HTML, CSS – böngészők programozása

Informatika 2 – Hallgatói segédlet

1. Szükséges előismeretek

A gyakorlat célja a webes megjelenítés alapját képező HTML és CSS nyelvek begyakorlása. A feladatok építenek ezeknek a nyelveknek az alapszintű ismeretére. Ez a hallgatói segédlet ezeket az ismereteket mutatja be.

Az ismertető a Honlapépítés a XXI. században című könyv bevezője alapján készült. A könyv letölthető a webről: http://webmatrix.devportal.hu/Home/Tananyag

2. A HTML – avagy milyen nyelven értenek a böngészők

A HTML gyökerei a nyolcvanas évek végére nyúlnak vissza, amikor a svájci CERN1-ben dolgozó fizikus, Tim Berners-Lee azzal a problémával szembesült, hogy a korabeli dokumentum formátumok nem megfelelőek a fizikusok által előállított kutatási eredmények megjelenítésére. Az egyik fájó pont a képek, ábrák, illusztrációk és általánosságban a gazdag formázási lehetőségek hiánya volt, a másik pedig az, hogy a kutatási eredmények általában egymásra épülnek, kapcsolódnak egymáshoz, amit az akkori fájlformátumok nem tudtak kezelni. Ezen problémák megoldására Tim Berners-Lee két ötlettel állt elő, melyek a mai napig meghatározzák a HTML nyelvet:

1. A dokumentum ne csak egyszerű szöveg legyen, hanem a tartalmat hordozó részt lássuk el címkékkel, melyek kiegészítő információkat (metaadatokat) adnak a szöveghez. Hasonló célú egyszerűbb jelöléseket a nyomdatechnikában már korábban is használtak (pl. egyszeres aláhúzás: dőlt, kettős aláhúzás: félkövér betű), itt a nagy újdonság a formalizmus és címkekészlet megalkotása volt, így jött létre a HTML mint jelölőnyelv (markup language).
2. A dokumentumok „mutassanak túl” önmagukon, azaz olyan hiperszövegeket (hypertext) tartalmazzanak, amelyek bármely részlete egy másik dokumentumra hivatkozhat. Ez az, amit ma úgy ismerünk, hogy egy weboldalon rákattinthatunk hivatkozásokra (linkekre), aminek hatására betöltődik egy másik weboldal.

2.1. A HTML nyelv felépítése

A HTML nyelv jelölőrendszere elemekből és attribútumokból épül fel. Az elemekkel tudjuk felcímkézni a szöveg egyes részeit, az attribútumokkal pedig ezeknek a címkéknek a tulajdonságait tudjuk meghatározni.

Az elemek (angolul element) a szöveget formailag egy kezdő címke (start tag) és egy záró címke (end tag) közé zárják:

```
<címke>szöveg</címke>
```

Az elem elején a kezdő címkét „kacsacsőrök” közé kell tennünk; a záró címkét szintén, de ott még egy per-jelre is szükség van, hogy egyértelművé tegyük, az az elem vége. A címkék természetesen nem fognak megjelenni a weboldalon, hanem csak kiegészítő információkat hordoznak, amiket a böngésző értelmez.

A címke viselkedését attribútumokkal tudjuk meghatározni, melyekből több is kapcsolódhat egy elemhez:

```
<címke attribútum1="érték1" attribútum2="érték2">szöveg</címke>
```

Az attribútumokat és értékeiket a kezdő címkénél adhatjuk meg, méghozzá úgy, hogy az értéket idézőjelek ("érték") vagy aposztrófok ('érték') közé tesszük.

A legtöbb elem a fenti formában egy szöveg formáját vagy szerepét (pl. címsor) határozza meg, de vannak más jellegű elemek is (pl. sortörés, kép hivatkozás). Ezért formailag léteznek ún. önbezáró címkék (self-closing tag), amelyeknek ilyen egyszerű a formájuk:

```
<címke />
```

Természetesen ehhez is kapcsolódhatnak attribútumok:

```
<címke attribútum1="érték1" attribútum2="érték2" />
```

Van lehetőség arra, hogy a HTML kódba megjegyzéseket (comment) tegyünk annak érdekében, hogy később emlékezzünk arra, mit miért csináltunk, vagy hogy éppen a weboldal melyik részének a kódját látjuk:

```
<!-- Itt következik a lábléc -->
```

Természetesen ezek a megjegyzések sem jelennek meg a böngészőben, de bárki számára láthatóak, aki a böngészőben kiválasztja a View Source menüpontot, ezért bizalmas információkat még véletlenül se írjunk ide.

> JÓ TUDNI: A HTML szabvány a fentieknél „lazább” szintaktikát is megenged. Például bizonyos esetekben a záró címke elhagyható (pl. új bekezdés kezdete egyben az előző végét is jelenti), az attribútumok értékeit nem kötelező idézőjelek közé tenni és az önbezáró elemeket sem kötelező jelölni. Ezek a lazán formázott HTML dokumentumok azonban sok bosszúságot okoznak a HTML szerkesztő programoknak és a HTML kódot értelmező böngészőknek is, ezért ma már inkább XHTML kódot szokás írni, amely az XML (eXtensible Markup Language) nyelv szigorú formai szabályait követi. A fent ismertetett szintaktika megfelel ezeknek az előírásoknak.

2.2. Fontosabb HTML elemek

Miután áttekintettük az elemek, címkék és attribútumok formáját, ismerkedjünk meg a legfontosabb HTML elemek jelentésével és használatával! Fontos látni, hogy a HTML nyelvben az elemek nevei rögzítettek, nem találhatunk ki találomra újabb elemeket, azokat kell használnunk, amik a szabványban megtalálhatóak.

2.2.1. A HTML dokumentum struktúrája

A HTML dokumentum első sora az ún. Document Type Definition, vagyis a DTD. A DTD határozza meg a böngésző számára, hogy a dokumentum a HTML szabvány mely verzióját követi, így a böngésző pontosan tudni fogja, hogy a dokumentumban milyen HTML elemek megengedettek, és melyiknek mi a jelentése.

Mi a gyakorlatok során a legújabb, ötös HTML verzióban előírt DTD-t fogjuk használni:

```
<!DOCTYPE html>
```

Fontos, hogy a HTML fájlunk elején szerepeljen ez a sor, különben előfordulhat, hogy a böngésző nem úgy jeleníti meg az oldalt, ahogy szeretnénk.

A !DOCTYPE után következik az oldal ún. gyökér eleme (root element), a html elem, ami tartalmazza a dokumentum fejlécét (head) és törzsét (body):

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Ez a fejléc-->
  </head>
  <body>
    <!-- Ez a szövegtörzs -->
  </body>
</html>
```

A fejlécben kap helyet a dokumentum címe, amit a böngésző az ablak címsorban megjelenít (title elem) és további leíró információk (meta elem), amik viszont nem jelennek meg az oldalon, például:

```
<head>
  <title>Bevezetés az internet programozásába</title>
  <meta name="author" content="Balássy György" />
  <meta http-equiv="Content-Language" content="hu" />
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
```

2.2.2. A szöveg struktúrája

Ha egy HTML dokumentumban rendezni, formázni szeretnénk a szöveget, akkor ezt kizárólag HTML elemekkel tehetjük meg. Azaz hiába teszünk sortöréseket a HTML kódba, vagy hiába teszünk egymás után sok szóközt, a böngészőben csak egyetlen szóköz fog megjelenni helyette. Ha másként szeretnénk, erre szolgáló címkéket kell használnunk.

Egy hosszabb szöveget a p (paragraph) elem segítségével tagolhatunk bekezdésekre:

```
<p>Első bekezdés</p>
<p>Második bekezdés</p>
```

Ha egy bekezdésen belül új sort szeretnénk kezdeni, akkor azt a br (break) elemmel tehetjük meg. A br elem önállóan áll, nincs sem záró címkéje, sem pedig attribútumai, mindössze ennyi:

```
<br />
```

A bekezdések közé címsorokat tehetünk, méghozzá hat méretben, melyeket a h1..h6 (heading) elemekkel jelölhetünk:

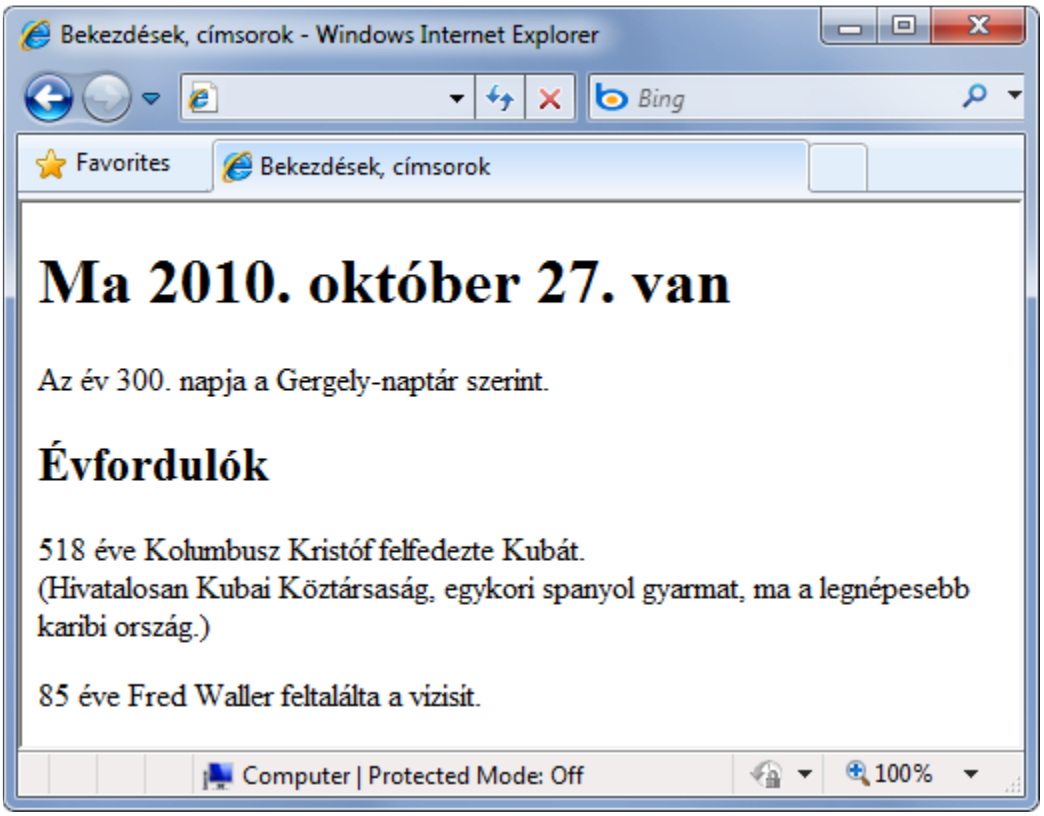
```
<h1>Főcím</h1>
<h2>Alcím</h2>
```

Rakjuk össze ezeket egyetlen példába:

```
<!DOCTYPE html>
<html>
<head>
<title>Bekezdések, címsorok</title>
</head>
<body>
<h1>Ma 2010. október 27. van</h1>
Az év 300. napja a Gergely-naptár szerint.
<h2>Évfordulók</h2>
```

```
<p>
518 éve Kolumbusz Kristóf felfedezte Kubát.<br />
(Hivatalosan Kubai Köztársaság, egykori spanyol gyarmat, ma a legnépesebb
     karibi ország.)
</p>
<p>85 éve Fred Waller feltalálta a vízisít.</p>
</body>
</html>
```

Mindez így jelenik meg a böngészőben:



Érdemes megfigyelni, hogy a böngésző nagyobb betűkkel jeleníti meg a címsorokat, a bekezdések között automatikusan hagy távolságot, sőt a hosszú sorok tördeléséről is gondoskodik. Minden HTML elemre igaz, hogy van egy alapértelmezett megjelenési módja, amit természetesen megváltoztathatunk, ha mi másként szeretnénk. Erről részletesebben a Cascading Style Sheets c. fejezetben lesz szó.

Ha listákra van szükségünk, akkor kétféle elemet használhatunk attól függően, hogy számozott vagy csak egyszerű felsorolásos listát szeretnénk megjeleníteni az oldalon. A számozott listák keretét ol (ordered list) elemek jelentik, amin belül az egyes lista tételeket li (list item) címkék jelzik:

```
<ol>
  <li>Első</li>
  <li>Második</li>
  <li>Harmadik</li>
</ol>
```

Ha nincs szükségünk a sorszámokra, akkor az ul (unordered list) elemmel egyszerű felsorolásos listát készíthetünk:

```
<ul>
```

```
  <li>Piros</li>
  <li>Fehér</li>
  <li>Zöld</li>
</ul>
```

Igény szerint a felsorolásokat és számozott listákat több szint mélyen akár egymásba is ágyazhatjuk, a böngésző automatikusan gondoskodni fog a megfelelő sorszám vagy lista ikon megjelenítéséről.

A szöveg tagolására használható még a hr (horizontal rule) elem is, amely egy vízszintes vonallal választja el a felette és alatta lévő tartalmat. A br elemhez hasonlóan a hr is önbezáró:

```
<hr />
```

A használata nagyon egyszerű:

```
Felső tartalom
<hr />
Alsó tartalom
```

2.2.3. Hivatkozások

Az egyes weboldalaink között a kapcsolatot hiperhivatkozásokkal (hyperlink) teremthetjük meg, amit az a (anchor=horgony) elem valósít meg:

```
<a href="http://www.bme.hu"
  title="Ugrás a BME oldalára"
  target="_blank">BME honlap</a>
```

A nyitó és a záró címke között szerepel az a szöveg, ami meg fog jelenni a weboldalon, a href attribútumban pedig azt a webcímet kell megadnunk, amire ezzel a hivatkozással ugrani lehet. A title attribútumba olyan segítség írható, amely megjelenik a böngészőben, amikor a felhasználó a hivatkozás fölé viszi az egeret (ún. tooltip). A target attribútumban azt adhatjuk meg, hogy hova töltődjön be a hivatkozott weboldal. Ha az értéke _blank, akkor a böngésző új ablakban fogja megnyitni az oldalt.

Szintén az a elem használható oldalon belüli ugrások, például tartalomjegyzék vagy az oldal tetejére mutató hivatkozás létrehozására. Ehhez meg kell jelölnünk a hivatkozni kívánt részt az id attribútummal:

```
<a id="LapTeteje" />
```

Majd az erre mutató linket be kell szúrnunk a kívánt helyre:

```
<a href="#LapTeteje">Ugrás az oldal tetejére</a>
```

Az a elemen belül tetszőleges elem állhat, például ha egy képet szeretnénk kattinthatóvá tenni, akkor a kép megjelenítésére szolgáló img elemet tehetjük a link belsejébe:

```
<a href="http://www.bme.hu">
<img src="logo.gif" />
</a>
```

2.2.4. Szemantikai elemek

Az eddig bemutatott elemek többnyire a szöveg megjelenésén változtattak. A továbbiakban lássunk pár olyan HTML elemet, amelyek jelentést (szemantikát) is társítanak a szöveghez! Láttunk már erre

példát a címsorok esetén, hiszen a <h1> elem azonkívül, hogy megnagyobbítja a szöveget még azt is jelenti, hogy az a pár szó az oldal címe.

Hasonlóan az és a elem nemcsak dőlten (em) és félkövéren (strong) jeleníti meg a tartalmazott szöveget, hanem egyben azt is jelenti, hogy ez hangsúlyozott illetve kiemelt tartalom.

A <cite> elem nemcsak hogy alapértelmezés szerint sok böngészőben dőlten jeleníti meg a tartalmazott szöveget, de azt is jelenti, hogy egy hivatkozott forrásról van szó:

A külső forrásból átvett rövidebb idézeteket a <q> elemmel, a hosszabbakat pedig a <blockquote> elemmel jelezhetjük (ez utóbbiban több bekezdés is lehet, <p> elemekkel jelölve), mindkét esetben az opcionális cite attribútummal hivatkozhatunk a forrás URL-jére:

```
<q cite="http://hu.wikiquote.org/wiki/Woody_Allen">
  Nem vagyok nagy ivó. Szilveszterkor két martini után megpróbáltam elrabolni
  és Kubába téríteni egy liftet.
</q>
```

A rövidítésekhez és mozaikszavakhoz az <abbr> és <acronym> elemeket használhatjuk. Ezeknek az elemeknek a törzse mindkét esetben a rövid verzió, a kifejtést a title attribútumban adhatjuk meg. A title attribútumra általában az jellemző, hogy a böngészők egy felugró tipp ablakban jelenítik meg a tartalmát, így ha a rövidítés fölé viszi a felhasználó az egeret, akkor rögtön látja a rövidítés feloldását is:

```
<acronym title="National Aeronautics and Space Administration">NASA</acronym>
```

Ezek a szemantikai HTML elemek (és a többi, amit itt nem tudtunk felsorolni) nagyban hozzájárulnak ahhoz, hogy a weboldalaink ne csak az emberi szem számára szépen megjelenő szövegek legyenek, hanem a szöveg egyes részeinek szerepe a feldolgozó programok - például a keresőmotorok vagy a vakok és gyengénlátók által használt képernyőolvasó szoftverek - számára is egyértelmű legyen.

2.2.5. Táblázatok

A HTML nyelvben egy egész sor elem szolgál arra, hogy táblázatokat tudjunk megjeleníteni az oldalainkon. Egy három soros és két oszlopos táblázatot így készíthetünk el:

```
<table summary="Kiadások">
  <thead>
    <tr>
      <th>Hónap</th>
      <th>Összeg</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Összesen:</td>
      <td>600</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Január</td>
      <td>100</td>
    </tr>
```

```
    <tr>
      <td>Február</td>
      <td>200</td>
    </tr>
    <tr>
      <td>Március</td>
      <td>300</td>
    </tr>
  </tbody>
</table>
```

A táblázatot a <table> elem jelzi, melyben az egyes sorokat <tr> (table row), azon belül az egyes cellákat pedig <td> (table data) elemek definiálják.

Ha több cellát szeretnénk összevonni vízszintesen vagy függőlegesen, akkor azt a <td> elem colspan és rowspan attribútumaival tehetjük meg, ahol értékként azt kell megadnunk, hogy hány cellát kívánunk összevonni:

```
<table>
  <!-- Első sor két cellával -->
  <tr>
    <td>első</td>
    <td>második</td>
  </tr>
  <!-- Második sor egy cellával -->
  <tr>
    <td colspan="2">összevonva</td>
  </tr>
</table>
```

A cellák között a <th> (table header) elemekkel különböztethetjük meg a fejléc cellákat. Ha precízen jelölni szeretnénk, hogy sor vagy oszlop fejlécről van szó, akkor a scope="col" vagy scope="column" attribútumot kell használnunk.

Ha nagyobb táblázatról van szó, akkor azon belül célszerű jeleznünk a fejlécet, a láblécet és a táblázat törzsét, ami például az oldal kinyomtatásakor lehet hasznos információ a böngészőnek. A táblázatnak ezeket a részeit a <thead>, a <tbody> és a <tfoot> elemek jelölik. Elsőre kicsit szokatlan lehet, hogy a <tfoot> elemnek a <tbody> előtt kell állnia, hogy a böngésző a feldolgozás során időben hozzáférjen az abban szereplő információkhoz.

2.2.6. Űrlapok

A weboldalak nemcsak információk megjelenítésére használhatóak, hanem adatok bekérésére is szolgálhatnak. Ilyenkor űrlapokat kell készítenünk, amiket a felhasználó kitölt, majd az adatokat a böngésző egy HTTP POST kéréssel eljuttatja a szerverre. Hogy a szerver oldalon mi történik az elküldött adatokkal, az a webszerveren futó alkalmazás feladata – erre fogunk példákat látni a PHP gyakorlaton.

Íme, egy egyszerű űrlap:

```
<form action="urlap.aspx" method="post">
  <label for="veznev">Vezetéknév:</label>
  <input id="veznev" type="text" />
  <br />
```

```
  <label for="kernev">Keresztnév:</label>
  <input id="kernev" type="text" />
  <br />
  <input type="submit" value="Mehet" />
</form>
```

A <form> elem jelzi a böngésző számára, hogy itt egy űrlapról van szó. A <form> elem action attribútumában azt az URL-t kell megadnunk, ahova a böngésző a megadott adatokat el fogja küldeni, a method attribútumban pedig azt, hogy HTTP GET vagy POST formában várja a szerver az adatokat. Ezek nélkül az űrlap nem sokat ér.

Az űrlapon belül a mezőket és a mezőkhöz tartozó címkéket célszerű összekapcsolni egymással, hogy logikailag is legyen kapcsolat, ne csak a megjelenítéskor kerüljenek egymás mellé. A címkéket a <label> elem hordozza, a mezőkhöz tartozó beviteli vezérlők többségét pedig az <input> elem jeleníti meg. E kettő között úgy lehet kapcsolatot teremteni, hogy az <input> elem id attribútumában megadunk egy tetszőleges, az oldalon belüli egyedi azonosítót és erre hivatkozunk a <label> elem for attribútumában.

Azonosítóként bármit használhatunk, de a bevált gyakorlat szerint olyan azonosítót választunk, ami rövid, egyértelmű, illetve nem tartalmaz ékezeteket és szóközöket.

Az <input> elem type attribútumában kell megadnunk, hogy pontosan milyen beviteli mezőre van szükségünk. Ezek a lehetséges értékek:

- text: szövegdoboz
- checkbox: jelölőnégyzet
- radio: rádiógomb
- password: jelszó mező (szövegdoboz, de nem látszanak a beírt karakterek)
- file: fájl feltöltés
- hidden: rejtett

Az input vezérlővel gombokat is tehetünk az űrlapunkra, szintén a type attribútum beállításával:

- submit: elküldi az űrlap tartalmát a <form> elemben megadott címre
- reset: kitörli az összes űrlap mezőbe beírt értéket
- button: egyszerű nyomógomb, JavaScript kódot kell írnunk, amivel tetszőleges feladatot bízhatunk a gombra

Az input vezérlőnek van még két hasznos attribútuma. A title attribútumba olyan súgó szöveget írhatunk, ami akkor jelenik meg, ha a felhasználó a vezérlő fölé viszi az egeret. Az accesskey attribútumban pedig egy billentyűt határozhatunk meg, amivel a felhasználó (ha az ALT gombbal együtt, vagy más böngészőkben a SHIFT+ALT gombbal együtt nyom le) gyorsan „beleugorhat” a szöveg kurzorral a mezőbe. Így az űrlap nemcsak egérrel lesz kezelhető, hanem a billentyűzet használatával is mozoghatnak a felhasználók az egyes mezők között.

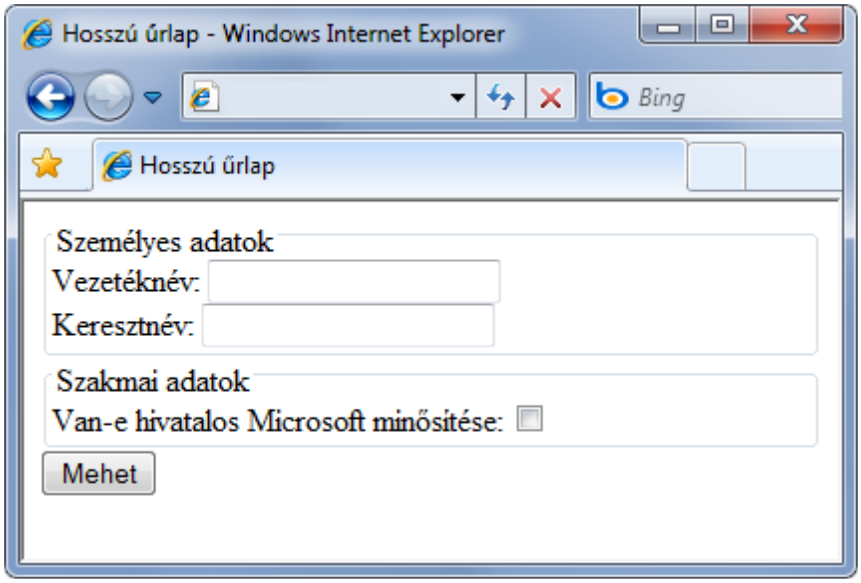
Az input vezérlőn kívül gyakran használunk még legördülő listákat, hogy a felhasználónak ne kelljen gépelnie, hanem előre meghatározott értékek közül választhasson ki egyet. A lehetséges értékeket a <select> elemen belül elhelyezett <option> elemekkel kell felsorolni:

```
<select id="szinek">
  <option value="P">Piros</option>
  <option value="F">Fehér</option>
  <option value="Z">Zöld</option>
</select>
```

Mikor a felhasználó választ egy elemet és elküldi az űrlapon megadott adatokat a szerverre, az <option> elem value attribútumában megadott értéket fogja megkapni a szerver, ezért fontos ennek az attribútumnak az egyértelmű, program által könnyen feldolgozható értékekkel történő kitöltése.

Ha egy űrlap hosszú, akkor célszerű azt részekre bontani, erre szolgál a <fieldset> elem, amin belül az egyes részeknek a <legend> elemmel adhatunk címet:

```
<form action="urlap.aspx" method="post">
<fieldset>
<legend>Személyes adatok</legend>
<!-- Ide jönnek a személyes adatok mezői -->
</fieldset>
<fieldset>
<legend>Szakmai adatok</legend>
<!-- Ide jönnek a szakmai adatok mezői -->
</fieldset>
</form>
```



2.3. Validálás

Az előző fejezetekben számos elem szerepét és lehetőségeit tekintettük át, ám ezek mellett a HTML nyelv még sok, ritkábban használt elemet definiál. Azon kívül, hogy a HTML nyelvben csak ezeket az előre definiált elemeket használhatjuk, van még egy fontos megkötés, mégpedig az elemek egymáshoz viszonyított helyzetére vonatkozóan: a HTML szabvány szigorúan meghatározza, hogy az egyes elemeknek mely elemek lehetnek a gyermekei. Például nem lehet <p> elemeket egymásba ágyazni – szerencsére nem is lenne értelme bekezdésen belül bekezdést írni.

Sajnos nem mindenhol ennyire egyszerű és logikus a helyzet, ezért célszerű valamilyen programmal ellenőrizni, hogy az általunk elkészített HTML kód helyes-e. Erre kiválóan használható a W3C által biztosított Markup Validation Service (http://validator.w3.org). Ez egy ingyenes és nyilvános szolgáltatás, amelyre feltölthetjük az elkészített HTML oldalunkat, és válaszul egy részletes jelentést találunk a benne található hibákról. Ha a kódunk helyes, akkor pedig instrukciókat kapunk arra vonatkozóan, hogyan jelezhetjük oldalunk kiválóságát egy erre szolgáló logó segítségével.

3. Cascading Style Sheets stíluslapok – avagy mitől lesz szép az oldal

Egy weboldal készítésekor alapvető elvárás, hogy az oldalnak olyan arculatot adhassunk, amilyet csak szeretnénk. Nemcsak strukturálni szeretnénk a tartalmat, hanem formázni is, azaz szeretnénk megadni, hogy az egyes részletek, címek, bekezdések, képek hol és hogyan jelenjenek meg. A HTML nyelvben erre természetesen már a kezdetektől fogva van lehetőség. Egyes elemek tartalmazhatnak megjelenítésre vonatkozó attribútumokat (pl. align: igazítás), vagy például a font elemmel megadhatjuk a betűtípust, betűméretet és a színt. Íme egy példa:

```
<p align="center">
<img src="logo.jpg" border="1" />
<font size="10" color="red">
Ez itt egy nagy piros szöveg középre igazítva.
</font>
</p>
```

Ez a megközelítés működőképes, de nem túl szép, ugyanis a tartalom és a megjelenés nagyon keveredik egymással, amiből számos kellemetlenség következik:

- Egy teljes oldal forráskódját nagyon nehéz átlátni, hiszen a tiszta tartalmat felszabdalják a megjelenítésért felelős címkék és attribútumok.
- Nehéz következetes dizájnt készíteni, hiszen például a címsorokat minden esetben helyben kell megformáznunk, ügyelve arra, hogy mindenhol következetesen ugyanúgy nézzenek ki.
- Mivel minden oldalon újra és újra definiáljuk ugyanazokat az arculati beállításokat, ezért a hálózati forgalom jelentősen megnövekedhet.
- Nem lehet a webhely arculatát egyszerűen megváltoztatni, hiszen a dizájn részletei a tartalom között bújnak meg.

A nyilvánvaló megoldás az lenne, ha a HTML kódból ki tudnánk emelni a megjelenítésért felelős elemeket és attribútumokat, helyettük a szövegszerkesztőkben megszokott módon stílusokat definiálnánk, és a szövegben csak az így kialakított stílusokra hivatkoznánk. Szerencsére erre van megoldás, pont erre szolgál a Cascading Style Sheets (CSS) technológia, sőt ma már ez a javasolt és elvárt megközelítés a dizájn és az egységes arculat kialakítására.

3.1. CSS attribútumok

Cascading Style Sheets stíluslapokkal dolgozva a beállításokat attribútum név – attribútum érték párosokkal adhatjuk meg. Az attribútumok neve és értéktartománya rögzített, és gyakorlatilag az összes megjelenéssel kapcsolatos területet (méretek, távolságok, színek, pozíciók stb.) lefedik, így az ide vonatkozó HTML attribútumokra már nincs is szükség. A HTML és CSS attribútumok elnevezése néhol teljesen egyező, máshol csak hasonló. Például a fenti kódrészletben használt color attribútum létezik CSS-ben is, az align="center" HTML attribútum helyett viszont a text-align CSS attribútumot kell használnunk:

```
text-align: center;
color: red;
```

Formailag annyi megkötés van, hogy itt nincs szükség idézőjelekre, az egyenlőségjel helyett pedig kettőspontot kell használnunk. Egyszerre több CSS attribútumnak is adhatunk értéket, ebben az esetben pontosvesszővel kell elválasztanunk a név-érték párosokat egymástól.

Ha megjegyzést szeretnénk írni a CSS kódunkba, akkor azt /* és */ jelek között tehetjük meg:

```
/* Kiemelés */
color: red;
font-weight: bold;
```

3.2. Elhelyezés

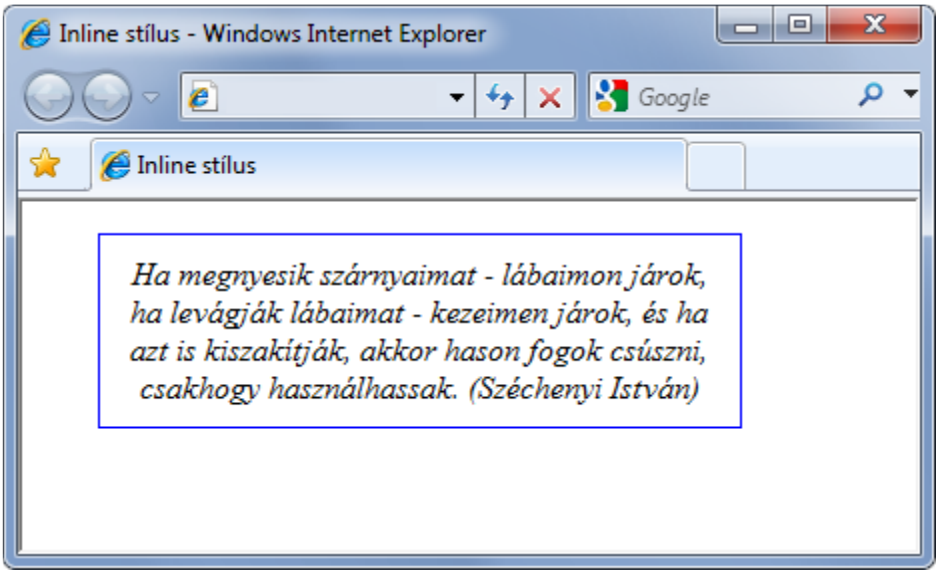
A következő kérdés nyilvánvalóan az, hogy ezeket a CSS beállításokat hogyan kapcsoljuk a HTML kódunkhoz. Erre háromféle lehetőségünk van:

Használhatunk ún. inline stílust, ekkor a formázandó HTML elem style attribútumába kell írnunk a CSS attribútumokat és azok értékeit:

```
<p style="font-style: italic; border: 1px solid blue; margin-left: 30px;
width: 300px; padding: 10px; text-align: center;">
  Ha megnyesik szárnyaimat - lábaimon járok, ha levágják lábaimat – kezeimen
  járok, és ha azt is kiszakítják, akkor hason fogok csúszni, csakhogy
  használhassak. (Széchenyi István)
</p>
```

Az inline stílusnak megvan az az előnye, hogy bárhol használhatjuk, minden komolyabb előkészítés nélkül. A hátránya pedig természetesen az, hogy így a stílusbeállításaink továbbra is több helyen lesznek az oldalon, többszörösen növelik az oldal méretét, nekünk kell szinkronban tartani őket stb.

A fenti stílusbeállításoknak köszönhetően az egyszerű bekezdés így jelenik meg a böngészőben:



Egy másik lehetőség a CSS kód elhelyezésére a stílus blokk használata. Ebben az esetben tipikusan a HTML oldal head elemében hozunk létre egy style elemet, és oda írjuk a CSS kódunkat. Ekkor persze felmerül az a kérdés, hogy honnan fogják tudni az oldal egyes részei, hogy mely beállítások vonatkoznak rájuk, dekésőbb visszatérünk. Egyelőre csak a style blokk szintaktikáját akarjuk megismerni:

```
<style type="text/css">
/* A body elem és a gyermek elemeinek stílusa */
body
{
font-family: Verdana, Arial, Sans-Serif;
font-size: 75%;
background-color: #f4e7bb;
```

```
}
</style>
```

A stílus blokk lehetőséget ad arra, hogy a HTML oldalban egy helyre koncentráljuk a megjelenítésre vonatkozó részeket, így ha esetleg át kell szabnunk az oldal arculatát, egy központi helyen tehetjük meg azt a CSS kód átírásával. Az a hátrány azonban így is megmarad, hogy a webhely által használt összes oldal között nekünk kell szinkronban tartani a CSS blokkokat.

Ezen segít a harmadik megoldás, amikor nem a HTML fájlban helyezzük el a CSS kódot, hanem egy külső fájlban, aminek tipikusan a .css kiterjesztést szoktuk adni. Ekkor a HTML oldal head elemében egy link elemmel hivatkozhatunk az oldalhoz tartozó stíluslapra:

```
<link href="style.css" rel="stylesheet" type="text/css" />
```

Ez a legpraktikusabb megoldás, hiszen ilyenkor a teljes webhelyünkhöz tartozó összes stílusbeállítás egyetlen (vagy néhány) fájlban van, erre hivatkozik az összes oldal, így ha módosítani kell valamit, akkor azt elég egyetlen helyen megtenni.

Jogosan teheti fel bárki a kérdést, hogy mi van akkor, ha mind a három megoldást egyszerre használom? Ebben az esetben a böngésző alkalmazni fogja az összes stílusbeállítást, mégpedig a következő sorrendben, az általánostól a specifikusabb irányba haladva:

1. Külső CSS fájl
2. Oldalon belüli stílus blokk
3. Az elemhez tartozó stílusbeállítások

Ez azt jelenti, hogy ha egy elemre a külső CSS fájlban megadunk egy beállítást, majd ugyanarra az elemre és ugyanarra a tulajdonságára (például háttérszín) az oldalon belül megadunk egy másik beállítást, akkor az oldalon belül definiált felülírja a külső fájlban megadott értéket. Úgy is mondhatnánk, hogy a közelebbi nyer.

3.3. A CSS szelektorok

Az előző fejezetben a stílus blokknál már érintettük azt a problémát, hogy meg kell adnunk, hogy egy definiált stílus pontosan melyik HTML elemre vagy elemekre vonatkozzon. Erre szolgálnak a CSS szelektorok (CSS selectors).

Ha azt akarjuk, hogy egy beállítás az összes HTML elemre érvényes legyen, akkor a * (csillag) szelektort kell használnunk. Például a keretet (border), a külső- (margin) és a belső margót (padding) így szüntethetjük meg (a vastagságuk ill. a szélességük méretét kell számmal megadni):

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

Látható, hogy formailag annyi követelmény van, hogy miután egy szelektorral meghatároztuk a kiválasztandó elemek körét, a rá vonatkozó stílusbeállításokat kapcsos zárójelek között kell megadnunk.

Hivatkozhatunk a HTML elem nevére is, ebben az esetben az összes helyen, ahol az adott elemet használjuk, automatikusan a megadott stílussal fog megjelenni. Például a második szintű címsor (h2) elé (fölé) tehetünk 25 pixel távolságot (padding-top) és aláhúzhatjuk (text-decoration) a szöveget:

```
h2 { text-decoration: underline; padding-top: 25px; }
```

A harmadik szelektor lehetőség, hogy egy konkrét elemre hivatkozunk az id attribútumán keresztül. Ehhez a HTML kódban egyedi értéket kell adnunk az id attribútumnak, és ezt kell megadnunk a CSS-ben is egy # jel után. Az alábbi példában egy adott címsort kis kapitális stílussal jelenítünk meg:

```
<style type="text/css">
#idezetek { font-variant: small-caps; }
</style>
<h2 id="idezetek">Idézetek</h2>
```

Könnyen lehet, hogy nem az elem összes előfordulását, és nem is egy konkrét elemet akarunk formázni, hanem csak néhány kitüntetett helyen akarunk alkalmazni egy stílust. Ebben az esetben célszerű létrehoznunk egy saját stílus osztályt, aminek tetszőleges nevet adhatunk, a lényeg az, hogy pontot írjunk elé. Például definiálhatunk egy szerzo osztályt, ami meghatározza, hogy a szöveg legyen dőlt (font-style) és kisebb (font-size):

```
<style type="text/css">
.szerzo {
font-style: italic;
font-size: 80%;
}
</style>
```

Ezután azon a helyen, ahol használni akarjuk, a HTML elem class attribútumában tudunk rá hivatkozni:

```
<p class="szerzo">(Deák Ferenc)</p>
```

Így az oldalon több helyen is hivatkozhatunk erre a CSS osztályra, akár különböző HTML elemekben is.

> JÓTANÁCS: Célszerű olyan elnevezéseket választani a CSS osztályainkhoz, amik arra utalnak, hogy mire szolgál az adott osztály, nem pedig arra, hogy hogyan néz ki. Például a fontos dolgok hangsúlyozására szolgáló osztálynál a kiemeles jó elnevezés, a nagypiros viszont nem, még ha a lényeges részeket az oldalon nagyobb, piros betűvel jelenítjük is meg. Ha követjük ezt a bevált gyakorlatot, akkor nagyon könnyű lesz a dizájnon módosítani, például a kiemeles osztályt bátran átszínezhetjük kékre, a neve továbbra is arra utal, hogy a fontos részek megjelenítésénél használjuk. A nagypiros osztályban is átírhatjuk a színezést más színre, működni fog, csak éppen nem fogjuk később érteni, hogy miért hívjuk nagypirosnak, ha kéken jelenik meg, azaz nehéz lesz a kódunkat karbantartani.

Ez a négy (*, elem, id, class) a leggyakrabban használt szelektor típus. Ezen kívül vannak még további speciális szelektorok, melyekre ez a gyakorlatanyag már nem tér ki.

3.4. Span és div

Mi van akkor, ha nincs is azon helyen semmilyen HTML elem, ahol formázást szeretnénk alkalmazni? Ebben a kódrészletben például szerepel a nemkívánatos (elavult, nem javasolt a használata, de a öngészők megértik) font elem a lényeg pirossal történő kiemelésére:

```
Ez itt a <font color="red">lényeges</font> rész.
```

Ha az ajánlásoknak megfelelően mellőzzük a font elemet, akkor marad a csupasz szöveg, de vajon milyen szelektorral tudunk egyetlen szóra hivatkozni?

Sajnos semmilyennel, mindenképp szükségünk van egy HTML elemre. Ha ilyen problémával találkozunk, hogy az oldalon belül egy kisebb részt szeretnénk megformázni, akkor beszúrhatunk egy span elemet, és arra alkalmazhatjuk a stílust a korábban már megismert módokon:

```
<style type="text/css">
.fontos {
color: red;
}
</style>
Ez itt egy <span class="fontos">lényeges</span> rész.
```

Ha az oldalnak egy nagyobb részét (például fejléc, jobb hasáb, friss hírek blokk stb.) szeretnénk formázni, akkor hasonlóan használhatjuk a div (division) elemet. Mindkét elemnek van style, class és id attribútuma, a formázás tehát hasonlóan megy mindkét esetben.

A nagy különbség, hogy a span ún. inline elem, a div pedig blokk elem. Ez első megközelítésben annyit jelent, hogy a span nem változtat az oldal elrendezésén, a div után viszont alapértelmezés szerint bekerül egy sortörés, ezzel is jelezve, hogy ott egy újabb nagyobb blokk következik. Ennél azonban fontosabb, hogy a span csak további inline elemeket tartalmazhat (pl. strong, em, abbr, de div vagy p nem), míg a div szinte bármilyen HTML elemnek lehet a szülő eleme, így akár bekezdéseket vagy további div és span elemeket is tartalmazhat. Spant majdnem bármilyen másik elembe tehetünk, a div azonban szigorúbb, például spanbe vagy p elembe nem ágyazhatjuk.

3.5. Öröklés

A CSS egyik legérdekesebb és leghasznosabb tulajdonsága, hogy a beállítások öröklődnek. Ennek köszönhető, hogy nem szükséges minden beállítást minden elemre definiálnunk, hanem elég a legkülső elemre megadnunk, onnan öröklődni fog a gyermek elemekre.

Lehet például egy bekezdésünk egy kiemelt résszel:

```
<p>
Jártál-e már <em>Makkoshotykán</em>?
</p>
```

Ha a bekezdést kékkel írjuk, automatikusan a kiemelés is kék lesz, nem szükséges arra külön megadnunk ugyanezt a stílust:

```
p { color: blue; }
```

Szerencsére nem minden beállítás öröklődik, hanem csak azok, amelyeknek tipikusan kívánatos az öröklődése. A keretezés (border) például nem öröklődik. Ha öröklődne, akkor a bekezdésre alkalmazott kereten belül megjelenne egy külön keret Makkoshotyka körül is.

Ha szeretnénk kikényszeríteni az öröklést, akkor az inherit értéket adhatjuk meg az attribútumnak:

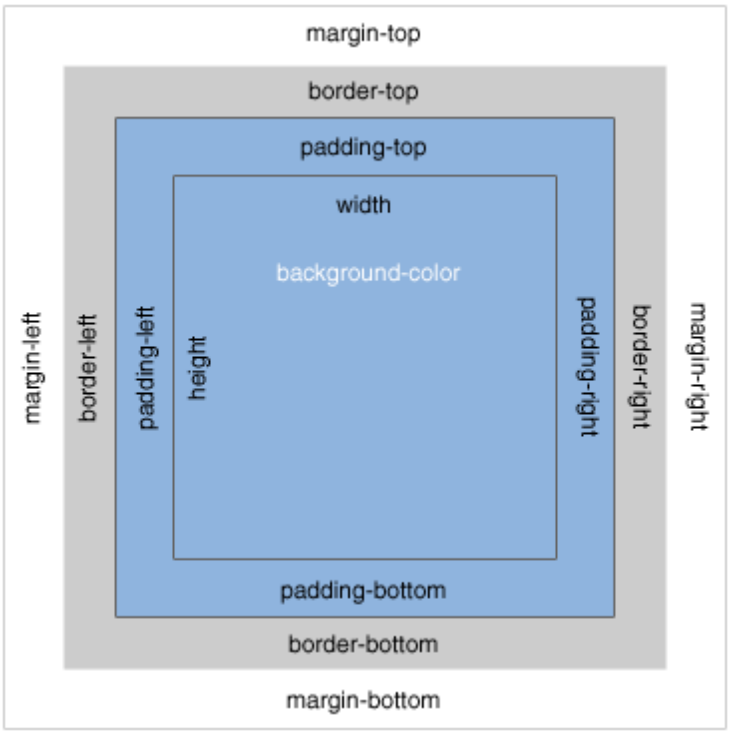
```
border: inherit;
```

A CSS-ben általában igaz az, hogy a specifikusabb beállítások fontosabbak, mint az általánosabb beállítások, ezért előfordulhat, hogy a szülőtől öröklődő értéket egy gyermek elemre megadott stílus felülírja. Ha ezt nem szeretnénk, használhatjuk az !important kiegészítést:

```
em { color: red; }
p { color: blue !important; }
```

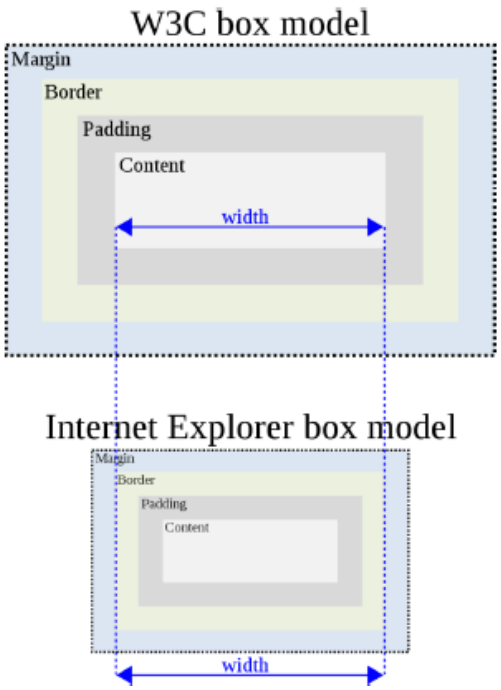
3.6. Doboz modell

Az elemek méretének meghatározásához és helyes pozicionálásához érdemes megismerkednünk a CSS doboz modellel (CSS box model). Az alábbi ábra szemlélteti, hogy egy adott HTML elem (leggyakrabban div) esetén pontosan hol is állítjuk a belső margó (padding), keret (border) és külső margó (margin) értékeket:



Bár a doboz modell logikusnak és egyértelműnek látszik, történeti okokból sajnos a böngésző gyártók nem egyformán valósították meg, emiatt könnyen előfordulhat, hogy ugyanazt az oldalt különböző böngészőkben megtekintve pár pixel különbséget tapasztalunk. Ez többnyire nem okoz gondot, azonban ha nagyon kicentizzük az oldalt és mindent pixelre kikalkulálunk, akkor lehet, hogy kellemetlen élményben lesz részünk, például az egyik böngészőben egymás mellé kerül két elem, míg egy másik böngészőben egymás alá tördelődnek, mert ott éppen nem férnek ki egy sorba.

Az alábbi ábra például a W3C szabványnak megfelelő és az Internet Explorer régebbi (IE9-nél korábbi) változatai által alkalmazott doboz modell közötti különbséget szemlélteti:



JÓTANÁCS: Bár a szabványok és a böngészők egyre jobban közelednek egymáshoz, egyelőre sajnos nem a doboz modell a CSS egyetlen böngészőfüggő része (lásd QuirksMode.org). A váratlan problémák elkerülése érdekében ezért célszerű az oldalunkat minden böngészőben ellenőrizni, amit a webhelyen támogatni szeretnénk. Az oldal megjelenésének teszteléséhez jól használható az ingyenesen letölthető IE Tester és az online szolgáltatásként elérhető BrowserShots.

3.7. Oldalelrendezések

A webhely arculatának kialakításakor az egyik első feladatunk az oldalak elrendezésének kialakítása lesz. Ki kell találnunk, hogy milyen széles és milyen magas lesz az oldal, lesz-e fejléc, lábléc, esetleg menü – és persze azt is, hogy mindez hol és mekkora méretben fog megjelenni.

Hagyományosan erre létezik egy nagyon egyszerű megoldás, a táblázat alapú oldalelrendezés (table-based layout). Ennek a lényege, hogy a HTML nyelv által biztosított táblázat elemekkel (table, td, tr) valójában az egész oldalt egy nagy, láthatatlan keretekkel megrajzolt táblázatként képzeljük el. Például a bal hasáb a táblázat első oszlopa, a jobb hasáb a táblázat utolsó oszlopa, a felső menü pedig egy nagy cella, összevonva a táblázat összes oszlopát. Ha a nagy külső táblázat valamelyik celláját tovább kell osztanunk, akkor abban újabb táblázatot helyezünk el, ahol a sorokat és oszlopokat az igényeknek megfelelően vonjuk össze. A végeredmény egy gigantikus táblázat rengeteg, amiből szerencsére a végfelhasználó semmit nem vesz észre.

A táblázatos elrendezésnek a „súlyosbított” fajtája az ún. spacer GIF-ek alkalmazása. Ennél a megoldásnál az oldal egyes elemeinek pozicionálását átlátszó (azaz rejtett) képfájlok tetszőleges átméretezésével oldják meg, amelyek így épp a kívánatos mértékben tolják szét az oldalon lévő rejtett táblázatot.

Ennek a megközelítésnek azonban számos hátránya van:

1. A HTML nyelv táblázattal kapcsolatos elemeit táblázatos adatok megjelenítésére találták ki. Egy táblázatnak lehet fejléc sora, lábléc sora és benne az egyes cellákban adatok szoktak megjelenni, és ezeket az adatokat lehet soronként vagy oszloponként olvasni. Egy szó mint

száz, a table elem arra való, hogy a felhasználónak egy a hagyományos értelemben vett táblázatot jelenítsünk meg.

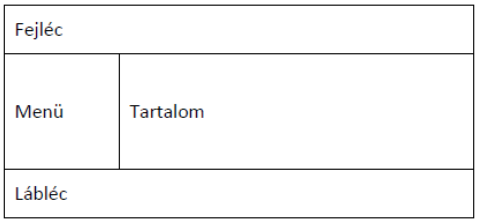
2. A table elem szemantikai jelentését bizonyos programok kihasználják, például a látássérültek által használt képernyőolvasók vagy a keresőmotorok. Ezek az alkalmazások speciálisan értelmezik a táblázatot, ezzel segítve a felhasználót.
3. A táblázat alapú oldalelrendezés nagyon sok HTML kódot eredményez, mely nemcsak az oldal letöltését és betöltését lassítja, hanem ráadásul nagyon nehezen átlátható, azaz nehezen tudunk később belejavítani.

Ezek a megoldások a maguk idejében sem voltak ideálisak, ma már kifejezetten idejétmúltak, sőt nem túlzás azt állítani, hogy napjainkban igen ciki ezeket használni pozicionálásra.

3.7.1. Táblázatmentes oldalelrendezés

A korrekt megoldás a táblázatmentes oldalelrendezés (tableless layout) alkalmazása. A táblázatmentes elrendezésnél az alapelv, hogy mindent stílusbeállításokkal próbálunk a megfelelő helyen és a megfelelő formában megjeleníteni. Ez a legtöbbször azt jelenti, hogy az oldalon létrehozunk számos div elemet (pl. fejléc, menü, kereső doboz, lábléc stb.) és ezeket CSS-sel méretezzük és pozicionáljuk.

A következő példában az egyik leggyakoribb oldalelrendezést készítjük el táblázatok nélkül. A cél egy olyan oldal, aminek van egy 50 pixel magas fejléce, egy 20 pixel magas lábléce, a kettő közötti rész pedig függőlegesen 30%-70% arányban osztott, és mindez összesen 950 pixel szélességben középre rendezve jelenik meg az oldalon, valahogy így:



Mivel az oldal egyes részei önmagukban is összetettek lehetnek (képek, bekezdések, felsorolások stb.), ezért minden egyes részt önálló div elembe zárunk, amiket pedig az id attribútumon keresztül egyedi névvel látunk el:

```
<div id="keret">
<div id="fejlec">Fejléc helye</div>
<div id="balmenu">Bal menü helye</div>
<div id="tartalom">Tartalom helye</div>
<div id="lablec">Lábléc helye</div>
</div>
```

A CSS fájlban ezekre az egyedi azonosítókra hivatkozva állítjuk be az oldal egyes részeinek megjelenését. Az egész oldal legyen 950 pixel széles és középre rendezett:

```
#keret{
width: 950px;
margin-left: auto;
margin-right: auto;
}
```

Majd jöjjön az 50 pixel magas fejléc:

```
#fejlec{
height: 50px;
}
```

A következő két div a balmenu és a tartalom egymás mellett:

```
#balmenu{
width: 30%;
float: left;
}
#tartalom{
width: 70%;
float: right;
}
```

Végül pedig a lábléc, ami a teljes szélességet kitölti és 20 pixel magas:

```
#lablec{
height: 20px;
clear: both;
}
```

Ezzel készen is vagyunk. Látható, hogy a HTML kód nagyon egyszerű maradt, és az azonosítóknak köszönhetően jól olvasható, hogy az egyes blokkok az oldal melyik részét jelentik. A CSS kód sem bonyolult, ráadásul nagyon rugalmas, hiszen könnyen vehetjük magasabbra a fejlécet vagy éppen szélesebbre a menüt.

4. Ellenőrző kérdések

- Mire szolgál a HTML?
- Hogyan lehet egy dokumentumban sortörésekt megadni?
- Hogyan lehet egy dokumentumban címsorokat megadni?
- Hogyan lehet egy dokumentumba képeket ágyazni?
- Hogyan lehet egy dokumentumban 2x3 méretű táblázatot megadni?
- Mire szolgál a CSS?
- Milyen lehetőségeket ismer egy böngészőben megjelenítendő szöveg formázására?
- Mi a probléma a táblázatokra épülő oldalelrendezésekkel?
- Mire szolgál a div elem?
- Mire szolgálnak a CSS szelektorok?
- Hogyan lehet CSS-ben előírni, hogy minden főcím piros színnel jelenjen meg?