély kettős vonallal lesz kirajzolva az elem hátterére. A két vonal összes szélessége és a köztük levő köz megegyezik a `border-width` értékével;
- ? `groove` - A szegély 3D süllyesztett hatást keltve lesz kirajzolva;
- ? `ridge` - A szegély 3D domború hatást keltve lesz kirajzolva;
- ? `inset` - A szegély 3D beékel stílust keltve lesz kirajzolva;
- ? `outset` - A szegély 3D kiemelt hatást keltve lesz kirajzolva;

'border-top'

Értéke lehet: `border-top-width||border-style||szín`

Alapértelmezés: Nincs meghatározva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a 'gyorstulajdonság' egy elem felső szegélye szélességének, stílusának és színének beállítására szolgál.

```
H1 { border-top: thick solid red }
```

Fenti példa a `H1` elem felső szegélyének vastagságát, vonaltípusát és színét állítja be. A felsorolásból kihagyott értékek alapértelmezésükkel lesznek helyettesítve.

```
H1 { border-top: thick solid }
```

Mivel ebben a példában a szín nem szerepel, a szegély színe megegyezik az elem szegélyének színével.

'border-right'

Értéke lehet: `border-top-width||border-style||szín`

Alapértelmezés: Nincs meghatározva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a 'gyorstulajdonság' egy elem jobb oldali szegélye szélességének, stílusának és színének beállítására szolgál. Használata megegyezik a `border-top` tulajdonságéval.

'border-bottom'

Értéke lehet: `border-top-width||border-style||szín`

Alapértelmezés: Nincs meghatározva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a 'gyorstulajdonság' egy elem alsó szegélye szélességének, stílusának és színének beállítására szolgál. Használata megegyezik a `border-top` tulajdonságéval.

'border-left'

Értéke lehet: `border-top-width||border-style||szín`

Alapértelmezés: Nincs meghatározva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a 'gyorstulajdonság' egy elem bal oldali szegélye szélességének, stílusának és színének beállítására szolgál. Használata megegyezik a border-top tulajdonságával.

'border'

Értéke lehet: border-width||border-style||szín

Alapértelmezés: Nincs meghatározva

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

A border tulajdonság egy 'gyorstulajdonság', amelynek segítségével egy deklaráción belül állítható be a border-top, border-right, border-bottom és a border-left tulajdonság értéke.

A következő példában mutatott első deklaráció eredménye megegyezik a második deklaráció eredményével:

```
P { border: solid red }
```

```
P {  
    border-top: solid red  
    border-right: solid red  
    border-bottom: solid red  
    border-left: solid red  
}
```

Eltérően a margin és padding tulajdonságoktól, a border tulajdonság nem tud különböző értékeket beállítani a különböző oldalakhoz. Ehhez a többi szegélytulajdonságot kell használni.

'width'

Értéke lehet: hossz | százalékos | auto

Alapértelmezés: auto

Alkalmazható: Blokkszintű elemekhez és helyettesített elemekhez

Öröklődik: Nem

Százalékos értékek: Viszonyítási alap a szülő elem szélessége

Ez a tulajdonság általában szöveg-elemekhez használatos, de igen hasznos a helyettesített elemeknél (pl.: képek) is. Ezen a módon megadható egy elem szélessége. Az elemek méretezésénél az oldalak aránya változatlan marad, ha a height tulajdonság értéke auto.

```
IMG.icon { width: 100px }
```

Ha a width és a height tulajdonság értéke is auto; az elem saját eredeti méretével jelenik meg. Negatív értékek nem alkalmazhatók. A width tulajdonság és a margó, valamint a kitöltés közötti kapcsolat leírásáért lásd a formázásmodell fejezetet.

'height'

Értéke lehet: hossz | auto

Alapértelmezés: auto

Alkalmazható: Blokkszintű elemekhez és helyettesített elemekhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság általában szöveg-elemekhez használatos, de igen hasznos a helyettesített elemeknél (pl.: képek) is. Ezen a módon megadható egy elem szélessége. Az elemek méretezésénél az oldalak aránya változatlan marad, ha a width tulajdonság értéke auto.

```
IMG.icon { height: 100px }
```

Ha a width és a height tulajdonság értéke is auto; az elem saját eredeti méretével jelenik meg. Negatív értékek nem alkalmazhatók. Ha az elem, amelyre alkalmazzuk nem helyettesített elem, a böngészők figyelmen kívül hagyhatják a height tulajdonságot (auto értékkel helyettesítik).

'float'

Értéke lehet: left | right | none

Alapértelmezés: none

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ha none értékét alkalmazzuk, az elem ott jelenik meg, ahol a szövegben az őt beillesztő kódsor található. Ha értéke left (right); az elem a bal- (jobb-) oldalon fog megjelenni, és a szöveg a jobb (bal) oldalán fog körbefutni. Ha értéke left, vagy (right); az elem blokkszintűként viselkedik.

```
IMG.icon {  
    float: left;  
    margin-left: 0;  
}
```

Fenti példa az összes 'icon' osztályba tartozó IMG elemet a szülő elem bal oldalára helyezi el. A float tulajdonságot leggyakrabban soron belüli képekhez használjuk, de jól alkalmazható szöveges elemekhez is.

'clear'

Értéke lehet: none | left | right | both

Alapértelmezés: none

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság határozza meg, hogy elem melyik oldalán engedélyezi lebegő elemek megjelenését. Pontosabban: E tulajdonság értékei sorolják fel azokat az oldalakat, ahol lebegő elemek nem jelenhetnek meg. Ha a clear értéke leftre van beállítva, az elem, amelyre alkalmazzuk, a bal oldalán levő bármely lebegő elem alá kerül.

```
H1 { clear: left }
```

Osztályozó tulajdonságok

Ezek a tulajdonságok sorolják az elemeket kategóriákba, ezenkívül beállítják megjelenítési tulajdonságaikat is. A felsorolás-tulajdonságok írják le, a listaelemek formázási előírásait. A felsorolás-tulajdonságok minden elemhez alkalmazhatóak és normál módon öröklődnek a leszármazási fastruktúrán. Mindamellet megjelénítési hatást csak a listaelemekre gyakorolnak. A HTML-ben tipikusan ide tartoznak a LI elemek.

'display'

Értéke lehet: block | inline | list-item | none

Alapértelmezés: block

Alkalmazható: Minden elemhez

Öröklődik: Nem

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság határozza meg, hogy egy elem legyen-e, és hogyan legyen ábrázolva a megjelenítőn (a megjelenítő lehet képernyő, nyomtatási termék, stb...).

Egy block értékkel rendelkező elem új dobozt kap. A doboz elhelyezését és a szomszédos dobozokhoz viszonyított helyzetét a formázásmodell határozza meg. Általában, a H1, vagy P jellegű elemek block-típusúak. A list-item érték hasonló a blockhoz, kivéve, ha listajelölő is van hozzáadva. A HTML-ben tipikusan ilyen (listajelölővel ellátott) elem a LI.

Egy inline értékkel rendelkező elem soron belüli dobozt kap, ugyanabban a sorban, mint az őt megelőző tartalom. A doboz méretezése formázott tartalmának mérete szerint történik. Ha tartalma szöveg, sorokra osztható, és minden sornak külön doboza lesz. A margó-, szegély-,

és kitöltés-tulajdonságok használatosak az inline elemekhez, a sortörésnél nincs látható hatásuk.

A none érték használata kikapcsolja az elem megjelenítését, beleértve leszármazott elemeit és az öt körülvevő dobozt is.

```
P { display: block }
EM { display: inline }
LI { display: list-item }
IMG { display: none }
```

A legutolsó definíció kikapcsolja a képek megjelenítését.

A display tulajdonság alapértelmezett értéke a block; de minden böngésző rendelkezik alapértelmezett értékekkel minden HTML elemre vonatkozóan, amely a HTML specifikációban írt javasolt megjelenítésen alapul.

A böngészők figyelmen kívül hagyhatják a display tulajdonságot, a stíluslap szerzője által definiált helyett használhatják saját alapértelmezett értékeiket is.

'white-space'

Értéke lehet: normal | pre | nowrap

Alapértelmezés: normal

Alkalmazható: Blokkszintű elemekhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

Az angol 'whitespace' kifejezésre nem találtam egy magyar szót; körülírva: a szavak, mondatok közötti *üres közt* jelenti, ami nem azonos a szóközökkel. A továbbiakban -egyéb megegyezés híján- *közként* fogom említeni.

Ez a tulajdonság írja le, hogyan legyenek értelmezve a közök egy elemen belül: normal módon (amikor a közök egy szóközzé vannak tömörítve), pre-ként (úgy viselkedjen, mint a PRE a HTML-ben), vagy nowrap módon (a sortörés BR eleméhez hasonló módon):

```
PRE { white-space: pre }
P { white-space: normal }
```

A white-space tulajdonság alapértelmezése a normal, de a böngészőknek általában minden HTML elemhez van alapértelmezett értékük, a HTML specifikációban leírt elemkirajzolási előírások szerint.

A böngészők figyelmen kívül hagyhatják a display tulajdonságot, a stíluslap szerzője által definiált helyett használhatják saját alapértelmezett értékeiket is.

'list-style-type'

Értéke lehet: disc|circle|square|decimal|lower-roman|upper-roman|lower-alpha|upper-alpha|none

Alapértelmezés: disc

Alkalmazható: display: list-item tulajdonság-érték párral rendelkező elemekhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

E tulajdonság használatával határozható meg a listajelölő megjelenése, ha a list-style-image tulajdonság értéke none, vagy a kép, amelyre a benne szereplő hivatkozás mutat, nem elérhető.

```
OL { list-style-type: decimal } /* 1 2 3 4 stb... */
OL { list-style-type: lower-alpha } /* a b c d stb... */
OL { list-style-type: lower-roman } /* i ii iii stb... */
```

'list-style-image'

Értéke lehet: <url> | none

Alapértelmezés: none

Alkalmazható: display: list-item tulajdonság-érték párral rendelkező elemekhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

Ez a tulajdonság állítja be azt a képet, ami listaelem-jelölőként alkalmazható. Ha a kép elérhető, helyettesíteni fogja a `list-style-type` tulajdonságban beállított jelölőt.

```
UL { list-style-image: url(images/styleimage.gif) }
```

'list-style-position'

Értéke lehet: `inside` | `outside`

Alapértelmezés: `outside`

Alkalmazható: `display: list-item` tulajdonság-érték párral rendelkező elemekhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A `list-style-position` értéke határozza meg a lista-jelölő pozícióját a tartalomhoz képest.

'list-style'

Értéke lehet: `[disc|circle|square|decimal|lower-roman|upper-roman|lower-alpha|upper-alpha|none]` || `[inside|outside]` || `[url|none]`

Alapértelmezés: Nincs definiálva

Alkalmazható: `display: list-item` tulajdonság-érték párral rendelkező elemekhez

Öröklődik: Igen

Százalékos értékek: Nincs értelmezve

A `list-style` tulajdonság egy 'gyorstulajdonság', amelynek segítségével a `list-style-type`, a `list-style-image` és a `list-style-position` tulajdonságok értékei állíthatók be.

```
UL { list-style: upper-roman inside }
```

```
UL UL { list-style: circle outside }
```

```
LI.square { list-style: square }
```

A `LI` elemre közvetlenül beállított `list-style` tulajdonság nem várt eredményt hozhat.

Tekintsünk egy példát:

```
<STYLE TYPE="text/css">
  OL.alpha LI {list-style: lower-alpha }
  UL LI      {list-style: disc }
</STYLE>
<BODY>
  <OL CLASS="alpha">
    <LI>1. szint
    <UL>
      <LI>2. szint
    </UL>
  </OL>
</BODY>
```

Mivel a fenti példában az első definíciónak nagyobb az egyedisége, felül fogja bírálni a második definíció előírásait és csak a `lower-alpha` felsorolás-stílus fog érvényesülni. Ezért csak a felsorolás jellegű elemeknél ajánlott beállítani a `list-style` tulajdonságot.

```
OL.alpha { list-style: lower-alpha }
```

```
UL      { list-style: disc }
```

Fenti példában az `OL` és `UL` elemekre beállított értékek -hála az öröklődési szabályoknak- vonatkozni fognak a `LI` elemekre is.

Egységek

Hosszúság egységek

A hosszúságértékek meghatározott formátuma egy opcionális `+`, vagy `-` jelből (alapértelmezett a `+`), az azt közvetlenül követő számból és a számot közvetlenül követő egység-azonosítóból (annak kétbetűs rövidítéséből) áll. A 0 (nulla) szám után az egység-azonosító opcionális. Néhány tulajdonság használatánál engedélyezett a negatív érték használata, de ez bonyolíthatja a formázásmodellt és lehetnek megvalósításbeli korlátozásai is. Ha egy negatív

érték nincs támogatva, az a legközelebbi támogatott értékkel lesz helyettesítve. A hosszúság-egységeknek két típusa használható: a *relatív* és az *abszolút*. A relatív egységek e méretet egy másik hosszúság-tulajdonsághoz viszonyítva adják meg. Azok a stíluslapok, amelyek a méreteket relatívan adják meg, könnyebben skálázhatók egyik médiatípusról a másikra (pl.: számítógép képernyőről nyomtatóra). A százalékos egységek (lásd alább) hasonló előnyöket kínálnak fel.

A következő relatív egységek használhatók:

```
H1 { margin: 0,5em } /* -szoros, az elem fontméret-magasságához képest */
H1 { margin: 1ex } /* x-magasság, az 'x' bezű magasságához képest */
P { font-size: 12px } /* pixelben, a vászonhoz képest */
```

Az *em* és *ex* értékek az elem fontméretéhez viszonyulnak. E szabály alól az egyetlen kivétel a *font-size* tulajdonság, ahol az *em* és *ex* értékek a szülő elem fontméretéhez viszonyítandók.

A *pixel* egységek (ahogy a példa mutatja) a vászon (leggyakrabban számítógép-képernyő) felbontásához viszonyulnak. Ha a kimeneti eszköz pixelsűrűsége nagyban különbözik a számítógép képernyőjétől, a megjelenítő eszköz¹ átszámolja a pixelértékeket. A javasolt *referencia pixel* mérete egy pixel látható szöge az olvasó ember karhosszúságában levő 90 dpi pixelsűrűségű eszközön. Egy átlagos karhosszúságot 71 cm-nek (28 inch) véve, egy pixel 0,0227° alatt látszik.

A gyermek elemek nem a relatív, hanem a számított értéket öröklik:

```
BODY {
    font-size: 12pt;
    text-indent: 3em; /* értsd: 36pt */
}
H1 { font-size: 15pt }
```

Fenti példában a *H1* elemek *text-indent* tulajdonságának értéke 36pt lesz, nem pedig 45pt.

Az abszolút hosszúság-egységeket csak akkor érdemes használni, ha a kiviteli eszköz fizikai tulajdonságai ismertek. A következő abszolút egységek támogatottak:

```
H1 { margin: 0,5in } /* inch; 1 inch = 2.54 cm */
H2 { line-height: 3cm } /* centiméter */
H3 { word-spacing: 4mm } /* milliméter */
H4 { font-size: 12pt } /* pont; 1 pt = 1/72 inch */
H4 { font-size: 1pc } /* pica; 1 pc = 12 pt */
```

Abban az esetben, ha a meghatározott méret nem támogatható, a böngészők megpróbálják hozzávetőlegesen megközelíteni.

Százalékos egységek

A százalékos értékek meghatározott formátuma egy opcionális +, vagy - jelből (alapértelmezett a +), az azt közvetlenül követő számból és a számot közvetlenül követő % jelből áll. A százalékos egységek mindig valamely más egységre vonatkoznak, ez leggyakrabban hosszúság-egység. Minden tulajdonságnál, amelyeknél százalékos egységek alkalmazhatók, meg van határozva, hogy a százalékos egység mire hivatkozik. A hivatkozási alap leggyakrabban az adott elem fontmérete.

```
P { line-height: 120% } /* Az elemnél alkalmazott 'font-size' 120%-a */
```

Minden örökölt CSS tulajdonságnál, ha értéke százalékosan van megadva, a leszármazott elemek a számított értéket öröklik, nem a százalékosat.

Színjelölések

A színek meghatározása történhet a szín nevével, vagy numerikusan, a szín RGB kódjával. A javasolt színmegnevezések a következők: *aqua*, *black*, *blue*, *fuchsia*, *gray*, *green*, *lime*, *maroon*, *navy*, *olive*, *purple*, *red*, *silver*, *teal*, *white* és *yellow*. Ez a 16 szín található a Windows VGA palettáján, RGB értékük nincs meghatározva ebben a specifikációban.

```
BODY { color: black; background: white }
```

```
H1 { color: maroon }
H2 { color: olive }
```

Az RGB modell számszerű színmeghatározásokat használ. A következő példák ugyanazt a színt eredményezik.

```
EM { color: #f00 } /* #rgb */
EM { color: #ff0000 } /* #rrggbb */
EM { color: rgb(255,0,0) } /* egész számok: 0 - 255 */
EM { color: rgb(100%,0%,0%) } /* százalékos : 0% - 100% */
```

Az RGB értékek hexadecimális formátuma: egy # karakter, amelyet közvetlenül követ vagy három, vagy hat hexadecimális karakter. A háromszámjegyű RGB kifejezések (#rgb) hatszámjegyű formába (#rrggbb) a számjegyek ismétlésével, nem 0-k (nullák) hozzáadásával konvertálhatók. (Tehát a #b0 kifejtve #bb00) Ez biztosítja, hogy a fehér (#ffffff) meghatározható legyen rövid formában is (#fff), függetlenül a megjelenítő eszköz színmélységétől. Az RGB értékek funkcionális formátuma: az rgb karkterlánc, amelyet közvetlenül követ a három színérték vesszővel elválasztott felsorolása (vagy három egész szám 0 - 255 értékhatáron belül, vagy három százalékos kifejezés 0% - 100% értékhatáron belül). A megadott értékhatárokon kívüli számértékek nem értelmezhetők, csonkítva lesznek. A következő három deklaráció értelmét tekintve megegyezik.

```
EM { color: rgb(255,0,0) } /* egész számok: 0 - 255 */
EM { color: rgb(300,0,0) } /* csonkítva 255 -re */
EM { color: rgb(110%,0%,0%) } /* csonkítva 100% -ra */
```

URL

Az URL rövidítés a Uniform Resource Locator kifejezést takarja, amelynek magyar megfelelője: Egységes Erőforrás Helymeghatározás.

```
BODY { background: url(images/hatter.jpg) }
```

Az URL kifejezés formája: url kulcsszó, amelyet közvetlenül követ zárójelben (()) opcionálisan egyszeres, vagy kétszeres idézőjelek (' , ") közé zárva maga az URL. Relatív URL megadásakor az elérési utat nem a dokumentumhoz kell viszonyítani, hanem a stíluslaphoz.

Összhang

Egy böngésző, amely egy dokumentum megjelenítéséhez CSS-t használ, akkor felel meg a CSS specifikáció által támasztott követelményeknek, ha:

- ? Elér el minden hivatkozott stíluslapot és specifikációjuk szerint értelmezi őket;
- ? A deklarációkat a rangsor fejezetben leírt rangsor szerint rendezi;
- ? A CSS funkcionalitását a megjelenítő eszköz korlátai közé tudja illeszteni.

Egy böngésző akkor felel meg a CSS specifikációban megfogalmazott stíluslap-követelményeknek, ha:

- ? kimenete érvényes CSS stíluslap.

Egy böngésző, amely egy dokumentum megjelenítéséhez CSS-t használ, és kimenete stíluslap, akkor felel meg a CSS specifikációnak ha a fentiekben említett *mindkét* követelménycsoportot kielégíti.

Egy böngésző sem tudja teljesíteni a CSS valamennyi lehetséges funkcióját: a böngészők akkor felelnek meg a CSS támasztotta követelményeknek, ha az alapvető funkciókat képesek teljesíteni. Az alapvető funkciók a teljes CSS specifikációból állnak, kivéve azokat a részeket, amelyek kifejezetten kivételként szerepelnek. Azokat a funkciókat, amelyek nem tartoznak a CSS alapvető funkciói közé, továbbfejlesztett (advanced) funkcióknak nevezzük.

Példák a megjelenítő eszköz korlátaira: Korlátozott erőforrások (fontok, színek), korlátozott felbontás (a margók nem jelennek meg helyesen). Ezekben az esetekben a böngésző csak

megközelíti a stíluslap előírásait. Vannak böngészők, amelyek lehetővé teszik a felhasználó számára a megjelenítés megválasztását.

Előre-kompatibilis elemzés

Ez a leírás a CSS 1-et mutatja be. Előre láthatóan a későbbi verziók több új tulajdonságot vezetnek be. Ez a fejezet azt mutatja be, hogy mit tesznek a böngészők, ha olyan deklarációval találkoznak, amelyek a CSS jelen kiadásában nem érvényesek.

- ? Az ismeretlen tulajdonságot tartalmazó deklarációkat figyelmen kívül hagyják.

Például; ha a stíluslap a

```
H1 { color: red; rotation: 70deg }
```

deklarációt tartalmazza, a böngésző úgy veszi figyelembe, mintha csak a

```
H1 { color: red }
```

deklarációt tartalmazta volna.

- ? Az érvénytelen értékeket, vagy *érvénytelen részeket tartalmazó értékeket* a böngésző szintén figyelmen kívül hagyja, a következők szerint:

```
IMG { float: left } /* CSS-nek megfelelő */
```

```
IMG { float: left top } /* a 'top' nem értéke a 'float'-nak */
```

```
IMG { background: "red" } /* a kulcsszavak nem kerülhetnek idézőjelbe */
```

```
IMG { border-width: 3 } /* hiányzik a mértékegység */
```

Fenti példában a böngésző csak az első deklarációt értelmezi, így a stíluslap *tényleges* tartalma:

```
IMG { float: left }
```

```
IMG { }
```

```
IMG { }
```

```
IMG { }
```

- ? Figyelmen kívül hagyja a böngésző az érvénytelen at (@) kulcsszavakat is, minden egyébbel, ami követi, egészen a következő pontosvesszőig (;), vagy kapcsos zárójelpárig ({ }). Példaképp tételezzük fel, hogy a stíluslap a következőket tartalmazza:

```
@three-dee
```

```
{
  @background-lightning:
  {
    azimuth: 30deg;
    elevation: 190deg
  }
}
```

```
H1 { color: red }
```

```
}
```

```
H1 { color: blue }
```

Mivel a @three-dee a CSS szerint érvénytelen, így az egész deklaráció a harmadik jobb kapcsos zárójelig (}) bezárólag érvénytelen. A böngésző egyszerűen kihagyja, a stíluslap értelmezhető része a következő lesz:

```
H1 { color: blue }
```

Részletesebben:

Egy CSS stíluslap, bármely verziójában készült is, utasítások sorozatát tartalmazza. Az utasításoknak két fajtája van: az *at előírások* és az *előíráskészletek*. Az utasítások körül lehetnek közök (whitespace) is (tabulátor-, szóköz-, ujsor-karakterek).

A CSS utasítások gyakran a HTML dokumentumba vannak beágyazva; megvan a lehetőség arra, hogy ezeket az utasításokat elrejtjük a régebbi böngészők elől. Erre a célra a HTML szabvány 'megjegyzés' (comment) jele szolgál: <-- Megjegyzés --> - a két jel közé írandó a CSS utasítássor.

Az at-előírások egy azonosítóként szereplő at-kulcsszóval kezdődnek, amely előtt közvetlenül egy **at**, vagyis @ karakter áll (pl.:@import, @page). Az azonosító tartalmazhat betűket, számjegyeket és más karaktereket (lásd alább). Egy at-előírás tartalma az első pontosvesszőig (:), vagy a következő blokkig tart. Ha egy böngésző olyan at-előírással találkozik, amely nem az @import kulcsszóval kezdődik, figyelmen kívül hagyja az egész at-előírást és az elemzést a következő utasítással folytatja. Szintén figyelmen kívül hagyja az @import kulcsszót is, ha az nem a stíluslap legelején található.

```
@import "stilus.css"
H1 { color: blue }
@import "masikstilus.css"
```

Fenti példában a második @import utasítás a CSS1 szerint érvénytelen. A CSS értelmező kihagyja az egész at-előírást, a stíluslapot a következőképpen értelmezi:

```
@import "stilus.css"
H1 { color: blue }
```

Egy blokk nyitó kapcsos zárójellel ({}) kezdődik és a neki megfelelő záró kapcsos zárójelig (}) tart. Köztük bármely egyedi karakter előfordulhat, a zárójelek (()), kapcsos zárójelek ({ }) és szögletes zárójelek ([]) kivételével, amelyek a blokkon belül csak párban fordulhatnak elő. A fent említett karakterek közé zárt utasítások egymásba ágyazhatók. Az egyszeres (') és dupla (") idézőjelek szintén párban fordulhatnak elő; a közéjük zárt karakterlánc szöveggént lesz értelmezve. A következőkben bemutatunk egy példát a blokk értelmezésére. Figyeljük meg, hogy az idézőjelek között szereplő záró kapcsos zárójel *nem párja* a nyitó kapcsos zárójelnek, valamint a második egyszeres idézőjel (') egy 'függő' karakter, nem párja a nyitó idézőjelnek:

```
{ causta: "}" + ({7} * '\'' ) }
```

Egy előíráskészlet egy szelektorból és a hozzá tartozó deklarációkból áll. A szelektorként értelmezendő karakterlánc az első nyitó kapcsos zárójelig ({}) tart (de azt nem foglalja magába). Azt az előíráskészletet, amely nem érvényes CSS szelektorról kezdődik, a böngészők figyelmen kívül hagyják. Tételezzük fel, hogy egy stíluslap a következőképp néz ki:

```
H1 { color: blue }
P[align], UL { color: red; font-size: large }
P EM { font-weight: bold }
```

Fenti példa második sora érvénytelen CSS szelektort tartalmaz, a böngésző a következőképp fogja értelmezni:

```
H1 { color: blue }
P EM { font-weight: bold }
```

Egy deklarációs blokk egy nyitó kapcsos zárójellel ({}) kezdődik és a hozzá tartozó záró kapcsos zárójelig (}) tart. Köztük 0 (nulla), vagy több, pontosvesszővel (;) elválasztott deklaráció állhat.

Egy deklaráció egy tulajdonságnévből, kettőspontból (:) és egy tulajdosság-értékből áll. Mindegyik körül lehet köz (whitespace). A tulajdonságnév egy (a korábban leírtaknak megfelelő) azonosító. A tulajdosság-értékben bármely karakter szerepelhet, de a zárójelek (()), kapcsos zárójelek ({ }), szögletes zárójelek ([]), egyszeres (') és dupla (") idézőjelek csak párban fordulhatnak elő. A zárójelek, szögletes zárójelek és kapcsos zárójelek egymásba ágyazhatók. Az idézőjelek között található karakterek szöveggént lesznek értelmezve.

Annak érdekében, hogy a jövőben meglevő tulajdonságokhoz új tulajdonságokat és értékeket lehessen hozzáadni, a böngészőknek figyelmen kívül kell hagyniuk bármely érvénytelen tulajdonságnevet, vagy tulajdosság-értéket. Valamennyi CSS1 tulajdonság csak akkor fogadható el érvényesnek, ha megfelel a nyelvtani és szemantikai előírásoknak. Példaként lássuk a következő stíluslap előírást:

```
H1 { color: red; font-style: 12pt }
P { color: blue; font-vendor: any; font-variant: small-caps }
```

```
EM EM { font-style: normal }
```

Az első sor második deklarációjában a '12pt' érvénytelen érték. A második sor második deklarációjában a 'font-vendor' definiálatlan tulajdonság. A CSS értelmező ezeket a deklarációkat kihagyja, így a stíluslapot a következőképp fogja értelmezni:

```
H1 { color: red }  
P { color: blue; font-variant: small-caps }  
EM EM { font-style: normal }
```

Megjegyzések bárhova beilleszthetők, ahol közők (whitespace) előfordulhatnak. A CSS definiál ezenkívül még helyeket, ahol közők előfordulhatnak és megjegyzések beírhatók. A következő szabályok mindig betartandók:

- ? Minden CSS stíluslap kis-nagybetű érzéketlen, kivéve a stíluslap azon részeit, amelyeket nem a CSS vezérel (pl.: a fontcsalád nevek és az url-ek kis-, és nagybetű érzékenyek. A CLASS és ID attribútumok a HTML felügyelete alatt állnak.)
- ? A CSS1-ben a szelektorok (elemnevek, osztályok és ID-k) csak a következő karaktereket tartalmazhatják: a-z, A-Z, 0-9, az Unicode karaktereket 161-255 -ig, valamint a kötőjelet (-); nem kezdődhetnek kötőjellel, vagy számjeggyel; tartalmazhatnak 'függő' karaktereket, és bármely numerikus kóddal jelölt Unicode karaktert (lásd a következő pontot).
- ? A 'balper' jel (\) után következő legfeljebb négy hexadecimális számjegy (0..9A..F) egy Unicode karaktert jelent.
- ? Bármely karakter - hexadecimális számot kivéve - megfosztható speciális jelentésétől, ha elé egy 'balper' jelet helyezünk. Példa: "\" - szöveg, amely egy dupla idézőjelet tartalmaz.

JavaScript

Mi is az a JavaScript?

A JavaScript egy objektum alapú programozási nyelv, melyet a Netscape fejlesztett ki eredetileg LiveScript néven. A LiveScript és a Java ötvözéséből alakult ki a JavaScript, melyet először a Netscape Navigator 2.0-ban implementáltak. Sokan nevezik a nyelvet objektum orientáltnak, ami azonban nem teljesen igaz, hiszen olyan alapvető tulajdonságok, mint például az öröklődés, hiányoznak belőle. A JavaScript a Java appletektől és a Java programoktól eltérően futásidőben kerül értelmezésre. Tehát a böngésző letölti az oldalt, megjeleníti a tartalmát, majd értelmezi a JavaScript (vagy más nyelven írt scriptek) sorait. Már most fontos leszögezni, hogy a JavaScript nem Java! Szintaktikájában és felépítésében ugyan nagyon hasonlít a nevét adó programozási nyelvre, azonban lehetőségei sokkal korlátozottabbak.

JavaScript és a böngészők

Mivel a JavaScript interpretált nyelv, a programunk csak az oldal betöltésekor fog lefutni, addig a HTML kód sorai között pihen. Ennek a megoldásnak az az előnye, hogy a hibás programsorokat könnyedén tudjuk javítani, hátránya viszont az, hogy a fáradtságos munkával megírt scriptünkhöz bárki hozzáférhet, aki megtekinti oldalunkat. Ha egy programot a böngésző futtat, annak vannak pozitív és negatív tulajdonságai is. Előnyként értékelhető az, hogy a scriptünk platformfüggetlen lesz, tehát ugyanúgy fog futni egy Macintosh gép böngészőjében, mint a sokat dicsért Windows-os környezetben. Hátrányai közé sorolható viszont az, hogy a script hibátlan futtatása - a szabványok többféle értelmezésének köszönhetően - erősen a használt böngésző típusának függvénye. Jelentős eltérések fedezhetők fel már csak Windows-os böngészők JavaScript implementációinak összehasonlítása során is. Nézzünk csak meg egy Internet Explorer 5.0-ra optimalizált oldalt

Netscape, vagy Opera böngészőkkel! Jó esetben csak néhány funkció nem működik az oldalon, máskor azonban az egész weblap működésképtelenné válhat. További problémák merülhetnek fel, ha oldalunk látogatója olyan böngészőt használ, mely nem támogatja, vagy nem ismeri a JavaScriptet, bár ennek valószínűsége manapság igen kicsi. E böngészők ugyanis hajlamosak arra, hogy a számukra értelmezhetetlen utasításokat egyszerűen megjelenítik, mintha az a weblap szövegének része lenne, elcsúfítva és feltárva ezzel gondosan elkészített oldalunkat. Hogy ezt elkerüljük az utasításainkat a HTML kód megjegyzései közé kell beillesztenünk: a "<!--" és "-->" szekvenciák közé. Az újabb böngészőket nem zavarja ez a kis csalafintaság, a HTML szabványt pontosan követő régebbi változatok pedig nem zavarodnak össze tőle, mert egyszerűen kommentként értelmezik a valójában script-kódot tartalmazó oldalrészeket. A fentiek miatt nagyon fontos, hogy szem előtt tartsuk a következő dolgokat: scripteket csak korlátozott mértékben alkalmazzunk, és lehetőleg úgy, hogy azt több böngésző segítségével is kipróbáljuk. A másik dolog, hogy ha amennyiben lehetőség van rá, alkalmazzunk statikus HTML megoldásokat a feladatok ellátására és szerver-oldali ellenőrzéseket a bemenő adatok validálására arra az esetre, ha a JavaScript kódunk ezt a feladatot a böngésző vezérője és/vagy beállításai miatt nem képes elvégezni. Hogy egy kicsit érthetőbb legyen: ha script segítségével automatikusan átirányítunk egy oldalt egy másikra (később leírjuk hogyan), akkor tegyünk az oldalra egy linket is, amelyik a másik oldalra vezet, azon böngészők számára, akik nem értik a JavaScriptet. A problémák egy része ugyan a böngésző azonosításával (szintén később) elkerülhető, azonban jobb a scriptek használatával csínján bánni.

JavaScript beágyazása a HTML dokumentumba

Miután megismertük a nyelv történetét és sajátosságait, lássuk az első programot, melynek feladata mindössze annyi lesz, hogy egy kis felbukkanó üzenetablakban megjeleníti a "Hello World!" szöveget.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
alert("Hello World!");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

A példa egyszerű, de ugyanakkor nagyon szemléletes (remélhetőleg). A HTML dokumentumunk fejrészébe illesztettünk be egy programsort a <SCRIPT> tagok közé, így a böngészővel tudattuk, hogy a HTML kódot más nyelven írt scripttel szakítjuk meg. Mivel többféle scriptnyelv is létezik (pl.: VBScript), és ezek közül többet is használhatunk egy dokumentumon belül, a böngészőnek megmondhatjuk, hogy mely scripteket hogyan kell értelmeznie. Ehhez a SCRIPT tag LANGUAGE attribútumát kell beállítanunk. Ha például egyszerűen a JavaScript szót írjuk ide be, az a böngésző számára a JavaScript 1.0-s verzióját jelenti. Ha olyan szolgáltatásokat is használni szeretnénk, melyek ebben a verzióban még nem szerepelnek, az attribútumnak adjuk értékül a JavaScript1.2-t, és máris új funkciókkal bővülnek scriptünk lehetőségei. Térjünk vissza egy pillanatra a programhoz, mely tulajdonképpen egyetlen sorból áll, mely elvégzi a kitűzött feladatot, azaz megjeleníti üzenetünket. Ehhez az ALERT() metódust (ha jobban tetszik függvényt, de legyünk objektum alapúak), használtuk. Az alert eljárás használata rendkívül egyszerű a zárójelek között kell megadnunk a megjelenítendő szöveget. Ha szövegünk statikus, azaz nem változik, idézőjelek között kell beillesztenünk a függvénybe. Az alert segítségével a változók értékeit is megjeleníthetjük, de erről majd később lesz szó.

Legfontosabb események

Egy HTML oldal életében is vannak események, például amikor valaki végre betölti lapunkat, amikor kattint rajta, vagy amikor elhagyja azt. Ezeket az eseményeket, vagy ha jobban tetszik felhasználói interakciókat scriptünk képes lekezelní beépített eseménykezelői segítségével. A legfontosabb eseményeket foglaljuk össze az alábbiakban.

Esemény neve Eseménykezelő Mikor következik be

load	onLoad	a HTML oldal betöltésekor
click	onClick	az objektumra történő kattintáskor
change	onChange	ha az űrlap mezőjének értéke megváltozik
mouseover	onMouseOver	az egérmutató az objektum fölött van

A felsorolás természetesen nem teljes, azonban néhány alapvető ismeretet szerezhetünk belőle. Az első, amit fontos tudni, hogy értelemszerűen nem minden objektumhoz rendelhetünk hozzá minden eseményt (a későbbiekben egy nagyobb táblázat segít majd ebben eligazodni). A második fontos tudnivaló az eseménykezelők helyes leírásának módja, a JavaScript ugyanis - akárcsak az igazi Java - különbséget tesz a kis és nagybetűk között. Ennek megfelelően az eseménykezelők neveit mindig kis kezdőbetűvel, a szóösszetételeknél azonban nagybetűvel kell írni, így lesz az onmouseover-ból onMouseOver.

Használjuk amit tudunk, 2. példa

A következő példában tudjuk is alkalmazni a most megszerzett ismereteinket. Mindössze annyi a dolgunk, hogy egy gombra való kattintáskor írjuk ki a "Rámkattintottál!" szöveget. Ehhez létre kell hoznunk egy HTML formot, amibe beágyazhatjuk a gombot. A gombra való kattintáskor tehát bekövetkezik egy click esemény, mely meghívja az onClick eseménykezelőt, melynek hatására lefut a scriptünk, és kiírja a szöveget. A forráskód tehát a következő:

```
<HTML>
<BODY>
<FORM>
<INPUT TYPE="BUTTON" NAME="GOMB" VALUE="Kattints ide!"
onClick="alert('Rámkattintottál!')">
</FORM>
</BODY>
</HTML>
```

Scriptünk írása közben használhatunk szimpla és dupla idézőjeleket is, és kombinálhatjuk is őket, de mindig ügyeljünk a sorrendre. Ha egy szöveget szimpla idézőjellel kezdtünk, akkor azzal is zárjuk be, egyértelműen kifejezve ezzel a szöveg és az utasítás határait.

Link a Script-re

Ennek a fejezetnek két fontos tanulsága lesz. Az első, hogy a JavaScriptünket nemcsak események hatására hívhatjuk meg, hanem szabványos HTML linkek segítségével is, a második pedig az, hogy ahány böngésző, annyi szokás. Mit is jelent ez? Azt már tudjuk, hogy a Netscape-ben jelent meg először a JavaScript, és mint látványos és jól használható scriptnyelv hamarosan el is terjedt, és több böngészőbe is beépítették. A böngészők fejlesztői azonban nem egységesen implementálták a nyelvet, és ez érdekes megoldásokhoz vezetett. IE 4.0 alatt tesztelve oldalunkat semmi baj nem származik abból, ha egy képhez rendeljük hozzá az onClick eseményt - ami a JavaScript szabályai szerint súlyos hibának számít - sőt, oldalunk még megfelelően is fog működni. Ha azonban ezt az oldalt a Netscape valamely kiadásával tekintjük meg jobb esetben nem történik semmi, ha a képre kattintunk, rosszabb esetben hibaüzenetet is kapunk. Akkor vajon mi a teendő ha egy képhez az onClick eseménykezelőt szeretnénk rendelni, és szeretnénk oldalunkat lehetőleg Netscape és Opera böngészőkkel is élvezhetővé tenni? Ekkor használhatjuk a következő megoldást: képhez ugyan nem, de

linkhez már rendelhetünk Click eseményt. Tehát a képet, mint szabványos linket hozzuk létre viszont href attribútumának a következő értéket adjuk: href="javascript:void(utasítások, függvények)". A következőképpen:

```
<HTML>
<BODY>
<a href="javascript:void(alert('Működik!'))">link, ami akár egy kép is
ehet</a>
</BODY>
</HTML>
```

Tehát létrehoztunk egy linket - ami nem csak szöveg, hanem akár egy kép is lehet - aminek href tagjában megadtuk, hogy a link valójában nem egy másik HTML oldalra mutat, hanem egy JavaScript utasításra, ami esetünkben az alert függvény meghívása. A függvény értéket adhat vissza, és a visszaadott érték a képernyőn jelenne meg, amit viszont a VOID() megakadályoz. Ezt a megoldást csak akkor kell alkalmaznunk, ha a meghívott eljárásnak van visszaadott értéke, tehát esetünkben nem.

Csoportosítsunk, avagy a függvények

A függvényeket teljesen szabadon definiálhatjuk a következő szintaktikával:

```
function fuggveny(valtozo) {
utasitasok
...
}
```

Az ellenőrzést és az adatok elküldését elvégezhetjük egy függvény segítségével, így az eseménykezelő meghívása után csak a függvényünk nevét kell írunk. Ennek a függvénynek átadjuk az űrlap megfelelő mezőjének értékét és kezdődhet is a vizsgálat. Az ilyen függvényeket a HTML dokumentum elején szokás definiálni, elkerülve azt, hogy előbb hívjuk meg a függvényt, mint ahogy a leírása szerepelne. A legcélszerűbb ha függvényeinket még a fejrészben deklaráljuk, így ezek a problémák nem fordulhatnak elő. A függvény írása során egész csomó változóval dolgozhatunk, így fontos hogy ismerjük azok hatáskörét. A változók érvényességi köre attól függ, hogy hol deklaráltuk őket. Ha a függvényeken kívül, akkor az egész dokumentumra érvényes, ha függvényen belül, akkor csak az adott függvényen belül érhetjük el őket. A függvények használatát mutatja be a következő egyszerű példa, mely egy figyelmeztető ablakban írja ki, hogy az OK és a Mégse lehetőségek közül melyiket választottuk.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function dontes() {
if (confirm('Mit választasz?'))
alert('OK');
else
alert('Mégse');
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<input type="button" value="Válassz!" onClick="dontes()">
</FORM>
</BODY>
</HTML>
```

A fenti script értelmezéséhez néhány alapvető ismeret azért szükséges. A CONFIRM() metódus egy választó-ablakot jelenít meg a képernyőn, ahol az OK és a Mégse lehetőségek közül választhatunk. Ha az OK-t választjuk az eljárás igaz értékkel tér vissza, ha a Mégse gombra kattintunk hamis a visszatérési érték. Ennek megfelelően a függvényben lévő feltétel

teljesül, illetve nem teljesül, tehát a választásunknak megfelelő figyelmeztető ablak jelenik meg a képernyőn.

Objektumok, dokumentumok

Tudjuk, hogy a JavaScript objektum alapú, és nem objektum orientált nyelv, hiszen az objektumok egy szinten helyezkednek el, objektumai között mégis szoros, alá és fölérendeltségi viszony figyelhető meg. Az egész HTML dokumentumunk ilyen objektumokból épül fel, melyek alapkontextusát a window objektum adja. Az objektumoknak további objektumai, tulajdonságai és metódusai is lehetnek, amelyekre a pont-operátor segítségével hivatkozhatunk. Lássunk egy egyszerű példát: ha a HTML oldalunk URL-jét akarjuk megtudni, tehát azt az Internet-címet, amin keresztül a lapot elérjük, a document objektum location objektumának értékét kell kiíratnunk: document.location. Az objektumok metódusait is hasonló módon érhetjük el: a dokumentum név mezőjére való fókuszáláshoz például a mezőhöz tartozó FOCUS() metódust kell meghívunk: document.form.név.focus(). Az objektumokat nagyon sokféle módon elérhetjük programozás során, ami megkönnyíti a munkát, viszont nagyon zavaró is lehet, ha valaki nincs tisztában az objektumok hierarchiájával, ezért jól jöhet a következő segítség. A teljes HTML dokumentum objektumok sokaságából épül fel, és a JavaScript segítségével szinte minden egyes objektumnak meg tudjuk változtatni az összes tulajdonságát - hiszen erre találták ki. Építsünk fel egy HTML űrlapot, mely telis-tele van különböző beviteli mezőkkel rádiógombokkal, és checkbox-okkal. Ennek a dokumentumnak a fő objektuma a document objektum. Ennek további objektumai a formok, melyek további objektumokként tartalmazzák a beviteli mezőket. A beviteli mezőknek pedig további metódusai és tulajdonságai lehetnek. Így tehát egy beviteli mező értékét a következőképpen érhetjük el: document.form.mezo.value. Az objektumoknak nem kötelező, de lehet nevet adni. Az objektumok ekkor háromféle módon érhetők el. Az előbbi példánál maradva: a beviteli mező értéke a következőképpen érhető el:

- document.form.mezo.value
- document.forms[0].elements[0].value
- document.forms['form'].elements['mezo'].value

A fenti elérési módozatokat kombinálhatjuk is. Sokat segíthet azonban, ha tudjuk, hogy a dokumentum elemei a lap tetejétől kezdve kerülnek beszámozásra, tehát a document.forms[0] a dokumentum tetejéhez legközelebb eső HTML formot jelenti. Nézzünk egy példát: Az oldal tulajdonságainak módosítása lesz a feladat. Létrehozunk egy formot egy beviteli mezővel, amelynek kiírjuk, majd megváltoztatjuk az értékét. Ehhez két függvényt fogunk használni, egyet a kiíráshoz és egyet a megváltoztatáshoz.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function kiiras(mezo) {
alert(mezo);
}
function valtoztatas() {
document.form.szoveg.value="Megváltozott!";
}
</SCRIPT>
</HEAD>
<BODY>
<FORM name="form">
<input type="text" name="szoveg">
<input type="button" value="Kiir"
nClick="kiiras(document.form.szoveg.value)">
<input type="button" value="Valtoztat" onClick="valtoztatas()">
</FORM>
</BODY>
```

</HTML>

Ahhoz hogy a fent leírt feladatot el tudjuk végezni a KIIRAS() függvény meg kell hogy kapja a beviteli mező értékét meghívásakor, mely jelen esetben a document.form.szoveg.value tulajdonságon keresztül érhető el. A VALTOZTATAS() eljárásnak ezzel szemben nincs szüksége semmilyen paraméterre, hiszen feladata csak annyi, hogy a mező értékét megváltoztassa. A fenti példa ugyan nagyon egyszerű, de vegyük észre, hogy ezzel a módszerrel, tehát egy egyszerű értékadással, módosíthatjuk minden objektum tulajdonságát, dinamikusán változtatva ezzel oldalunk megjelenését. Kicserélhetjük a háttérszínt, de akár a betűk színét is megváltoztathatjuk. Ha a beviteli mezőnek nem adunk nevet, akkor a változtatáskor a fentiekben leírt módon járhatunk el. Ha tehát az VALTOZTATAS() függvényt kicseréljük a következőre, minden ugyanúgy fog működni.

```
function valtoztatas() {  
document.forms[0].elements[0].value="Megváltozott!";  
}
```

Ha hibáztunk

Mi történik, ha hibát vétünk a program írása közben? A többi programozási nyelvben ilyenkor a fordítótól figyelmeztetést vagy hibaüzenetet kapunk. JavaScript esetében a fordítást és a futtatást is egy böngésző végzi, így a hibajelzéseket innen kell várnunk, néhány esetben sajnos hiába. Azért hiába, mert a böngészők "természetesen" ebben is különböznek. Ha a Netscape hibát jelez ("Javascript error!" Felirat jelenik meg az állapot sorban), akkor a címsorba kell beírunk a "javascript:" sort, és máris rámutat a hiba okára. Ez a funkció akkor is jól használható, ha csak ellenőrizni szeretnénk egy-egy utasítást, mivel azokat itt is tesztelhetjük. Írjuk be a címsorba, hogy "javascript:alert('Hiba!)" és máris megjelenik a figyelmeztető ablak. Ez a funkció rendkívül egyszerűen és hatékonyan használható.

Az Explorer szolidabban (egy kis sárga háromszög a status sor sarkában) jelzi a hibánkat, és a hiba okának felderítésében sem jár el a Netscape gondosságával.

Események

Ahhoz, hogy hatékonyan használhassuk a JavaScript nyelvet, szükségünk lesz egész csomó esemény kezelésére, hiszen a felhasználói beavatkozások, tevékenységek irányítják lapunk működését. Az események lekezeléséhez - mint már tudjuk - eseménykezelőket kell használnunk. Ezek egy kis részét már ismerjük, de a programozás során sokkal több esemény megkülönböztetésére lesz szükségünk. Amit fontos még tudni, hogy nem minden esemény rendelhető hozzá minden objektumhoz, elemhez. Hogy tisztán lássuk az események - eseménykezelők - objektumok viszonyát, nézzük meg alaposan az alábbi táblázatot.

Esemény	eseménykezelő	objektum	Akkor következik be...
Blur	onBlur	window, frame, form	elemei ha a fókuszt egy másik objektumra kerül
Click	onClick	gomb, radio-gomb, checkbox, link	ha az objektumra kattintunk
Change	onChange	text, textarea, select	ha megváltozik az adott mező értéke
Focus	onFocus	window, frame, form	elemei ha a fókuszt az elemre kerül
Keydown	onKeyDown	document, image, link, text, textarea	ha egy billentyű lenyomásra kerül
Keyup	onKeyUp	document, image, link, text, textarea	ha egy billentyű felengedésre kerül
Load	onLoad	document	amikor a HTML oldal betöltődik
Mouseout	onMouseOut	area, link	ha az egér elhagyja az objektumot

Mouseoveron	MouseOver	area, link	ha az egér az elem fölé kerül
Select	onSelect	text, textarea	ha a mező valamely részét kiválasztjuk
Submit	onSubmit	form	ha a submit gombra kattintunk

Ha esetleg a fenti táblázat értelmezéséhez íme egy kis magyarázat: az első oszlopban szerepel az esemény neve, a másodikban a hozzá tartozó eseménykezelő, a harmadik oszlop tartalmazza azon objektumok listáját, melyekhez az eseménykezelő hozzárendelhető, és végül a negyedik oszlopban láthatjuk, hogy az adott esemény mikor következik be. Nézzünk egy gyakorlati példát:

Változó képek

Ha a HTML kódban tag-be berakjuk a NAME="kep" attribútumot, akkor képünkre a következő módon hivatkozhatunk: document.kep, a forrásfájlt pedig a document.kep.src tulajdonságon keresztül érhetjük el, illetve változtathatjuk meg. A feladat rendkívül egyszerű lesz: változzon meg a kép, ha a felhasználó fölé viszi az egeret, nyerje vissza eredeti állapotát, ha elvitte onnan, és legyen más akkor is, ha rákattintottunk. Lássuk a programot:

```
<html>
<body>
<a href="javascript:alert('Lenyomva is más a kép, de sajnos hamar
isszaugrik')">

</a>
</body>
</html>
```

Mi történik, ha a cserélendő képek túl nagyok, vagy a felhasználó sávszélessége túl kicsi? Nagyon egyszerű, a képek kisebb nagyobb késéssel fognak megjelenni, hiszen letöltésük csak az első hivatkozás pillanatában fog elkezdődni (azaz pl. a 2.jpg akkor, amikor először mozgatjuk az egeret a kép fölé), és a kicsi sávszélesség miatt akár jó néhány másodpercbe is beletelhet mire a böngészőnek sikerül a képet letöltenie, és végre megjelenítheti azt. Ez időnként elég zavaró lehet. Azonban erre is van megoldás.

Előtöltött képek

Létre kell hoznunk új objektumokat, mégpedig olyanokat, amelyek képek, de nem jelennek meg az oldal betöltődéskor a dokumentum felületén. Ezt az Image objektum használatával érhetjük el. Lássuk először a forráskódot, majd sorról sorra a magyarázatot.

```
<html>
<script>
elsoKep = new Image();
elsoKep.src = "1.jpg";
masodikKep = new Image();
masodikKep.src = "2.jpg";
function kepMutat(forras) {
    if (forras==1) document.kep.src= elsoKep.src;
    else document.kep.src= masodikKep.src;
}
</script>
<body>
<a href="javascript:alert('Előre töltött képekkel...')">

</body>
```

```
</html>
```

A HTML kód remélem már mindenkinek világos: létrehozunk egy képet - ami feltétlenül link kell, hogy legyen -, amely különböző események hatására különböző függvényeket hív meg. A hangsúly itt a függvényeken, illetve a JavaScript kódon van. A script törzsében - tehát a függvény definícióján kívül - létrehozunk két új objektumot, melyek így a függvényből és azon kívülről is elérhetőek lesznek. Ezek az objektumok Image, azaz kép objektumok, melyeket a következőképpen hoztunk létre: `elsoKep = new Image()`. Tehát az `elsoKep` egy új (new) kép objektum lesz (Image), melynek, a következő sorban, az `src` attribútumában megadjuk a képet tartalmazó fájlnek a nevét. Hasonlóan hozzuk létre a második képünket is. Ezzel a módszerrel a képek már az oldal letöltődése közben letöltésre kerülnek a memóriába vagy a böngésző helyi gyorsítótárába, így a képcserekor csak meg kell jeleníteni őket, nem kell azok letöltésére várnunk. Nincs más dolgunk, mint írni egy függvényt, ami a képek attribútumainak változtatásait elvégzi. Erre a feladatra szolgál a `kepMutat` eljárás, ami a paraméterétől függően az `elsoKep` vagy a `masodikKep` objektumokban tárolt képeket jeleníti meg, egy feltételtől függően. A fenti módszerek alkalmazásával - akár többtucat képet is használhatunk - nagyon látványossá tehetünk már egy egyszerű oldalt is, azonban itt sem érdemes túlzásokba esni, és túldíszíteni a lapunkat.

Időzítések

A JavaScript lehetőséget ad az utasítások időzítésére. Ez az időzítés tulajdonképpen azt jelenti, hogy az adott utasítás csak bizonyos idő múlva fog végrehajtódni. Erre a funkcióra sokszor szükségünk lehet, ha valamit késleltetni szeretnénk, például a felhasználónak időt szeretnénk adni egy szöveg elolvasására, mielőtt új oldalt töltenénk be. Az ilyen esetekre találták ki a `setTimeout()` metódust, melynek paraméterei a következők: `setTimeout("utasítás",idő)`. Az eljárás a paraméterül kapott utasítást - melyet mindig idézőjelek közé kell tennünk - csak adott idő múlva hajtja végre. Ha tehát azt szeretnénk, hogy 5 másodperc múlva ugorjunk egy másik oldalra, akkor a következő utasítást kell használnunk: `setTimeout("location.href='uj_oldal.html'",5000)`, ugyanis az idő paraméterül szolgáló számokat milliszekundumokban (azaz ezred másodpercekben) kell megadnunk.

Feladatunk az lesz, hogy egy beviteli mezőben bizonyos késleltetéssel írjunk ki egy szöveget, mintha valaki éppen most gépelne be azt. Lássuk először a forráskódot majd a hozzá tartozó magyarázatot.

```
<html>
<script>
function kiiras() {
    var kiirandoSzoveg="Ezt szépen lassan jelenítjük meg...";
    var kiirtSzoveg=document.form.mezo.value;
    var kiirandoHossz=kiirandoSzoveg.length;
    var kiirtHossz=kiirtSzoveg.length;
    if (kiirtHossz<kiirandoHossz) {
        document.form.mezo.value=kiirandoSzoveg.substring(0,kiirtHossz+
1);
        setTimeout("kiiras()",300);
    }
}
</script>
<body onLoad="kiiras()">
<form name='form'>
<input type='text' name='mezo' size='35'>
</form>
</body>
</html>
```

A HTML kódban mindössze egy beviteli mezőt definiálunk a hozzá tartozó form-mal, és az oldal betöltődésekor meghívjuk a `kiiras` eljárást. Nézzük akkor a függvényünket sorról sorra.

Az első sorban definiálunk egy változót, melyben elhelyezzük a kiírandó szöveget. A második sorban létrehozunk még egy változót, melyben a már kiírt szöveget tároljuk el. Mivel szükségünk lesz mind a kiírandó, mind a kiírt szöveg hosszára, ezért a következő utasítások segítségével ezt a két értéket tároljuk el egy-egy változóban. Következik egy feltétel, mely eldönti, hogy folytassuk-e a kiíratást, vagy befejeződik scriptünk futása, azaz már kiírtuk a kívánt szöveget. Ekkor a kiírt karakterek számát növeljük a substring metódus segítségével, majd újra meghívjuk függvényünket, egy kis késleltetés beiktatásával.

Azt hiszem, nem árt, ha a feltétel igaz ágában szereplő két sort kicsi tovább elemezzük. A kiirandoSzoveg változónak használjuk a substring metódusát, melynek két paramétere van. Az első paraméter mondja meg, hogy hányadik karaktertől kezdve, a második pedig hogy hányadik karakterig írjuk ki a stringet. Tehát ha stringünk a "valami" szóból áll, akkor annak string.substring(0,2) metódusa a "va" szótagot adja vissza értékül. Mivel esetünkben a második paraméter mindig egyel növekszik, így minden lépésben egyel több karakter jelenik meg a kiírandó stringből a képernyőn. Most következik a késleltetés, mely 0,3 másodperc múlva újra meghívja a kiiras() eljárást, és így újabb betű jelenhet meg a beviteli mezőben.

Ezt a módszert, amikor a függvény saját magát hívja meg rekurciónak nevezzük, és ilyen, vagy ehhez hasonló problémák (faktoriális kiszámítása) nagyon jó hasznát vehetjük.

Status sor

A status sorban állandó és változó információk is megjelenhetnek, éppúgy, mint egy beviteli mezőben. A két mezőbe való írás módja sem tér el jelentősen egymástól, így kézenfekvő, hogy az előbbi példában használt szövegkiíró scriptet ültessük át a status sorra. A kód csak annyiban változik, hogy nem kell formot deklarálnunk, és a beviteli mező helyett a scriptünk kimenete a status sor lesz.

```
<html>
<script>
function kiiras() {
    var kiirandoSzoveg="Így készül az internetes futófény, már csak le
kellene törölni...";
    var kiirtSzoveg=window.status;
    var kiirandoHossz=kiirandoSzoveg.length;
    var kiirtHossz=kiirtSzoveg.length;
    if (kiirtHossz<kiirandoHossz) {
        window.status=kiirandoSzoveg.substring(0,kiirtHossz+1);
        setTimeout("kiiras()",100);
    }
}
</script>
<body onLoad="kiiras()">
Ne ide nézz, hanem egy kicsit lejjebb...
</body>
</html>
```

A kódban történt változtatások szerintem maguktól értetődőek, ezért nem is szeretném túlmagyarázni a dolgot. A scriptünkkel - remélhetőleg - csak egyetlen gond van, mégpedig az, hogy ugyan hajlandó kiírni a szöveget, azonban futása ekkor be is fejeződik, és a status sor újra unalmassá válik. Erre a problémára nyújt gyógyírt a következő példa.

SWITCH használat

Nézzünk egy olyan programot, amely eltávolítja egy szövegből az ékezeteket:

```
<html>
<script>
function check() {
```

```

var
utolsoKarakter=document.form.text.value.substring(document.form.text.value.
length-1);
var ujKarakter="";
switch (utolsoKarakter) {
case "é" : ujKarakter="e"; break;
case "É" : ujKarakter="E"; break;
case "á" : ujKarakter="a"; break;
case "Á" : ujKarakter="A"; break;
case "í" : ujKarakter="i"; break;
case "Í" : ujKarakter="I"; break;
case "û" : ujKarakter="u"; break;
case "Û" : ujKarakter="U"; break;
case "ú" : ujKarakter="u"; break;
case "Ú" : ujKarakter="U"; break;
case "ö" : ujKarakter="o"; break;
case "Ö" : ujKarakter="O"; break;
case "ó" : ujKarakter="o"; break;
case "Ó" : ujKarakter="O"; break;
case "ü" : ujKarakter="u"; break;
case "Ü" : ujKarakter="U"; break;
case "ő" : ujKarakter="o"; break;
case "Ő" : ujKarakter="O"; break;
}
if (ujKarakter)
document.form.text.value=document.form.text.value.substring(0,document.form
.text.value.length-1) + ujKarakter;
}
</script>
<body>
Ide lehet bepötyögni a szöveget, amiből majd kiszedjük az ékezetes
betűket:<br>
<form name="form">
<textarea name="text" rows="10" cols="30" onKeyUp="check()"></textarea>
</form>
</body>
</html>

```

Létrehoztunk egy form-ot egy beviteli mezővel, és megadtuk, hogy minden egyes billentyű lenyomása után fusson le az általunk megírt ellenőrző függvény. Metódusunk nem túl bonyolult mindössze egy változó definíciót, egy switch-case szerkezetet, és egy feltételt tartalmaz. Az első lépésben beolvassuk a beviteli mezőben lévő sztringet, és levágjuk róla az utolsó karaktert, hiszen elég ezt vizsgálunk. Ezek után el kell döntenünk, hogy mit tegyünk ezzel a karakterrel. Ehhez egy switch-case szerkezetet használhatunk, mely a következőképpen működik. A switch a paraméterként megkapott változó, switch(valtozo), értékétől függően különböző utasításokat hajt végre. Ezeket az értékeket és a hozzájuk tartozó utasításokat a case "érték" : utasítás szintaktikával definiálhatjuk. A switch sajátossága, hogy minden egyes ilyen sor után, ha nem akarjuk, hogy a következő is végrehajtsódjon, egy break utasítást kell írunk. Ennek megfelelően a case "é" : ujKarakter="e"; break; sor abban az esetben, ha az utolsó leütött karakter "é" volt az ujKarakter változónak az "e" karaktert adja értékül, és befejezi a vizsgálatot. Mivel ezt minden ékezetes betűre elvégeztük, így mindegyik helyére saját ékezet nélküli párja helyettesítődik majd be. Ehhez azonban szükség van még egy lépésre: meg kell vizsgálunk, hogy megváltoztattuk-e a karaktert, azaz az ujKarakter változóba került-e valami. Ha a feltétel igaznak bizonyul, akkor a beviteli mezőben lévő stringhez a függvény által visszaadott karaktert fűzzük, a feltétel hamis volta esetén nem történik semmi.

A múlt hónapban megpróbáltam gyakorlati példákon bemutatni, hogyan is használhatjuk a JavaScript kódokat a weblapunk kialakításakor. Most ezt a hagyományt követve még

gyakorlatibb példákat szeretnék bemutatni, amik valós helyzetben is használhatók, és valódi problémákat oldanak meg. Az előző részben már volt egy kis program, az ékezetellenítő rutin, mely ilyen feladatot oldott meg, ennek most egy továbbfejlesztett változatával is megismerkedhetünk.

For ciklus használata

Az előző verzióknak két, viszonylag nagy, hibája van. Az egyik, hogy csak az utolsó karaktert vizsgálja, így ha valaki utólag szúr be valamit a szövegbe, akkor abban már nem cseréli le az ékezeteket. A másik hiba, hogy ha kivág/beilleszt módszerrel nagyobb szöveget másolunk a beviteli mezőbe, akkor az előbb említett okokból szintén elmarad a kívánt hatás. Mindkét probléma orvosolható a következő módszerrel. Minden egyes bevitelkor, egy *for* ciklus segítségével, az egész szöveget leellenőrizzük. Következzen most a javított változat, és egy kis magyarázat hozzá:

```
<html>
<script>

function check()
{
    var vizsgaSzoveg=document.form.text.value;

    for (var i=1; i<=vizsgaSzoveg.length; i++)
    {
        var vizsgaKarakter=vizsgaSzoveg.substring(i-1,i);
        var ujKarakter="";
        switch (vizsgaKarakter) {
            case "é" : ujKarakter="e"; break;
            case "É" : ujKarakter="E"; break;
            case "á" : ujKarakter="a"; break;
            case "Á" : ujKarakter="A"; break;
            case "í" : ujKarakter="i"; break;
            case "Í" : ujKarakter="I"; break;
            case "u" : ujKarakter="u"; break;
            case "U" : ujKarakter="U"; break;
            case "ú" : ujKarakter="u"; break;
            case "Ú" : ujKarakter="U"; break;
            case "o" : ujKarakter="o"; break;
            case "O" : ujKarakter="O"; break;
            case "ó" : ujKarakter="o"; break;
            case "Ó" : ujKarakter="O"; break;
            case "ü" : ujKarakter="u"; break;
            case "Ü" : ujKarakter="U"; break;
            case "ö" : ujKarakter="o"; break;
            case "Ö" : ujKarakter="O"; break;
        }
        if (ujKarakter)
        {
            vizsgaSzoveg=vizsgaSzoveg.substring(0,i-1) +
            ujKarakter + vizsgaSzoveg.substring(i);
            document.form.text.value=vizsgaSzoveg;
        }
    }
}
</script>
<body>
Ide lehet bepötyögni a szöveget, amiből majd kiszedjük az ékezetes
betűket:<br>
<form name="form">
<textarea name="text" rows="10" cols="30" onKeyUp="check()"></textarea>
</form>
```



```
</body>
</html>
```

Akkor lássuk mi is változott. A HTML *form*, illetve az ékezetek kiszûrését végzõ *switch-case* szerkezet, teljes egészében ugyan az, mint az elõzõ programban. Változott viszont a függvényünk szerkezete, és néhány utasítása. Az elsõ sorban egy egyszerű értékadással kiküszöböljük, a változónevek hosszú, és fáradságos begépelését. Ezek után egy *for* ciklus segítségével végigmegyünk az egész *string*-en, és egyesével levizsgáljuk a karaktereket. A karakterek eléréséhez a *substring* metódust használjuk a megfelelő paraméterezéssel. Megadjuk neki a kezdõ és vég indexet, amelyek közötti részre kíváncsiak vagyunk. A kiválasztás után következik a már ismert vizsgálat.

Ellenõrzések

A valódi internetes oldalak kialakításakor általában szükség van adatok bevitelére a felhasználó részérõl, azonban ezeket az adatok ellenõrizni kell a feldolgozásuk elõtt. Mivel itt kliens oldali programozásról írunk, így az adatok feldolgozásával nem, csak az ellenõrzésükkel foglalkozunk.

Ehhez szükség van HTML ismeretekre is, amit gyorsan átveszünk, hogy érthetõ legyen a probléma, és a megoldás is.

Már eddig is hoztunk létre *form*-okat (magyarra fordítva talán űrlapokat), de a valódi jelentőségükrõl még nem, vagy csak alig esett szó. Ezek az űrlapok biztosítják a felhasználó és a szerver közötti adatcserét. Az így kitöltött adatokat elküldjük a szerverre, ahol valamilyen CGI (Common Gateway Interface) program feldolgozza azt. Az űrlap különbözõ beviteli elemekbõl áll, ezekbõl mutatok be most röviden néhányat:

típus	Tulajdonságok
text	Sortörések nélküli szöveg bevitelére alkalmas mezõ.
textarea	Többsoros szöveg bevitelekor használjuk
select	Választólista, több lehetséges, rögzített érték közül választhatunk.
button	Nyomógomb, ami a felhasználói interakciókat hivatott lekezelni.
submit	Nyomógomb, ami a <i>form</i> adatainak elküldésére szolgál.
hidden	Speciális rejtett mezõ, amiben a szerver oldali program számára fontos adatokat kezelhetünk.

Form néhány eleme, és tulajdonságaik

A példák során csak a *text*, és a *submit*, illetve a *button* objektumokkal fogunk megismerkedni, mivel a feladatok nagy része ezek segítségével jól bemutatható. A késõbbiekben igyekszem majd a többi elem bemutatására is.

Egy egyszerű *form* a következõ elemekbõl áll: beviteli mezõk és *submit* vagy *button* nyomógomb. Általában a *submit* gombot használjuk az adatok elküldésére, de kerülõ úton a *button* gombbal is megoldható ez a feladat.

Lássunk akkor egy *form*-ot, melynek csak annyi a feladata, hogy két mezõ tartalmát elküldi a szervernek (a *<body>* és *<html>* tagokat most elhagyjuk):

```
<form name="form" action="feldolgoz.cgi" method="post">
név: <input type="text" name="nev"><br>
e-mail cím: <input type="text" name="email"><br>
<input type="submit" name="gomb" value="Elküld">
</form>
```

A *form*-nak körülbelül ezek azok a paraméterei amiket mindenképpen érdemes megadni. A *name* értelemszerûen az űrlap nevét jelenti, amivel hivatkozhatunk rá. Az *action* jelenti annak a CGI programnak a nevét, ami a feldolgozást végzi majd. A *method* paraméterben adhatjuk

meg, hogy milyen módon küldjük az adatokat. A *post* érték azt jelenti, hogy az adatokat egy "csomagként" kapja meg a szerver oldali program.

Amit érdemes megfigyelni az a *submit* gomb paraméterezése. A *name* magától értetődően az elem nevét jelenti, a *value* pedig azt a szöveget tartalmazza, ami a gomb felirataként megjelenik.

Ha tehát a gombra kattintunk, a mezők tartalma a *feldolgoz.cgi* programnak adódik át.

A küldés előtt, mint már említettem, ellenőrizni kell az adatokat, és itt használhatjuk fel a *JavaScript*-et.

"Üres szöveg"

A feladatunk az lenne, hogy ellenőrizzük le küldés előtt, hogy valamelyik mező tartalmaz-e üres adatokat. Ha hiányos információt adna meg valaki, akkor figyelmeztessük, és állítsuk le az adatok küldését. Ehhez az előző *form*-ot kicsit át kell alakítanunk, és írni kell hozzá egy ellenőrző függvényt:

```
<html>
<head>
<script>
    function ellenoriz()
    {
        if (document.form.nev.value=="")
        {
            alert("Nem adtál meg nevet!");
            return false;
        }
        if (document.form.email.value=="")
        {
            alert("Nem adtál meg e-mail címet!");
            return false;
        }

        return true;
    }
</script>
</head>
<body>
<form name="form" action="feldolgoz.cgi" method="post">
név: <input type="text" name="nev"><br>
e-mail cím: <input type="text" name="email"><br>
<input type="submit" name="gomb" value="Elküld" onClick="return
ellenoriz()">
</form>
</body>
</html>
```

Lássuk hogyan is működik a dolog. Először is meg kell mondanunk a böngészőnek, hogy az *Elküld* gombra történő kattintáskor hívja meg az *ellenoriz()* függvényt, és figyelje is hogy milyen értékkel tér vissza. A *return* kulcsszóval mondhatjuk meg, hogy a visszatérési értéket figyelembe véve küldjük el a *form*-ot, vagy állítsuk le a folyamatot. Ha tehát a meghívott függvény *true*, azaz igaz értékkel tér vissza, a folyamat tovább fut, ha azonban *false*, azaz hamis értékkel tér vissza a küldés megszakad. Nincs más dolgunk tehát, csak megírni úgy a függvényt, hogy ezeket az értékeket adja vissza. Egy egyszerű *if* szerkezettel, pontosabban kettővel, vizsgáljuk a mezők tartalmát. A vizsgálat történhet a benne lévő tartalom, vagy a tartalom hossza szerint, én az előbbit választottam. Tehát ha a mező üres, akkor egy üzenetet küldünk a felhasználó felé, majd hamis értékkel térünk vissza. Ha mindkét vizsgálaton túljutottunk, az azt jelenti, hogy nem volt hiba, és visszaadhatjuk az igaz értéket.

Itt szeretném megemlíteni, hogy az ellenőrzést más helyen is lehet végezni, a másik eljárás egyébként a *form onSubmit* eseményét használni.

Az előbbi pár oldaban próbáltam meg bemutatni a JavaScript alapvető tulajdonságait. A JavaScriptben lévő objektumok metódusainak pontos leírása a JavaScript referenciákban teljes mértékben megtalálható.