Nagy Gusztáv

Web programozás I.

0.7. verzió

2008. szeptember

Jogi nyilatkozat

Bevezetés

Felhasználás

Ezzel a jegyzettel arra vállalkozok, hogy a Kecskeméti Főiskola GAMF Karán tanuló műszaki informatikus hallgatók „kezébe” olyan írásos anyagot adjak, amely az előadások és gyakorlatok mellett további segítséget ad a Web kliens és szerver oldali nyelvei, alapvetőbb technológiái alapszintű megismerésére.

A jegyzet használatához nem nélkülözhetetlen, de erősen javasolt az előadások látogatása. A jegyzet alapvetően a tanórák menetéhez kapcsolódó lineáris feldolgozásra készült, de a fejezetek egy része nem épít a közvetlen megelőző fejezetekre.

Az egyes fejezetek tananyagának elsajátításához az elméleti rész átolvasása után az ellenőrző kérdések alapos átgondolását, valamint a gyakorló feladatok megoldása javasolt. E nélkül a tantárgy teljesítése a hallgatók többsége számára nem lesz eredményes.

A jegyzet feltételezi az alapvető programozási és hálózati alapismeretek meglétét. Ennek hiányában az anyag elsajátítására több időt kell fordítani.

0.7. verzió

A jegyzet jelenlegi verziója – szokás szerint – nem készült el a félévkezdésre olyan precizitással[1], mint szerettem volna. Ezért a félév során kisebb kiegészítések, módosítások, a következő félév kezdetekor pedig újabb verzió lesz várható.

Köszönet

A jegyzet elkészítéséhez elsősorban a W3Schools[2] oktatóanyagait használtam fel. Az anyagok fordításában résztvevő hallgatókat szintén köszönet illeti.

Végül szeretnék köszönetet mondani a Weblabor[3] szakmai közösségének, akiktől a legtöbbet tanultam – többek között – szemléletmódban.

Kecskemét, 2008. szeptember

a szerző

[1] Például az ábrák egységes formázása és számozása még várat magára ☹

[2] http://www.w3schools.com/

[3] http://weblabor.hu/

Tartalomjegyzék

1. Fejlesztőkörnyezet kialakítása

Mielőtt a web nyelveinek, technikáinak részleteiben elvesznénk, érdemes egy bevezető fejezetet szentelni a hasznos (és többnyire szükséges) előismeretek áttekintésének.

Általános esetben a szerver környezet kialakítása a rendszergazda feladatköréhez tartozik, mégis fontos, hogy alapszinten átlássuk a feladatokat, lehetőségeket.

1.1. Szerver operációs rendszer

Linux

Az elterjedtebb, és nem kimondottan asztali (*desktop*) használatra szánt Linux[4] disztribúciók telepítésével alapból egy működő web-, és adatbázis szervert kapunk. Akár régebbi, más felhasználások számára értéktelen vasra is telepíthetünk Linuxot, egy kisebb forgalmú honlapot tökéletesen képes kiszolgálni.

A nagy nevű disztribúciók friss verziói a mai csúcsgépek meghajtására és csúcs igények kiszolgálására alkalmas, szintén könnyen telepíthető lehetőséget nyújtanak a rendszergazdáknak.

Önálló szerver üzemeltetése nélkül, web-hosting szolgáltatás[5] igénybe vételével is többnyire Linux alapú szerverekkel találkozhatunk.

BSD

Kevésbé közismert, de egyes rendszergazdák körében stabilitása miatt nagy népszerűségnek örvendő Unix[6] operációs rendszerek. Az alapvető változatok szerver feladatokra optimalizáltak, így nagyszerűen alkalmasak a webes kérések kiszolgálásához.

Windows

Bár éles webes környezetben nem jelentős a részesedése, azért előfordul az alkalmazása. A tanulás szakaszában azonban sokszor a legkézenfekvőbb megoldás az egyetlen (többnyire Windows operációs rendszerrel működő) számítógépünket szerverré alakítani. A fejezet további részében ezzel a témával fogunk foglalkozni.

Megjegyzés: Ha a fejlesztéshez Windows operációs rendszert alkalmazunk, akkor érdemes néhány technikai dologra figyelni.

Először is az állománynevekben a Windows nem tesz különbséget kis-, és nagybetű között, ezért ha pl. HTML-ben vagy CSS-ben nem vagyunk következetesek, akkor Windows alatt még működni fog az oldal, de ha a kész munkát más operációs rendszert futtató gépre kell átvinni, akkor nagy bajban leszünk. Javasolható, hogy webes fájl esetén nagybetű soha ne szerepeljen a fájlnévben.

A Total Commander[7] az ilyen jellegű hibák elkerülése érdekében lehetővé teszi az FTP-ver átvitt állományaink kisbetűsítését is.

Hasonló okok miatt nem célszerű az állománynevekben ékezetes betűket vagy egyéb speciális karaktereket alkalmazni.

---

[4] http://www.linux.hu/

[5] http://tarhely.lap.hu/

[6] http://www.bsd.hu/

[7]http://www.totalcommander.hu/

Végül szintén fontos, hogy a könyvtárnevek megadásánál a \ helyett mindig a / jelet használjuk, és soha ne adjunk meg Windows alatt érvényes teljes elérési utat, pl. *<img src="C:\www\kep.gif">*.

1.2. Szerveralkalmazások

1.2.1 XAMPP integrált telepítő csomag

Mivel a szerver alkalmazások telepítése nem mindig egyszerű feladat, próbálkozhatunk előre csomagolt, és minden szükséges alkalmazást telepítő és bekonfiguráló programokkal is. Ezek közül csak egyet nézünk meg közelebbről, a többi alkalmazása hasonló. A szolgáltatások körében lehetnek jelentősebb eltérések is.

A telepítő-csomagok hátránya, hogy bár többnyire működnek, egy esetleges hiba előállása esetén a hiba megszüntetése eléggé bajos lehet, mivel kevésbé tudunk a csomag működésébe belelátni.

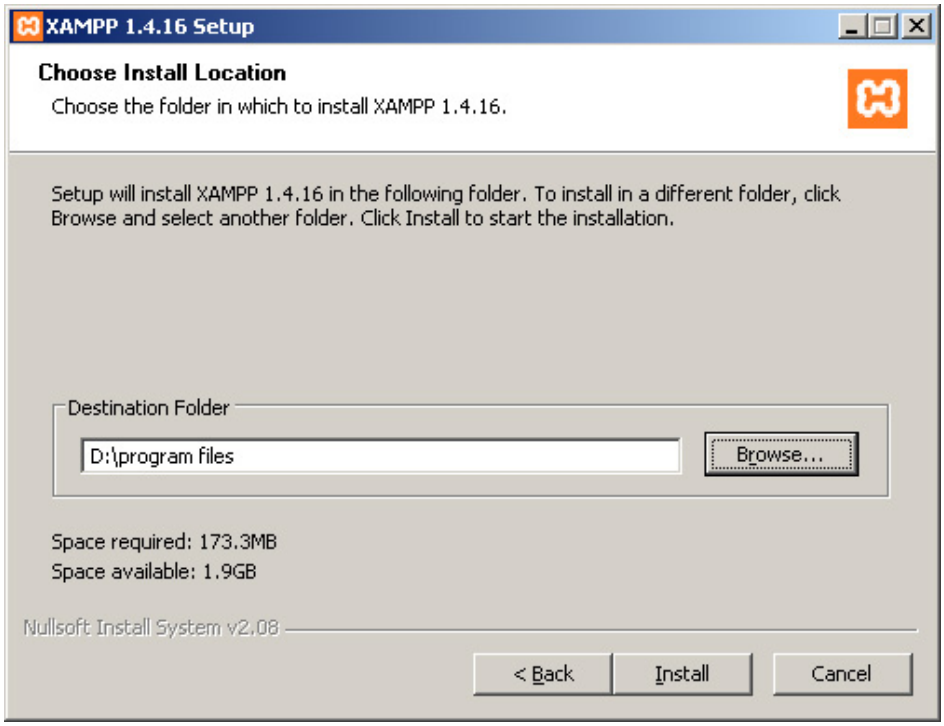
A szerző által leginkább ajánlott integrált tepepítő csomag az **XAMPP**[8]. Az XAMP for Windows 1.6.5 változata[9] a következő szoftvereket telepíti és konfigurálja:

- Apache HTTPD 2.2.6 + Openssl 0.9.8g
- MySQL 5.0.51
- PHP 5.2.5
- PHP 4.4.7
- phpMyAdmin 2.11.3
- FileZilla FTP Server 0.9.24
- Mercury Mail Transport System 4.52

A letöltött telepítőprogram lényegében a szokásos kérdéseket tesz fel, legfontosabb a telepítés helye:

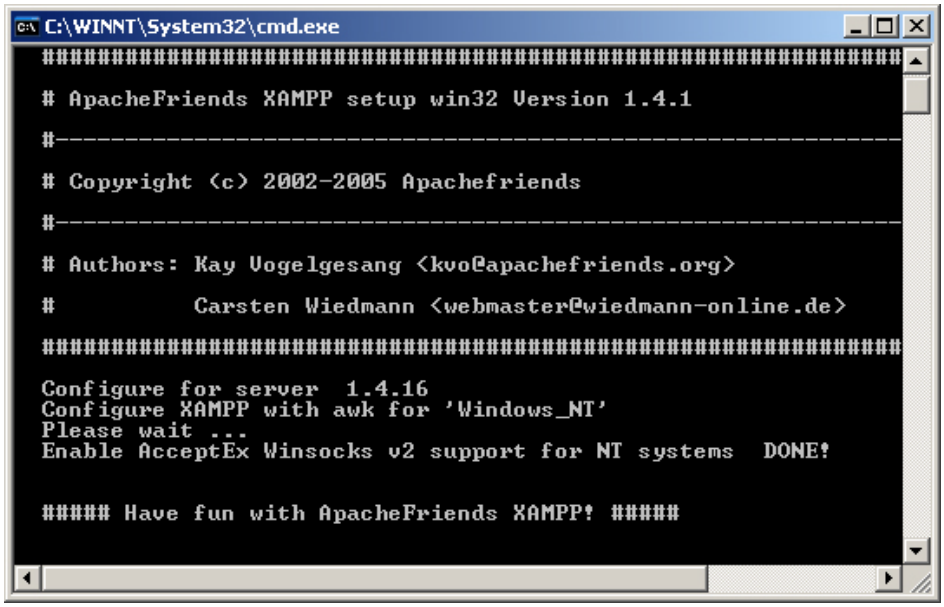
[8] http://www.apachefriends.org/

[9] http://www.apachefriends.org/en/xampp-windows.html



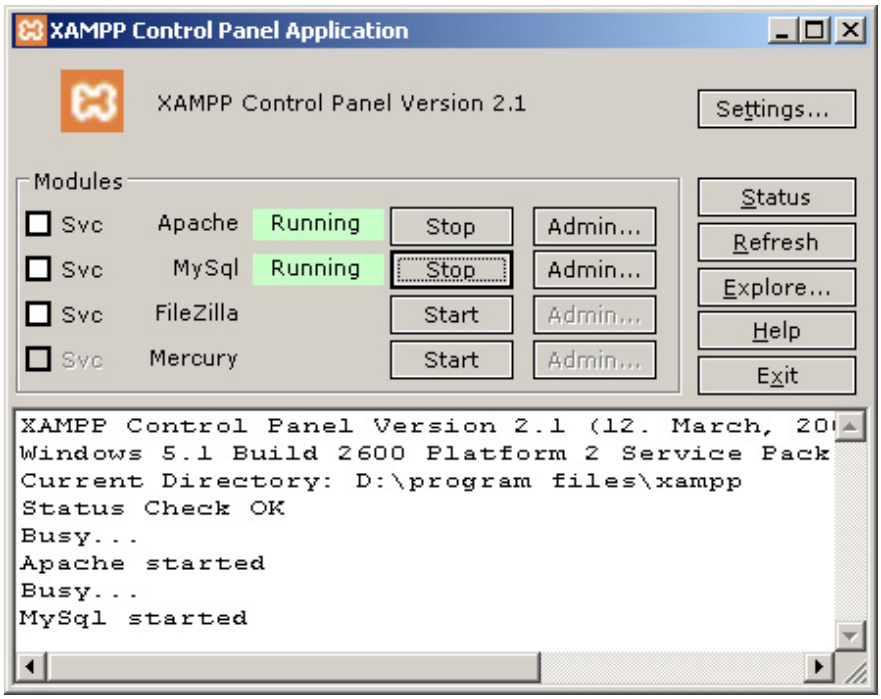
1.1. ábra: Az XAMPP szokásos telepítése

A telepítés végezhető parancssorból is a *setup-xampp.bat* parancsállomány futtatásával:



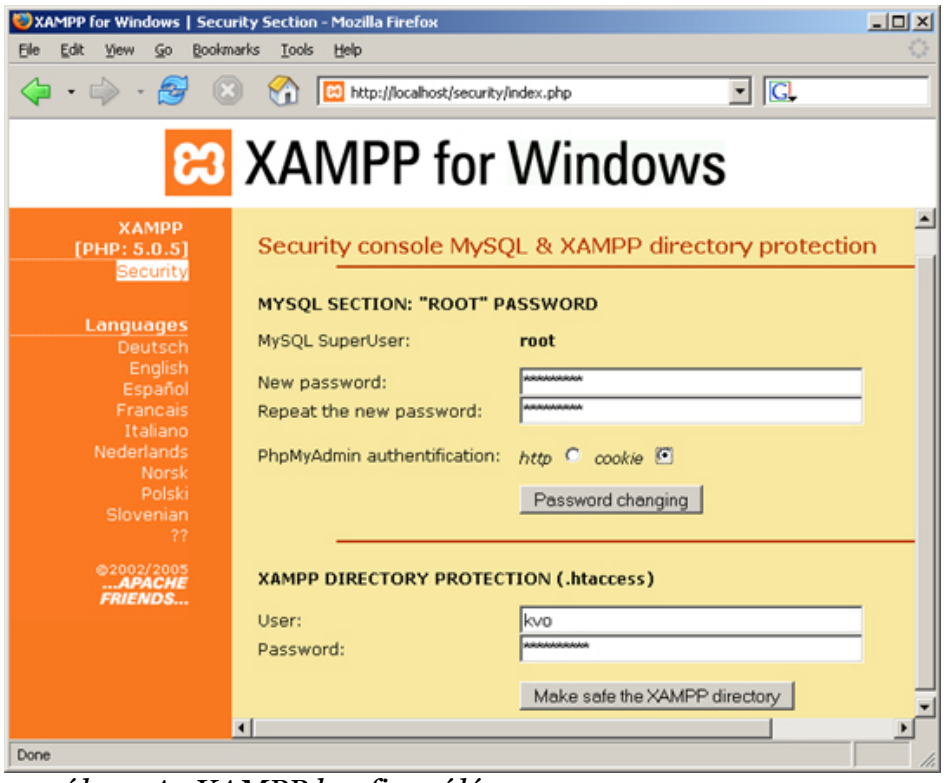
1.2. ábra: Az XAMPP parancssori telepítése

A telepítés után a Start menüből és parancssorból is vezérelhetjük az alkalmazásokat, de legegyszerűbb az XAMPP Control Panel alkalmazása:



1.3. ábra: XAMPP Control Panel

A telepítés után a feltelepült rendszer kipróbálása és a jelszavak megadása célszerű a *Security* oldalon:



1.4. ábra: Az XAMPP konfigurálása

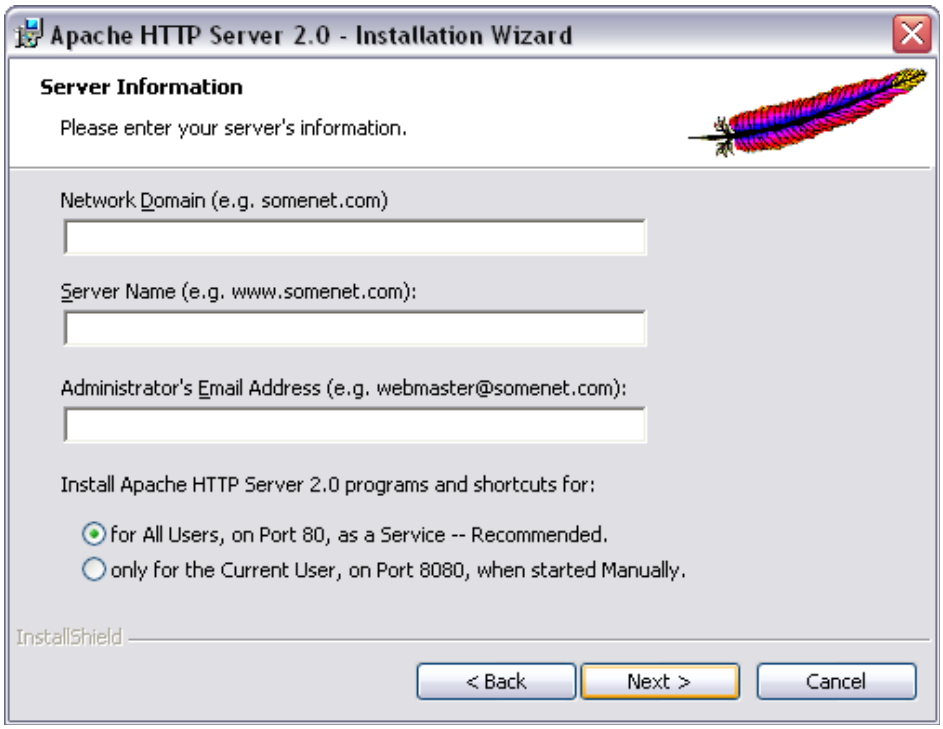
Ha az XAMPP, vagy más hasonló komplex programcsomag alkalmazása mellett döntünk, akkor a fejezet hátralevő részében bemutatott önálló alkalmazástelepítéseket már nem kell elvégeznünk, a rendszerünk kész a webfejlesztés tanulására.

1.2.2 A szerverek önálló telepítése

Ha a szükséges szerver alkalmazásokat külön-külön szeretnénk megtenni, akkor a következő sorrendben érdemes azokat megtenni.[10]

Az Apache telepítése

Az Apache webszerver letöltési oldaláról[11] töltsük le a megfelelő telepítő állományt. A telepítő varázsló ablakai közül a következőket érdemes kiemelni:



1.5. ábra: Apache telepítő

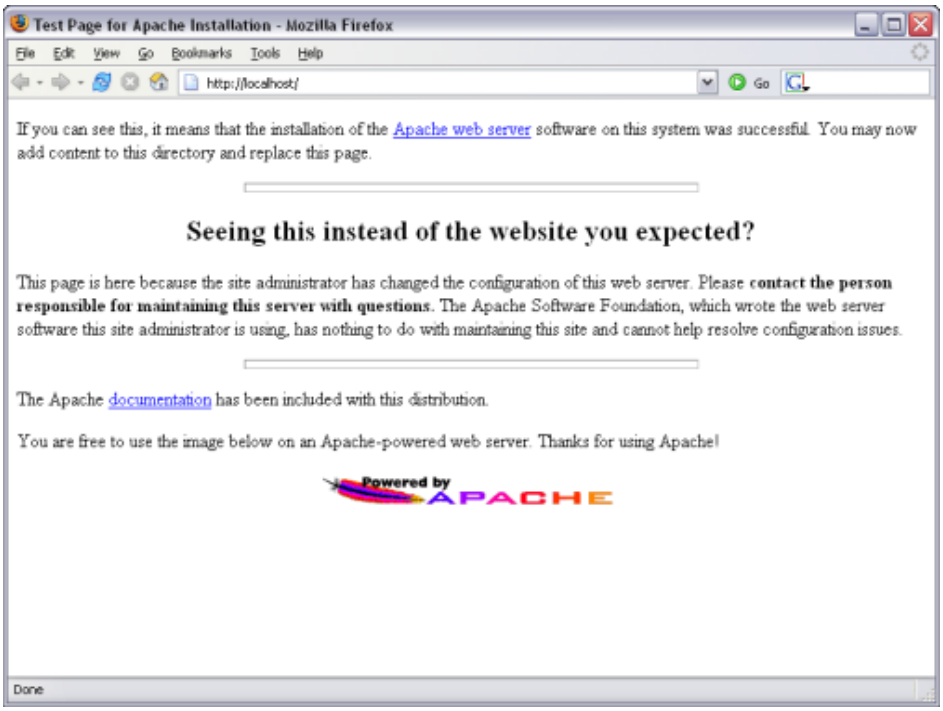
Saját gépre telepítés esetén kétszer *localhost* megadása javasolt, az e-mail címnek ebben az esetben nincs jelentősége. Érdemes szolgáltatásként (*service*) telepíteni a szervert, ugyanis ellenkező esetben csak manuálisan lehet indítani, és egy konzol ablak állandóan a tálcánkon fog csücsülni, stb. Természetesen szolgáltatásként telepítve is van lehetőség a szerver manuális menedzselésére, tehát ettől még nem kell a szervernek állandóan futnia.

Telepítés után a gép biztonsági beállításait, szoftvereit (elsősorban a tűzfalat) valószínűleg konfigurálni kell a sikeres használat érdekében.

A kipróbálás a böngészőbe beírt *localhost* címmel történhet. Sikeres telepítés esetén hasonlót kell látnunk:

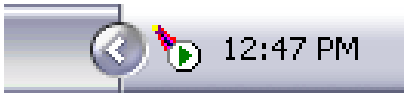
[10] További források:
http://weblabor.hu/cikkek/apachephptelepites, http://tanarurkerem.hu/node/15 (videó)

[11] http://httpd.apache.org/download.cgi



1.6. ábra: Sikeres Apache telepítés után

Az óra mellett megjelenik az Apache Monitor:



1.7. ábra: Apache Monitor

A Monitor ikonnal egyszerűen tudjuk indítani, leállítani, vezérelni a szervert. A képen látható zöld háromszög (mint a szokásos *Play* gomb) jelzi a szerver futását is.

A konfigurálással kapcsolatban meg kell még említeni a *httpd.conf* állományt, amelyben a szerver konfigurációja található. Az alapkonfiguráció kezdetben tökéletesen megfelel, esetleg a *DocumentRoot* beállítása lehet szükséges, ha máshova szeretnénk a webre szánt állományainkat helyezni.

Megjegyzés: A konfigurációs állományban az operációs rendszertől függetlenül mindig a / jelet kell használni elérési utak megadásához.

Megjegyzés: A konfigurációs állományt a szerver csak az indulásakor veszi figyelembe, így annak változása esetén a szervert újra kell indítani, pl. az Apache Monitor segédprogrammal.

A PHP telepítése

A PHP telepítése[12] is a kívánt telepítőkészlet letöltésével kezdődik. A letöltési oldalon[13] választhatunk, hogy a telepítő varázslót (*installer*) vesszük igénybe, vagy pedig a kézi telepítést választjuk (*zip package*).

Varázsló használata esetén kicsit kevesebb dolgunk lesz, ezért talán ezt érdemes választani. A telepítési folyamat során többnyire az alapértelmezett beállítások elfogadása ele-

[12] További források: http://weblabor.hu/cikkek/apachephptelepites, http://hu.php.net/manual/hu/install.windows.php

[13] http://www.php.net/downloads.php

gendő. Arra érdemes figyelni, hogy *Standard* helyett *Advanced* telepítést válasszunk, hogy legyen lehetőségünk néhány beállítás megadására.

Az említésre érdemes kérdések közé tartozik a hiba megjelenítés (*error reporting*) szintje, ahol a tanulás idejére mindenképpen a legtöbb támogatást nyújtó alapértelmezett beállítást érdemes választani. Kiválaszthatjuk még, hogy milyen webszervert használunk, és mi legyen az alapértelmezett kiterjesztése a PHP állományainknak. Itt is javasolt az alapértelmezett *.php* elfogadása. A feltett kérdések alapján a *php.ini* konfigurációs állomány is elkészül.

Apache konfiguráció ismét

Rossz hír, hogy az Apache még nem tud a PHP-nkről, ezért még vissza kell nyúlnunk a *httpd.conf* állományhoz.

Ajánlott modulként[14] telepíteni a PHP-t. Ehhez először is a megfelelő *dll* állomány használatát elő kell írnunk (az elérési utakat a PHP telepítése alapján kell megadni):

```
LoadModule php5_module "c:/php/php5apache2.dll"
```

A *php* kiterjesztéssel rendelkező állományokat a PHP értelmezőnek át kell adni, és csak annak kimenetét visszaadni a felhasználónak. Ezt a következő sorral tudjuk előírni:

```
AddType application/x-httpd-php .php
```

Szükséges a *php.ini* helyét is megadnunk:

```
PHPIniDir "C:/php"
```

Végül a *DirectoryIndex* felsorolásába érdemes első helyre tenni az *index.php* nevet, hogy ha az elérési út nem tartalmaz állománynevet, akkor az *index.php*-t próbálja az Apache betölteni elsőként.

Megjegyzés: Kérhetjük azt is az Apache szerverünktől, hogy a *.html* állományokat is dolgozza fel, de ezt ritkán alkalmazzuk.

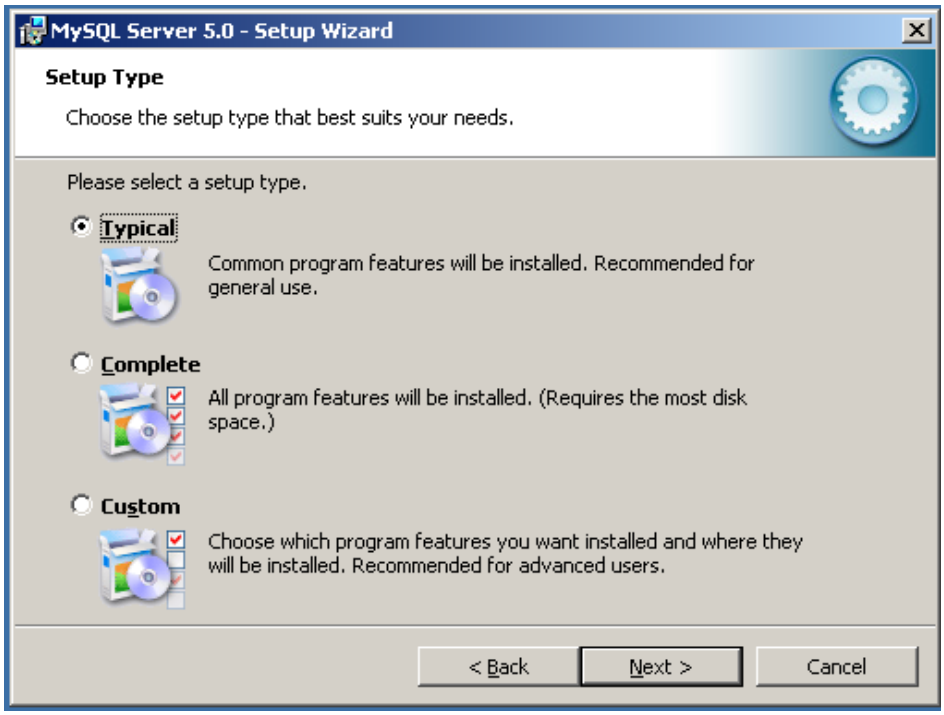
A MySQL telepítése

A MySQL telepítését is a telepítőcsomag letöltésével kell kezdenünk. Itt most további magyarázatok nélkül a MySQL Community Server letöltését[15] javasoljuk.

Első lépésként a tipikus (Typical) telepítést válasszuk:

[14] A modul használata itt a tanulás idejére ajánlható. Éles környezet esetén érdemes megfontolni a jóval biztonságosabb FastCGI módban való futtatást.
További információ: http://weblabor.hu/cikkek/phpfastcgimodban

[15] http://dev.mysql.com/downloads/mysql/5.0.html#downloads



1.8. ábra: Tipikus telepítés javasolt

Megjegyzés: Terjedelmi okokból a MySQL konfigurálásáról csak egy korábbi, de alapokat nagyon jól tisztázó cikket[16] tudunk ajánlani Grac Róbert tollából.

1.3. Fejlesztőeszközök

Fejlesztőeszközök témájában érdemes először a szélsőséges példákat áttekinteni.

Az eszközök közül a Jegyzettömb az a minimum, amivel még többé-kevésbé lehet webfejlesztést végezni, bár több okból sem célszerű. Egy jobb szerkesztő (például NotePad++[17]) viszont már igen jól használható.

A másik végletet azok a WYSIWYG (What *You See Is What You Get*) programok képviselik, amelyek úgy teszik lehetővé az oldal elkészítését, hogy (első ránézésre) semmilyen programozói, webes tudásra nincs szükség. Ebbe a kategóriába tartozik például a népszerű FrontPage. Ezen eszközöknek az előnyük a hátrányuk: bár adnak lehetőséget a kódszintű szerkesztésre, mégis erősen korlátozzák azt.

A szerző véleménye szerint a webfejlesztőnek olyan eszközökre van szüksége, amelyek úgy adnak támogatást, hogy a folyamatot a fejlesztő, és nem a program vezérli. A programozói szerkesztőprogramok e középső kategóriájának két alapvető szolgáltatása a kódszínezés és a kódkiegészítés. Ingyenes szoftverek közül kevés tud kódkiegészítést, azt is általában csak 1-2 nyelvre.

1.3.1 Context

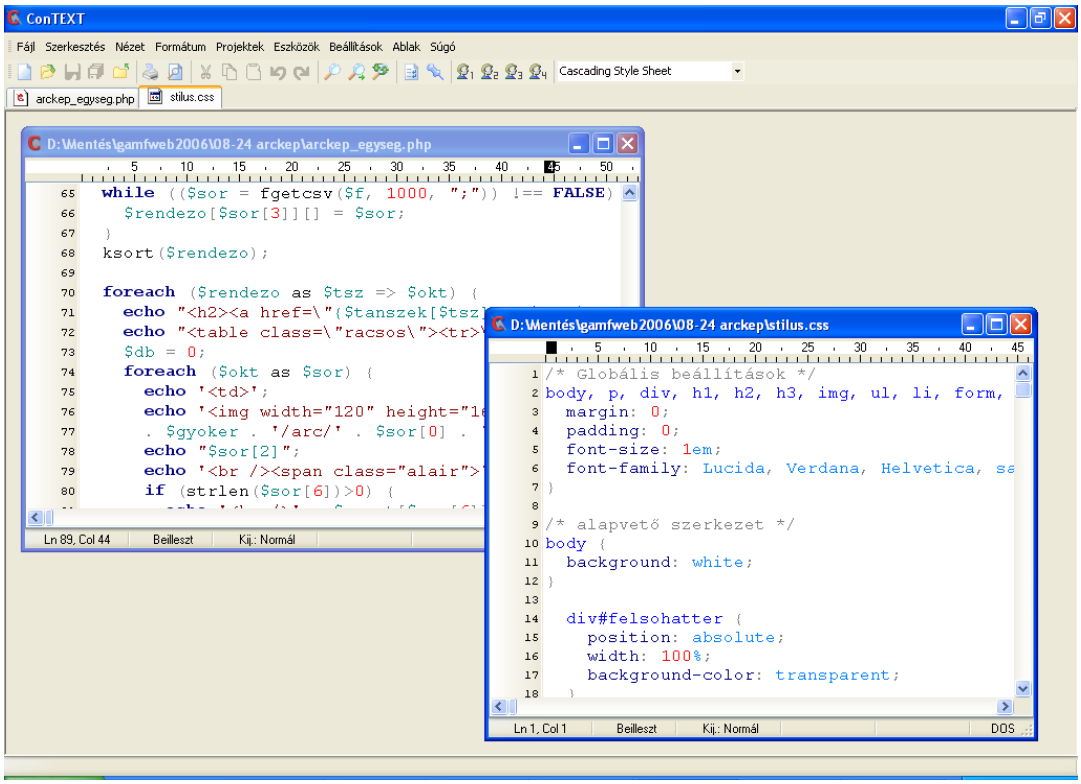
A kódszínezést nyújtó programok közül a szerző egyik kedvence a Context[18], amely – többek között – magyar felülettel több, mint 200 nyelv színezését tudja megoldani. Ezen

[16] http://weblabor.hu/cikkek/mysql41telepites

[17] http://notepad-plus.sourceforge.net/

[18] http://www.context.cx/

kívül rendkívül hasznos szolgáltatása a könnyen konfigurálható sablonbeillesztés. (Pl. egy üres HTML állomány esetén 4-5 billentyűleütéssel egy 5-6 soros minimális HTML oldalt hozhatunk létre.



1.9. ábra: A Context kódkiemelése

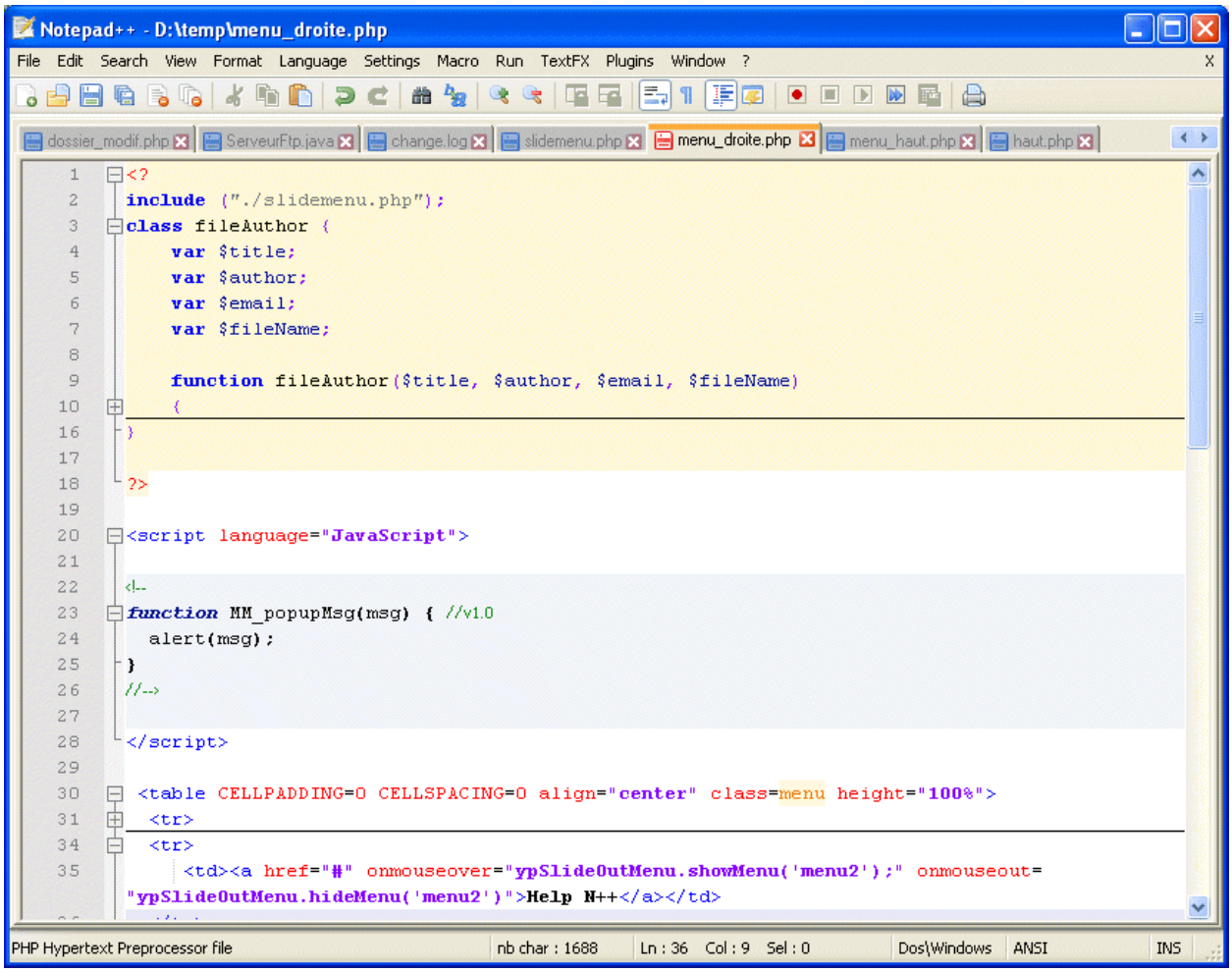
1.3.2 Notepad++

Notepad++ egy ingyenes forráskód-szerkesztő (és a hagyományos Jegyzettömb leváltására hivatott program), amely számos programozási nyelvet támogat a MS Windows környezetben.

A Notepad++ képességei

- Kódkiemelés és kódblokkok egységbe zárása
- 38 támogatott programnyelv
- WYSIWYG nyomtatás
- Felhasználó által állítható kódkiemelés
- Automatikus kiegészítés
- Több dokumentum megnyitása
- Többszörös nézet
- Változtatható nézetek
- Fájl állapotváltozásának automatikus figyelése
- Nagyítás és kicsinyítés
- Több-nyelvi környezet támogatása

- Könyvjelzők
- Összetartozó nyitó- és záró zárójelek színezése
- Makrók felvétele és lejátszása



1.10. ábra: A Notepad++ kódkiemelése

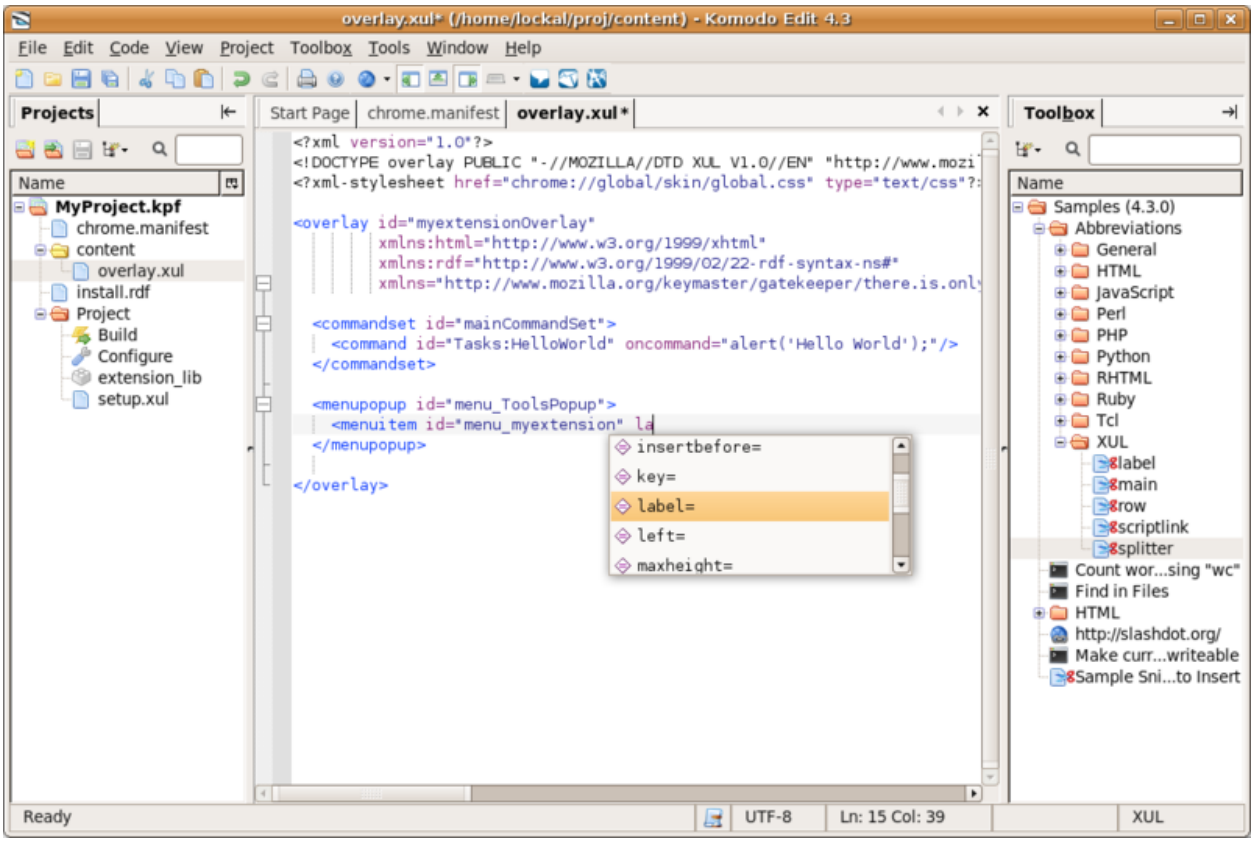
1.3.3 Komodo Edit

Amióta az ActiveState a Komodo Edit[19] integrált fejlesztőkörnyezetét ingyenessé tette, igazán profi eszközt használhatunk.

Néhány szolgáltatás

- Projekt kezelés
- Távoli fájlok szerkesztése
- Intelligens kódkiegészítés

[19] http://www.openkomodo.com/



1.11. ábra: Komodo Edit kódkiegészítés közben

1.4. Grafikus programok

Mivel ez a jegyzet nem grafikusok vagy dizájnerek, hanem webfejlesztők számára készült, ezért itt csak néhány alapvető dologról lesz szó.

A fejlesztő a valós életben többnyire kép(ek) formájában kapja meg az oldal látványtervét, valamilyen szöveges formátumban a szöveges részt, és nyers formában a tartalomhoz kapcsolódó fényképeket. Ebben az esetben a fejlesztő feladata az, hogy a látványterv alapján elkészítse az oldal HTML és CSS kódját, és „ráhúzza” minderre a dizájnt, a szöveget és a fényképeket. A grafikus programok szempontjából ez azt jelenti, hogy a dizájnt alkotó képet téglalap alakú részekre kell vágni, a fényképeket pedig méretre hozni, javítani, színkorrekciót, világosítást stb. végrehajtani. E feladatok ellátására részben a Paint is megfelel, de érdemes valamivel komolyabb szoftvert alkalmazni.

Gyakorlatilag itt is ízlés, szokás kérdése, hogy ki melyik programot választja. Az alapvető szükséges szolgáltatások:

- Mentés GIF, JPG és PNG formátumban.
- Átméretezés
- Kivágás
- Korrekciók
- Szűrés

1.4.1 Paint.NET

A szerző sokáig a shareware Jasc (azóta Corel) Paint Shop Pro híve volt, de a közelmúltban áttért a teljesen ingyenes Paint.NET[20] programra.

1.12. ábra: Paint.NET

További alapszolgáltatásai:

- rétegek kezelése
- effektusok használata

[20] http://www.getpaint.net/

2. Web alapismeretek

Ebben a fejezetben néhány olyan témát érintünk, amelyekkel a webfejlesztés megkezdése előtt tisztában kell lennünk.

2.1. Alapfogalmak

Kezdjük néhány alapfogalommal a Wikipédia[21] alapján:

2.1.1 Webböngésző

Webböngészőnek vagy böngészőnek nevezzük azon programokat, melyekkel az interneten található tartalmakat – legtöbbször weblapokat – lehet megtekinteni, illetve az Interneten át elérhető szolgáltatásokat használni.

A webböngészők a webszerverekkel **HTTP** protokollon keresztül kommunikálnak. A HTTP segítségével a böngészők adatokat küldhetnek a webszervereknek, valamint weblapokat tölthetnek le róluk.

A lapokat a böngésző az **URL** segítségével találja meg, mely a lap címét jelöli. Az URL a címhez tartozó protokollal kezdődik, például a http: a HTTP protokoll jelölése. Sok böngésző több más protokollt is támogat, mint például az ftp: az FTP, a https: pedig a HTTPS protokollhoz.

A weblaphoz tartozó fájl formátuma többnyire **HTML**, ez a HTTP protokollban a MIME tartalomtípus alatt van megadva. A legtöbb böngésző a HTML mellett támogat más formátumokat is, mint amilyen a JPEG, PNG és GIF képtípusok, valamint tovább bővíthető bennük a támogatott fájltípusok listája különböző beépülők használatával. A HTTP tartalom és az URL használatával a webfejlesztők képeket, animációkat, mozgóképeket és hangokat ágyazhatnak be vagy tehetnek elérhetővé a weblapokon.

A HTML a böngészőkkel együtt fejlődött, a „hivatalos” HTML-változatokat a W3C hagyta jóvá illetve készítette el. A böngészők sokfélesége és a cégek saját HTML módosításai kompatibilitási problémákhoz vezettek. A modern webböngészők (mint a Mozilla, Opera, Safari stb.) már pontosabban támogatják a HTML és XHTML szabványokat (a HTML 4.01-gyel kezdődően), melyekkel a weblapok azonosan kell megjelenjenek minden ilyen böngészőben. Az Internet Explorer jelenleg még nem támogatja tökéletesen a HTML 4.01 és XHTML 1.x szabványokat. Jelenleg sok weboldal már valamilyen könnyen kezelhető, azonnali eredményt szolgáltató, úgynevezett WYSIWYG szerkesztővel készült, mint amilyen a Macromedia Dreamweaver vagy Microsoft Frontpage.

Böngészőmotor

A böngészőmotor olyan szoftver, amely a webes tartalmat (mint például HTML, XML, kép fájlok stb.) és a formázás információit (mint például CSS, XSL stb.) mint formázott tartalmat jeleníti meg a képernyőn. A végső cél általában a monitor vagy a nyomtató. Egy böngészőmotort tipikusan webböngészők, e-mail kliensek, vagy más olyan alkalmazások használnak, amelyeknek webes tartalmak megjelenítése (és szerkesztése) a feladatuk.

[21] http://hu.wikipedia.org/

A „böngészőmotor” kifejezés akkor lett széleskörűen használatos, amikor a Mozilla projekt elkészítette sajátját Gecko néven, mint egy a böngészőtől elválasztott komponenst. Más szóval, a Mozilla böngészőmotorját újra felhasználták más böngészőkben is, és az emberek elkezdtek a Geckora úgy hivatkozni, mint egy különálló böngészőmotorra, inkább mintsem egy szimpla webböngésző részre hivatkoztak volna.

2.1.2 Webszerver

A webkiszolgáló/webszerver egy kiszolgáló, mely elérhetővé teszi a helyileg (esetleg más kiszolgálón) tárolt weblapokat a HTTP protokollon keresztül. A HTTP webszerverekhez webböngészőkkel lehet kapcsolódni.

Bár a webszerverek többnyire különböznek a részletekben, az alapvető funkcióik azonosak. Minden webszerver HTTP kéréseket fogad a hálózatról, és HTTP válaszokat küld vissza. A HTTP válasz az esetek többségében egy HTML dokumentum, de lehet még egyszerű szöveges fájl, kép, vagy más típusú fájl is.

Kérés

A webszerverek a klienstől kapott kérésekben többek között **URL címet** kapnak, melyet aztán kétféleképpen értelmezhetnek:

- A tartománynév után álló relatív mappa és fájl struktúrát hozzárendelik egy gyökérmappához. (a gyökérmappa a webszerver beállításaiban van megadva, és az adatokat kérő kliens számára láthatatlan)
- A tartománynév után álló relatív mappa és fájlstruktúra (vagy akár még a tartománynév is) teljesen független a kért címben szereplő struktúrától. Ebben az esetben meghatározott szabályok szerint formázza a kért címet. Ennek segítségével egy mappára irányuló kérés teljesen más mappára vagy akár egy fájlra is mutathat és fordítva.

A kliens például az alábbi URL-t kéri:

```
http://www.pelda.com/utvonal/fajl.html
```

A kliens webböngészője ezt értelmezve létrehoz egy kapcsolatot a www.pelda.com kiszolgálóval, és elküldi a következő HTTP 1.1 kérést:

```
GET /utvonal/fajl.html HTTP 1.1
Host: www.pelda.com
…
```

Válasz

A www.pelda.com címet megfelelteti a webszerver az adott gyökérmappához (pl. */var/www/pelda*), amelyhez hozzáfűzi a */utvonal/fajl.html* elérést – ezzel megtörtént a megfeleltetés a fájlrendszer erőforráshoz. A kért eredmény a szerveren tehát: */var/www/pelda/utvonal/fajl.html*.

Ezt követően a webszerver ellenőrzi, hogy hozzáférhető-e az adott kérés, ill. hogy létezike. Ha nem létezett akkor 404-es hibakóddal tér vissza, ha hozzáférhető akkor beolvassa, elvégzi rajta az esetleges további műveleteket, majd elküldi a kliensnek. A válasz természetesen szintén magában foglalja a megfelelő fejlécet.

A második megoldás esetében, az erőforrásokhoz történő megfeleltetés előtt a címet átformázza. Például:

```
www.pelda.com/Toplista/kutyak+es+macskak
```

URL kérést a következőképpen alakíthatja át:

```
/var/www/pelda/toplista.php?cim=kutyak+es+macskak
```

Modulok

Lehetőség van a válaszok feldolgozása előtt, az esetlegesen a kérésben érkezett adatok feldolgozására és ennek eredményének visszaküldésére. Ilyenkor a szerver oldalon futó webszerver-modulok illetve a webszerver által meghívott CGI rutinok végzik el ezt a feladatot. A programrészletek (webszerver-modulok) rendszerint a HTML kódba vannak beágyazva és maga a webszerver-program hajtja ezeket végre. Ilyenek például a PHP, ASP, JSP.

2.1.3 Webtárhely

A **virtuális webtárhely szolgáltatás** egy olyan internetes szolgáltatást ért a köztudat, ahol egy webszerver erőforrásait több felhasználó között osztják fel. Minden felhasználó egy a rendszer által dedikált tárhelyet foglal el, aminek nyilvános tartalma egyedi domén néven érhető el. Kisebb forgalmú szájtot költséghatékonyan lehet bérelt webtárhelyen üzemeltetni.

A webtárhely szolgáltatásnak általában tartalmaznia kell **adminisztrációs felületet** a nagyszámú felhasználó kézbentartható, és hatékony kezeléséhez.

Osztott tárhelyszolgáltatók rendszerint az egyes szolgáltatásokat fizikailag elkülönített kiszolgáló rendszereken oldják meg, az ügyfélkiszolgáló és adminisztrációs rendszer, a levelező kiszolgáló, az adatbázis szerver, a webszerver fizikailag elkülönített kiszolgálókon működik. A legtöbb webkiszolgáló alacsony költségű Linux vagy FreeBSD operációs rendszer alapú **LAMP szerver**.

Az egyes operációs rendszerekre épített szolgáltatások lényegében meghatározzák a felhasználó által elérhető technológiák csoportját is. Windows alapú webhoszting esetén a felhasználó választhat akár ASP.NET és Microsoft SQL Server de akár PHP és MySQL Server támogatást is; míg LAMP szerver esetén csak PHP scriptnyelvű webszájtokat készíthet MySQL Server támogatással.

2.1.4 HTTP protokoll

A HTTP (HyperText Transfer Protocol) egy információátviteli protokoll a világhálón. Az eredeti célja a HTML lapok publikálása és fogadása volt. A HTTP fejlesztését a World Wide Web Consortium és az Internet Engineering Task Force koordinálta RFC-k formájában. A legfontosabb RFC az 1999-ben kiadott RFC 2616[22], amely a HTTP/1.1 verziót definiálja. Jelenleg ez a legelterjedtebb verzió.

A HTTP egy **kérés-válasz alapú protokoll** kliensek és szerverek között. A kommunikációt mindig **a kliens kezdeményezi**. A HTTP klienseket gyűjtőnéven user agentnek is nevezik. A user agent jellemzően, de nem feltétlenül webböngésző.

A HTTP általában a TCP/IP réteg felett helyezkedik el, de nem függ tőle.

[22] http://www.w3.org/Protocols/rfc2616/rfc2616.html

Kérés (request)

Egy HTTP kérés első sora mindig *metódus erőforrás verzió* alakú, például így:

```
GET /images/logo.gif HTTP/1.1
```

Ezt a sort követheti tetszőleges számú fejléc sor *HEADER: érték* alakban, például így:

```
Accept: text/plain,text/html
Accept-Language: en
```

A fejléc sorok végét egy **üres sor** jelzi, melyet az opcionális **üzenettest** követ. A sorokat a CRLF (azaz kocsi vissza + soremelés) karakterpárral kell elválasztani. A fejléc végét jelző üres sor csak ezt a két karaktert tartalmazhatja, nem lehet benne szóköz és tabulátor sem.

Metódusok

HTTP protokoll 8 féle metódust definiál. A metódusok a megadott erőforráson végzendő műveletet határozzák meg.

HEAD	Ugyanazt adja vissza, mint a GET, csak magát az üzenettestet hagyja ki a válaszból.
GET	A megadott erőforrás letöltését kezdeményezi. Ez messze a leggyakrabban használt metódus.
POST	Feldolgozandó adatot küld fel a szerverre. Például HTML űrlap tartalmát. Az adatot az üzenettest tartalmazza.
PUT	Feltölti a megadott erőforrást.
DELETE	Törli a megadott erőforrást.
TRACE	Visszaküldi a kapott kérést. Ez akkor hasznos, ha a kliens oldal arra kíváncsi, hogy a köztes gépek változtatnak-e illetve mit változtatnak a kérésen.
OPTIONS	Visszaadja a szerver által támogatott HTTP metódusok listáját.
CONNECT	Átalakítja a kérést transzparens TCP/IP tunnellé. Ezt a metódust jellemzően SSL kommunikáció megvalósításához használják.

Biztonságosnak azokat a metódusokat nevezzük, amelyek csak információ lehívására szolgálnak és elvben nem változtatják meg a szerver állapotát. Más szóval mellékhatás nélküliek. Ilyenek például a HEAD, a GET, az OPTIONS és a TRACE. Fontos megjegyezni, hogy a gyakorlatban lehetnek a biztonságos metódusoknak is szerveroldali mellékhatásai.

Válasz (response)

A HTTP válasz első sora a státuszsor, amely *verzió státuszkód indoklás* alakú. A státuszkód egy három számjegyű szám, az indoklás egy angol nyelvű üzenet. Az előbbit inkább gépi, az utóbbit inkább emberi feldolgozásra szánták. Egy egyszerű példa státuszsorra:

```
HTTP/1.1 200 OK
```

A státuszkódok jelentését az RFC 2616 tartalmazza részletesen, az alábbi lista egy áttekintő osztályozást ad a kezdő számjegy alapján:

1xx:	Informatív – Kérés megkapva.
2xx:	Siker – A kérés megérkezett; értelmezve, elfogadva.
3xx:	Átirányítás – A kérés megválaszolásához további műveletre van szükség.
4xx:	Kliens hiba – A kérés szintaktikailag hibás vagy nem teljesíthető.
5xx:	Szerver hiba – A szerver nem tudta teljesíteni az egyébként helyes kérést.

A státuszsor után fejléc sorok következhetnek a HTTP kérésnél látott módon, például így:

```
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```

A fejléc sorokat itt is üres sor zárja, melyet az opcionális üzenettest követ.

Státuszkódok

A HTTP/1.0 verzió óta a HTTP válasz első sorát státuszsornak nevezik és tartalmaz egy háromjegyű státuszkódot, valamint egy rövid szöveges üzenetet a hibáról. Például így:

```
HTTP/1.1 404 Not Found
```

A kliens elsősorban a hibakód, másodsorban a fejléc sorok tartalma alapján kezeli a választ. Használhatóak egyedi státuszkódok is, mert a kliens ismeretlen kód esetén az első számjegy alapján már tudja osztályozni a hibát.

A szöveges üzenetekre a szabvány csak javaslatokat tartalmaz. A szerver küldhet lokalizált üzeneteket is:

```
HTTP/1.1 404 Nincs meg.
```

Ha a státuszkód hibára utal, akkor a kliens megjelenítheti a hibaüzenetet, hogy tájékoztassa a felhasználót a hiba természetéről. A szabvány megengedi azt is, hogy a kliens maga interpretálja a státuszkódot és az alapján saját üzenetet generáljon a felhasználónak, de ez zavaró lehet. A szabvány szerint a státuszkódot szánják gépi, a szöveges üzenetet pedig emberi fogyasztásra.

Munkamenet (session)

A HTTP egy állapot nélküli protokoll. Az állapot nélküli protokollok előnye, hogy a szervernek nem kell nyilvántartania felhasználói információkat az egyes kérések kiszolgálása között. A HTTP eredetileg nem arra készült, hogy felhasználók jelentkezzenek be rajta keresztül szerverekre és ott munkamenetet (session-t) indítsanak. Történetileg azonban úgy alakult, hogy a HTTP terjedt el széles körben más, felhasználói bejelentkezést támogató protokollok helyett, ami arra kényszerítette a webfejlesztőket, hogy **kerülőutakon** járva tárolják a felhasználók munkamenet-állapotait.

2.1.5 FTP protokoll

A File Transfer Protocol, a TCP/IP hálózatokon történő állományátvitelre szolgáló szabvány.

Gyakran van szükség arra, hogy valamilyen állományt hálózaton keresztül töltsünk le saját gépünkre, vagy egy állományt mások számára hozzáférhetővé tegyünk. Erre alkalmas az FTP, ami lehetővé teszi a különböző operációs rendszerű gépek között is az információcserét. A világon nagy mennyiségű információforrás áll rendelkezésre, melyek letöltése ilyen módon megvalósítható. A hozzáférési jog alapján kétféle kapcsolattípus létezik:

- letöltés, vagy feltöltés **nyilvánosan hozzáférhető** állományokból vagy állományokba,
- letöltés, vagy feltöltés olyan gépről, ahol **azonosítóval rendelkezünk**.

Azt a folyamatot, amikor egy távoli számítógépről fájlt mentünk a saját számítógépünk háttértárára, **letöltésnek** nevezzük; **feltöltésnek** nevezzük, ha a folyamat fordított irányban zajlik, és mi töltünk fájlt más gépére.

Az FTP kapcsolat **ügyfél/kiszolgáló alapú**, vagyis szükség van egy kiszolgáló (szerver) és egy ügyfélprogramra (kliens). Elterjedt protokoll, a legtöbb modern operációs rendszerhez létezik FTP-szerver és kliens program, sok webböngésző is képes FTP-kliensként működni.

2.1.6 Webcím

A webcím, más néven URL (mely a Uniform Resource Locator [egységes erőforrás-azonosító] rövidítése), az Interneten megtalálható bizonyos erőforrások (például szövegek, képek) szabványosított címe. Először Tim Berners-Lee alkotta meg a World Wide Weben való használatra. A jelenleg használt formátumot részletesen leírja az IETF RFC 1738[23] szabványa.

A webcím az Internet történetének alapvető újítása. Egyetlen címben összefoglalja a dokumentum megtalálásához szükséges négy alapvető információt:

- a **protokollt**, amit a célgéppel való kommunikációhoz használunk;
- a szóban forgó gép vagy **tartomány nevét**;
- a hálózati **port számát**, amin az igényelt szolgáltatás elérhető a célgépen;
- a fájlhoz vezető **elérési utat** a célgépen belül.

Egy tipikus, egyszerű webcím így néz ki:

[23] http://tools.ietf.org/html/rfc1738

```
http://hu.wikipedia.org:80/wiki
```

Ennek részei:

- A *http* határozza meg a használandó protokollt. A protokoll neve után kettőspont (*:*) írandó.
- A *hu.wikipedia.org* adja meg a célgép tartománynevét. Ez elé két perjel (*//*) írandó.
- A *80* adja meg a célgép azon hálózati portszámát, amin kérésünket várja; ez elé kettőspont (*:*) írandó. Ezt a részt gyakran teljesen elhagyhatjuk, például esetünkben a http protokoll alapértelmezett portszáma a 80.
- A */wiki/Bit* a kért elérési út a célgépen. Ez a rész mindig a perjellel (*/*) kezdődik.

A legtöbb böngésző nem is igényli, hogy a *http://* részt begépeljük egy weblap eléréséhez, hiszen az esetek döntő többségében úgyis ezt használjuk. Egyszerűen begépelhetjük a lap címét, például: *hu.wikipedia.org/wiki/Bit*. A címlap megtekintéséhez általában elég a tartomány nevét beírni, például *hu.wikipedia.org*.

A webcímek egyéb részeket is tartalmazhatnak, *http* esetében például az elérési út után, egy kérdőjel (*?*) mögé helyezve keresési kérdés szerepelhet, ami egy *get* metódusú HTML űrlapból származik. Az elérési út után, attól egy kettős kereszttel (*#*) elválasztva szerepelhet a hiperszöveg egy részére hivatkozó azonosító. Ez az azonosító nem része a webcímnek, de gyakran szerepel vele kapcsolatban.

Példák:

```
http://hu.wikipedia.org/w/wiki.phtml?title=Bit&action=history
http://hu.wikipedia.org/wiki/1999#Események
```

A webcím az URI egy olyan fajtája, ahol az azonosítás a dokumentum helye alapján történik.

Ha egy weblapot egyértelműen meghatároz egy webcím, akkor **rá mutató hivatkozást hozhatunk létre**. Ez a helyzet nem mindig áll fenn, pl. egy menüpont megváltoztathatja a lapon belül az egyik keret tartalmát, anélkül, hogy ennek az új kombinációnak külön webcíme lenne. Ezen kívül egy weblap függhet ideiglenesen tárolt információtól is. Még ha van is egy weblapnak vagy keretnek önálló webcíme, ez nem mindig nyilvánvaló annak a számára, aki rá mutató hivatkozást kíván létrehozni. Egy keret webcíme nem látszik a címsávban, és létrehozható címsáv nélküli ablak is. A webcím ilyenkor a lap forrása, illetve egyes részeinek „tulajdonsága” alapján meghatározható.

Azon kívül, hogy rá mutató hivatkozást hozzunk létre, egyéb okokból is kíváncsiak lehetünk egy lap webcímére: ha tartalmát önállóan akarjuk megjeleníteni, illetve, ha bizonyos megszorításokat (eszköztár nélküli, vagy kicsi, méretezhetetlen ablak) meg akarunk kerülni.

2.2. Webdizájn

Mielőtt a web nyelveinek megismerésébe vetnénk magunkat, röviden térjünk ki a dizájn kérdésére is. (A dizájn alatt itt most nem egy konkrét grafikai látványterv készítését, hanem magasabb szintű alapelvek figyelembevételét értjük.)

2.2.1 Nyúló vagy fix elrendezés

A nyúló (liquid) dizájn a böngésző **vízszintes méretétől** függően nyúlik, vagy húzódik össze, míg a fix dizájn mindig megtartja a tervezésnél megadott méreteit. Mindkettőnek megvannak a maga előnyei és hátrányai.

A fix tervezésnél gondolni kell a legkisebb felbontással böngészőkre szintén, mint ahogy arra is, miként fog megjelenni a dokumentum egy tágasabb környezetben. A nyúló dizájn problémája elsősorban, hogy szépen leginkább egynemű tartalom esetén lehet tervezni, illetve a tipográfia: túl széles illetve keskeny szövegek nem barátságosak a szemnek. (Zárójelben: weben a sorkizárt forma nem mindig előnyös.)

Áthidaló megoldás lehet a tartalom méretének maximalizálása.

2.2.2 Flash

A flash a Macromedia Flash által kiadott szoftver; kiterjesztése: SWF. Maga az SWF formátumot más szoftverek is képesek létrehozni, de csak a flash lejátszóban fut. Kezdetben vektoros rajzprogramnak indult, majd programnyelve az Actionscript segítségével a webdizájn egyik legdinamikusabb alapeszközévé vált. Ma már mind a nyelv, mind a lejátszó igen előrehaladott állapotában van (platformfüggetlen Flash player 9 és az Actionscript 1-2-és 3).

Előnyei

- látványosan interaktív, moziszerű struktúrát, hatékony, egyben dinamikus multimédiatartalmat és alkalmazásokat hozhat vele létre a szerző (játékok, zene, film, stb)

Hátrányai

- ha a gépen nincs flash lejátszó, a felhasználóhoz nem jut el az információ amit a flash tartalmaz
- a teljes letöltődést meg kell várni, mielőtt használhatnánk az oldalt
- nem átméretezhető
- a keresők még mindig nem megfelelően indexelik
- nem lehet könyvjelzőt elhelyezni
- az idegesítő, villogó reklámok miatt sokan eleve kikapcsolják

A megfelelő cél érdekében használt Flash betétek fel tudnak dobni egy oldalt, de a túlzott alkalmazásuk inkább hátrányos. Teljes honlapok Flashban fejlesztése pedig nagyon ritkán indokolható meg.

2.2.3 Keretek

A frame, azaz a keretes rendszerek lehetővé tették, hogy a szerzők több keretben való megjelenítést alkalmazzanak – ezekbe a keretekbe külön weblapokat hívtak meg, amik együttes megjelenése adta ki az összefüggő tartalmat. Ezáltal bizonyos információk a képernyő egyik keretében állandóan, más infromációk egy másik keretében görgethető és cserélhető módon jelennek meg.

Tipikus formája volt: stabil tartalomjegyzék (például navigációs menü) plusz görgethető tartalmi felület, ami a tartalomjegyzékben szereplő tételekre kattintva aktualizálható. Ma már nem nagyon szabadna használni.

2.2.4 CSS vagy táblázatok

A táblázatok

A HTML táblázatmodellje lehetővé teszi, hogy az adatok legkülönbözőbb formáit (szövegek, előre formázott szövegek, képek, linkek, űrlapok, más táblázatok stb.) sorokba és oszlopokba rendezett cellákhoz rendeljék. Az oldal akkor táblázat alapú, ha a lap felépítését (egymásba ágyazott, többszörösen összetett) táblázatokkal oldjuk meg.

A problémák

- böngészők előbb a < / table > jelet olvassák be és csak aztán töltik be a tartalmat. Ez azt eredményezi, hogy míg odáig el nem jutott nem látunk az oldal tartalmából semmit.
- amennyiben nincs megfelelő DOCTYPE az elején (az úgynevezett „quirks” mód) a böngésző nem fogja helyesen értelmezni, előfordulhat, főleg ha a böngészőnk régebbi típus, hogy minden table elem megtöri az öröklődést, és többször is definiálnunk kell: például a betűtípust
- a táblázatmodell 3 szintű, a táblázat sorait reprezentáló elemekkel kezdve, amikbe a cellákat reprezentáló elemek ágyazódnak be, az adatokig, amik a cellákban vannak elhelyezve. Bonyolultabb táblázat esetén a HTML egyik legösszetettebb elemével találkozunk, amit nehézkes karbantartani, bővíteni, átlátni.

A CSS térhódításával egyre inkább elfogadott az a nézet, hogy a táblázatokat használják arra, amire „való”: tehát társított adatok összefüggő, egymáshoz viszonyított felsorolására.

A CSS

Jelenleg a legelterjedtebb és egyúttal szabványos stíluslap típus a W3C CSS típusa (Cascading Style Sheet.) Első változata CSS1 néven 1996 decemberében, a második, kibővített, több médiumot is lefedő CSS2 1998 májusában jelent meg. A CSS tulajdonképpen egy deklaratív nyelv, amely a HTML nyelv prezentációs képességeinek kibővítését szolgálja. Felhasználóbarát, könnyen olvasható és írható (de elrontani is könnyű).

A CSS-ben az elemek osztályaihoz, egyes HTML elemtípusokhoz és egyedi elemekhez különböző megjelenítési stílusokat definiálhatunk, s a konkrét jellemzőket egy messzemenően szélesebb tartományból választhatjuk, mint amit a HTML elemek jellemzőinek választéka kínál.

Társíthatunk vele különböző megjelenítési eszközökhöz különböző stílusokat is, akár egy laphoz többet. Elhelyezhetjük a dokumentum belsejében, vagy külső fájlban Ebben az esetben egy stíluslap egy egész honlaparculatát is meghatározhatja, ezzel sávszélesség kímélő.

Hátrányai

- A böngészők a legfőbb CSS1 és 2 tulajdonságot támogatják, de nem mindegyiket, sőt nem is mindig egyformán. Célszerű meggyőződni a támogatottságról adott tulajdonság használata előtt.

2.2.5 A „vakbarát" honlap

Az extrém média által is használható honlap ma már igényesebb készítő esetén alapkövetelmény.

Ebben az esetben gondot kell fordítani arra, hogy a képi, vagy flash formában elhelyezett információ hozzáférhető legyen szövegesen is.

A CSS segítségével megadhatjuk a szándékolt médium specifikációját. Lehet akár egyetlen médiumleíró, akár vesszővel elválasztott médiumlista. Szabvány média típusok így a tapintásos braille eszközök, kis – vagy kézi eszközök, autós navigációs eszközök, nyomtatók, kivetítők.

CSS2-ben a voice-family tulajdonságot például csak a hang alakú (auralis) feldolgozók számára vezették be.

2.2.6 A szép honlap és a működő honlap

Egy hatékony weboldal dizájnját úgy kell megtervezi és megvalósítani, hogy egyensúlyban legyen a HTML-formátumú szövegek és a grafikák aránya. A személyes, illetve a portfólió oldalakon kívül a honlapnak elsősorban a látogató igényeit kell kielégítenie.

Egy oldal lehet alul-, de a másik végletbe esve (pl. túl sok grafikai elem) túldizájnolt is. Olyan szempontokat kell figyelembe venni, mint az átláthatóság, a jó navigáció, sok szöveges tartalom esetén az olvashatóság, az on-line marketinget figyelembe véve pedig ide sorolható a márkázás, a közvetlen értékesítés, a fogyasztók megtartása, bizalom, és egyszerű kommunikáció. Az oldalnak továbbá mind megjelenésében, mind tartalmilag hitelesnek kell lennie.

2.2.7 A leggyakoribb webdizájnhibák

Hogy egy dizájn minden szempontból sikeres legyen, az ügyfél, a dizájner és a felhasználó jó érzéssel távozzon a honlapról, elsősorban a célját kell, hogy teljesítse. Más és más kritériumok merülnek fel egy céges site tervezésénél és egy on-line galéria esetében, de vannak apró hibák, amik közös jellemzők a webes megjelenítésnél.

Túl sok reklám

A honlap alapvető rendeltetése, hogy az oldalak tartalomjegyzékét megmutassa, az érdeklődést felkeltse, és esetleg más hasonló témájú honlapok felé irányítsa a látogatót. A honlap azonban mégsem tárgymutató és nem egy gigantikus hirdetőfal. A reklám eszköze, hogy magára irányítja a figyelmet, és sok banner, reklámcsík stb. esetén maga a tartalom vész el, főleg ha minden egyformán hangsúlyos. Megjegyzendő, hogy általában animációkkal szokták ezt a hangsúlyozást megoldani, mert a mozgás a periférikus látására erős hatással van és bevonzza a tekintetet. Ez hosszú távon zavaró lehet, a felhasználó nem kapja meg az olvasáshoz, befogadáshoz szükséges nyugalmat.

A betűk

A szövegnek olvashatónak kell lennie. Zavaró a túl kicsi, és a szövegtestben a túl nagy betűméret. Különleges betűtípusokat több szempontból sem éri meg font-ként alkalmazni, megeshet, hogy nem lesz képes a kliens gépe megjeleníteni és a fejlécen, stb. kívül zavaró is lehet, gyengíti a professzionalitás élményét.

Flash intró

A célja az lenne, hogy látványos klip segítségével felkeltsük a látogató kíváncsiságát, aki ezáltal beljebb lép és több időt is tölt a honlapon. Ezzel szemben tapasztalat, hogy kevesen nézik végig, időrablásnak tekintik főleg ha gyengén van elkészítve. Leginkább a filmes és más látványos vizuális témával foglalkozó oldalakon hasznos. Ajánlott mindig ellátni a következő két funkcióval: a „skip”- tehát a mozi átugrásának lehetősége, amivel rögtön a belső tartalomra kerül és a hang lekapcsolása. A keresőoptimalizálásnál szintén gondot okozhat, hogy az intro oldalak nem tartalmaznak kulcsszavakat, hanem általában csak egy linket, néha egy kereszthivatkozást.

Linkek

A HTML egyik legfontosabb sajátossága, hogy a dokumentumokat nem lineális módon az elejétől a végéig kell elolvasni, hanem a szerzők csoportosíthatják, egymáshoz rendelhetik a dokumentumokat, amelyben kapcsolatot definiálhat az egyes dokumentumokok belül és között. Célszerű a honlapról külső tartalomra mutató linkek eltérő jelzéssel történő megvalósítása, a fő és almenük jól látható hierachikus rendezése.

Ugyanígy hibának számítanak a törött linkek, amelyek nem vezetnek sehová, az általuk célzott tartalom már elmozdult, vagy megszünt. Jelölni kell az E-mail kapcsolattartást szolgáló linkeket is.

A kereső optimalizálás

Talán mind közül a legnagyobb hiba, amit a dizájnerek ill. a szerzők/tulajdonosok gyakran elkövetnek az, hogy a kereső gépekre történő optimalizálásra csak utólag gondolnak. Itt különösen problémát jelenthet a flash tartalom, főleg ha a honlap egésze flash alapú.

2.3. A HTML szabványok

A webfejlesztők gyakran küzdenek a különböző böngészőkben és böngészőverziókban is **használható** oldalak kialakításáért. Ebben a helyzetben különösen fontos, hogy a webes szabványoknak megfelelő oldalakat hozzunk létre. A szabványok alkalmazása ezen kívül a jövőben megjelenő verziókkal is jó eséllyel használható oldalakat eredményez. (A gyakorlatban a böngészők szabvány-követése az új verziók megjelenésével javulni szokott, esetleg stagnál, legalábbis a szabvány adott verzióját tekintve.)

Ha csapatban dolgozunk, egyszerűbb lesz a mások szabványos kódját megérteni és módosítani. (Hasonlóan, mint ahogy a programozás során alkalmazott kódolási konvenciók egységes betartása is megtérül a csapatmunkánál vagy a későbbi karbantartásnál.)

Vannak fejlesztők, akik úgy tekintenek a szabványokra, mint ami megköti a kezüket, és ezért böngésző-specifikus trükköket alkalmaznak az oldalak készítésénél. Hosszú távon azonban ez a hozzáállás nem lesz kifizetődő.

Végül a keresőrobotok is jobban tudják értelmezni a szabványos oldalakat. Sok „álszakértő” ajánl nyakatekert trükköket a keresőoptimalizálás érdekében, pedig a szabványok követése az egyik[24] legalapvetőbb teendő.

2.3.1 A World Wide Web Consortium[25] (W3C)

A W3C Tim Berners-Lee, a WWW kitalálója által 1994-ben alapított szervezet. Célja, hogy a webből a lehető legtöbbet lehessen kihozni. Elsődleges tevékenysége a web szabványok (egész pontosan ajánlások) kidolgozása. A legfontosabb tagjai:

- IBM
- Microsoft
- America Online
- Apple
- Adobe
- Macromedia
- Sun Microsystems
- Budapesti Műszaki és Gazdaságtudományi Egyetem (2006 óta)

Akár személyesen is bekapcsolódhatunk az ajánlások kidolgozásába a W3C Magyar Iroda[26] által koordinált módon.

2.3.2 Validátorok

A W3C és más szervezetek is készítenek olyan programokat, honlapokat, amelyekkel a honlapunk szabványossága (szabványnak való megfelelése) tesztelhető. A hibák mellett sokszor segítséget is kapunk a szolgáltatások igénybevételével.

A fontosabb validátorokkal a későbbiekben fogunk ismertetni.

2.4. Ellenőrző kérdések

- Melyik a böngészők által támogatott legmagasabb HTML (nem XHTML!) verzió?
- Minek a rövidítése: HTTP, FTP, HTML, W3C?
- Mi a W3C feladata?
- Melyek jelenleg a szabványkövető(bb) böngészők?
- Milyen verziónál tartanak ma a népszerűbb böngészők?

[24] Ami még ennél is fontosabb, az a rendszeresen frissülő és minőségi tartalom.

[25] http://www.w3.org/

[26] http://www.w3c.hu/

3. HTML

A HTML nyelv az az alap, amivel minden webfejlesztőnek alaposan tisztában kell lenni. Ez a fejezet segítséget ad a HTML lehetőségeinek megismeréséhez, de sok nyelvi elemet már nem tartalmaz. A terjedelmi okokon kívül a következőkre kell elsősorban gondolni:

- Bizonyos HTML jellemzők a mai napra elavultnak tekinthetők. Itt elsősorban a kinézet esztétikai megjelenésére kell gondolni. A CSS használatával ugyanis sokkal több és jobb lehetőségünk lesz a kinézet leírására. A HTML a mai gyakorlatban már tisztán csak az információra, és annak struktúrájára figyel. Ezt egyébként **szemantikus kódolásnak** is nevezzük.
- Bizonyos tagok, tulajdonságok a böngészők által nem egységesen támogatottak, így ezeket a gyakorlatban is csak kevesen használják.

3.1. Bevezetés

Mi az a HTML?

- A HTML a *Hyper Text Markup Language* rövidítése
- A HTML állomány egyszerű szövegállomány, amely rövid jelölő tagokat tartalmaz
- A jelölő tagok alapján tudja a böngésző, hogyan kell megjelenítenie az oldalt
- A HTML állomány *html* kiterjesztéssel rendelkezik
- A HTML állományt egyszerű szöveges (*editor*) programokkal (pl. Jegyzettömb) is létrehozhatunk

Hogyan kezdjünk neki?

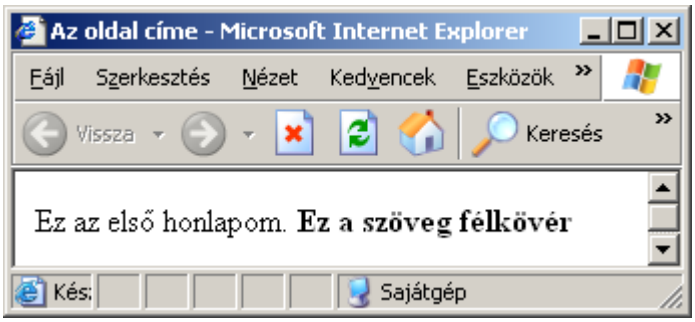
Windows operációs rendszer alatt indítsuk el a Jegyzettömböt (vagy inkább egy komolyabb editort), majd gépeljük be a következő szöveget:

```
<html>
  <head>
    <title>Az oldal címe</title>
  </head>
  <body>
    Ez az első honlapom. <b>Ez a szöveg félkövér</b>
  </body>
</html>
```

Mentsük el az oldalt *oldal.html* néven!

Megjegyzés: Ebben és a következő néhány fejezetben még nincs feltétlen szükségünk webszerver elérésére a tanuláshoz. Később majd FTP kapcsolaton keresztül a webszerverre fogjuk az oldalainkat feltölteni, majd webszerver elérésével használni azokat.

Nyissuk meg a böngészőnket, majd a Fájl menü megnyitás parancsát választva keressük meg az előbb elmentett *oldal.html* állományt! (További lehetőségünk, hogy a Windows Intézőnkben duplán kattintunk az állomány nevére, de az állomány böngészőre vonszolásával is célt érünk.) A következőhöz hasonlót kell látnunk a böngészőnkben:



3.1. ábra: HTML oldal a böngészőben

A példa magyarázata

A dokumentum első tagja a <*html*>. A böngésző erről fogja tudni, hogy hol kezdődik a HTML oldal. Az utolsó tag a </*html*>, itt ér véget a dokumentum a böngésző számára.

A <*head*> és </*head*> tagok közötti rész a fejléc információ. Az itt megjelenő szöveget a böngésző nem jeleníti meg közvetlenül.

A <*title*> tagok közötti szöveget a böngésző a címsorában jeleníti meg.

A <*body*> tagok közötti szöveg jelenik meg a böngésző ablakában.

A <*b*> és </*b*> tagok hatására a szöveg félkövéren (*bold*) jelenik meg.

HTML szerkesztők

Léteznek olyan szerkesztőprogramok, amelyekkel tényleges HTML ismeretek nélkül is lehet HTML oldalakat létrehozni. Ezeket a programokat *WYSIWYG* (*what you see is what you get*) *editoroknak* hívjuk, ismertebb, pl. a *FrontPage*, vagy a *Word*, ezek azonban kerülendők, ha minőségi HTML oldalakat szeretnénk létrehozni. Ezek a programok ugyanis kisebb-nagyobb mértékben „teleszemetelik” a kódot. (Elrettentésként érdemes megnézni egy *Word*-ből mentett weblapot.)

Érdemes inkább olyan szerkesztőprogramot választani, ahol a HTML kód fölött teljes ellenőrzéssel bírunk, ugyanakkor kiegészítő szolgáltatásokkal (pl. színkiemelés, tagok beszúrása gombnyomásra stb.) gyorsítani lehet a munkát.

Gyakran ismételt kérdések

Kérdés: A böngésző a HTML tagokkal együtt jeleníti meg a begépelt szöveget. Miért?

Válasz: A Windows alapértelmezett beállítása szerint a Jegyzettömb a *html* kiterjesztés után még egy *txt* kiterjesztést is tesz, tehát az állománynév (annak ellenére, hogy ez nem is látszik): *oldal.html.txt*. Érdemes más szerkesztő programot, pl. Total Commandert alkalmazni, vagy a Mappa beállításai között a bűnös „Ismert állománytípusok esetén a kiterjesztés elrejtése” beállítást kikapcsolni.

Kérdés: Változtattam a HTML dokumentumon, de a változások nem jelentek meg a böngészőben. Miért?

Válasz: A szerkesztő programban menteni, majd a böngészőben frissíteni kell a dokumentumot. (Nem szükséges semelyik programot bezárni, csak menet közben váltogatni a két program között.) Bizonyos esetekben a böngésző gyorsítótárából veszi az oldalt, és nem küld újabb kérést a szerver felé. Ilyenkor a gyorsítótár kiürítése vagy a Ctrl+F5 billentyűkombináció alkalmazása segít.

Kérdés: Melyik böngészőt használjam?

Válasz: A tanuláshoz célszerű olyan böngészőt választani, amely a szabványokat a lehető legjobban követi. A profi fejlesztőnek egyébként is minden elterjedtebb böngészőn és verzión tesztelni kell az oldalt. 2007 nyarán hazánkban és a környező országokban a böngészők 30-40%-a Firefoxot használ.

3.2. Tagok

A HTML állomány egyszerű szövegállomány, amely rövid jelölő tagokat tartalmaz.

A HTML tagok segítségével elemek definiálhatók.

HTML tagok jellemzői

- A HTML tagok jelölik ki a HTML elemeket
- A HTML tagot a < és > írásjelek veszik körül
  (ezek az írásjelek az egyszerű szövegekben nem engedélyezettek)
- A HTML tagok általában párban vannak, mint a <*b*> és </*b*>
- A pár első tagja a kezdő, a második a záró tag
- A szöveg (tartalom) a kezdő és a záró tag között helyezkedik el
- A HTML tagok kis-, és nagybetűvel is írhatók

HTML elemek

Az előző példában a következő példa egy elem:

```
<b>Ez a szöveg félkövér</b>
```

A HTML elem kezdő tagja <*b*>, a tartalmazott szöveg *Ez a szöveg félkövér*, és a záró tag </*b*>.

A következő is egy HTML elem:

```
<body>
  Ez az első honlapom. <b>Ez a szöveg félkövér</b>
</body>
```

Az elem kezdő tagja <*body*> és a záró tagja </*body*>

Miért alkalmazzunk kisbetűs tagokat?

Maga a HTML nyelv nem érzékeny a kis-, és nagybetűkre, de a HTML továbbfejlesztésének tekinthető XHTML már igen. Érdemes tehát ezt az írásmódot megszokni.

Tag tulajdonságok (attribútumok, jellemzők)

A tagok tartalmazhatnak tulajdonságokat is. Ezek a jellemzők járulékos információk az elem egészére nézve.

A <*body*> tag definiálja az oldal *body* elemét, ami tartalmazhat egy *bgcolor* tulajdonságot, amiből tudja a böngésző, hogy milyen háttérszínnel kell az oldalt megjeleníteni. Például, ha piros háttérszínt szeretnénk, a következő szöveget kell begépelnünk:

```
<body bgcolor="red">
```

A *<table>* elem egy táblázat elemet definiál. A tag tulajdonságaként megadható, hogy milyen vastag szegéllyel jelenjen meg a táblázat. A következő példa szegély nélkül jeleníti meg a táblázatot:

```
<table border="0">
```

A tulajdonságok név-érték párokkal adhatók meg, egymástól szóközzel elválasztva akár több is.

Melyik idézőjelet alkalmazzuk?

A nyelv elsősorban a (dupla) idézőjel alkalmazását írja elő. A böngészők az aposztróf jelet is elfogadják, mégis érdemes inkább a hagyományos idézőjelet alkalmazni. Bizonyos esetekben (pl. soron belüli JavaScript kód) a kettőt keverten kell alkalmaznunk.

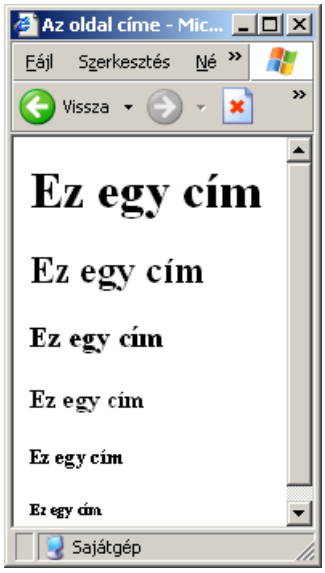
3.3. Alapvető HTML tagok

A HTML legalapvetőbb és leggyakrabban használt tagjai azok, amelyek segítségével címeket, bekezdéseket és sortöréseket lehet létrehozni.

Címek

A címek a *<h1>* ... *<h6>* tagok segítségével adhatók meg. *<h1>* a legnagyobb (legfelsőbb szintű) címet jelenti, *<h6>* pedig a legkisebbet. (Általában egy oldalon legfeljebb 2-3 szintet indokolt alkalmazni, ekkor pl. a *h1* és *h2* alkalmazható.) A következő példa bemutatja mind a 6 címet:

```
<h1>Ez egy cím</h1>
<h2>Ez egy cím</h2>
<h3>Ez egy cím</h3>
<h4>Ez egy cím</h4>
<h5>Ez egy cím</h5>
<h6>Ez egy cím</h6>
```



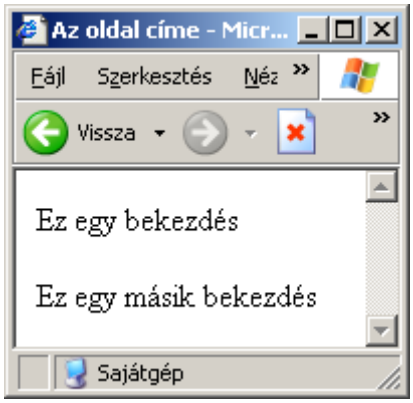
3.2. ábra: Címsorok

A címsorokhoz a böngésző alapértelmezetten térközöket is kapcsol.

Bekezdések

A bekezdéseket a <*p*> taggal lehet definiálni:

```
<p>Ez egy bekezdés</p>
<p>Ez egy másik bekezdés</p>
```

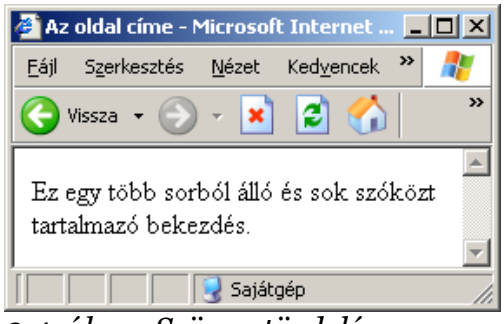


3.3. ábra: Bekezdések

A bekezdésekhez a böngésző alapértelmezetten térközöket is kapcsol.

A következő példában hiába szerepel az újsor és a több szóköz karakter, a böngésző minden elválasztó karakter-sorozatot egy szóközként értelmez és jelenít meg. A tényleges tördelést mindig a böngésző ablakmérete és a benne levő szöveg határozza meg.

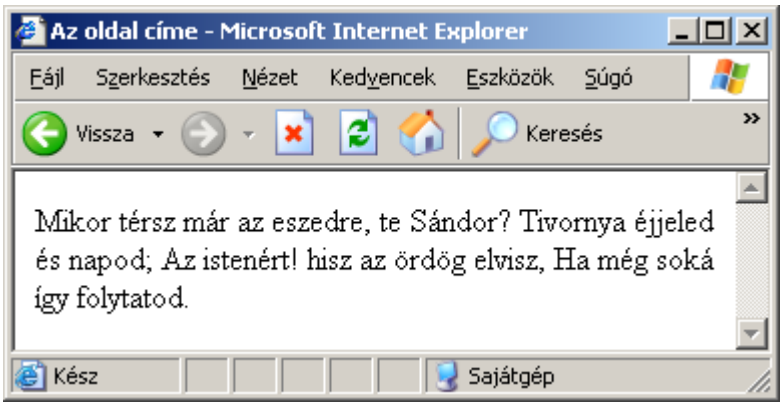
```
<p>
  Ez egy   több
  sorból   álló   és   sok
  szóközt    tartalmazó
  bekezdés.
</p>
```



3.4. ábra: Szöveg tördelése

A következő példa jól mutatja a kötött szerkezetű szöveg (például vers) idézésének nehézségét:

```
<p>
  Mikor tersz már az eszedre, te Sándor?
  Tivornya éjjeled és napod;
  Az istenért! hisz az ördög elvisz,
  Ha még soká így folytatod.
</p>
```

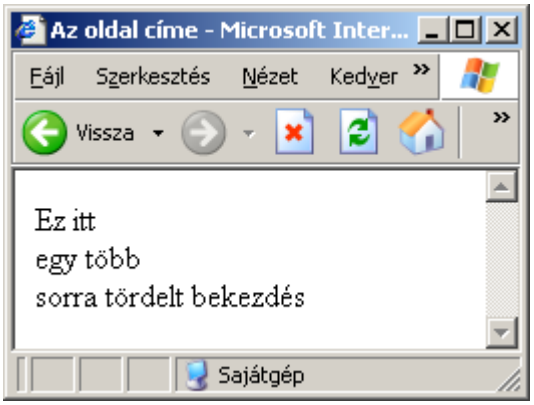


3.5. ábra: Szöveg tördelése

Sortörések

A <*br*> tag használható, ha új sort szeretnénk kezdeni, de nem akarunk új bekezdést kezdeni. A *br* kikényszeríti a sortörést.

```
<p>Ez itt<br>egy több<br>sorra tördelt bekezdés</p>
```



3.6. ábra: Több soros szöveg

A *br* elem üres, vagyis nincs külön záró tagja.

Megjegyzések

A megjegyzés tagot megjegyzések elhelyezésére használjuk. A böngésző nem veszi figyelembe a megjegyzésbe írt szöveget. (Természetesen a megjegyzés megtekinthető a forráskódban.)

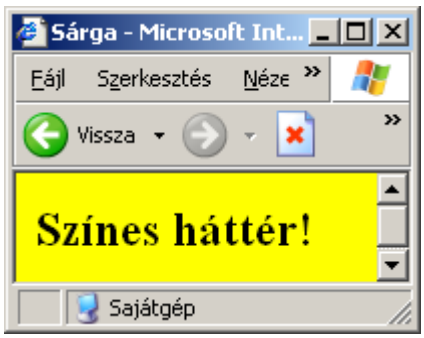
```
<!—Ez egy megjegyzés -->
```

A megjegyzésben nem fordulhat elő két kötőjel a > nélkül.

Színes háttér

A következő példa bemutatja, hogyan lehet háttérszínt megadni a HTML oldalunkhoz.

```
<html>
  <head>
    <title>Sárga</title>
  </head>
  <body bgcolor="yellow">
    <h2>Színes háttér!</h2>
  </body>
</html>
```



3.7. ábra: Színes háttér

Hasznos tippek

- Amikor HTML oldalakat hozunk létre, mindig nézzük meg más böngészőben, más képernyő (ablak) felbontásban is.
- Soha ne szóközökkel és Enterrel akarjunk szöveget formázni.
- A böngésző eldobja a szövegben talált többszörös szóközöket, és egyetlen szóközként veszi figyelembe.
- Üres *p* tagok használata szintén nem javasolt, inkább a *br* tag segítségével érdemes a sortörést kikényszeríteni.
- Sokan elhagyják a *p* záró tagot a bekezdés végéről. Az XHTML szigorúbb szabályai miatt ezt nem érdemes alkalmazni.
- Az oldal szakaszokra töréséhez alkalmazható a *hr* vízszintes vonallal elválasztó tag. Ez a tag is üres, záró pár nélküli.

Összefoglaló táblázat

Tag	Leírás
<*html*>	HTML dokumentumot definiál
<*body*>	A dokumentum törzsét definiálja
<*h1*> ... <*h6*>	Címsorokat definiál
<*p*>	Bekezdést definiál

<*br*>	Egyszerű sortörést szúr be
<*hr*>	Vízszintes elválasztó vonalat szúr be
<*!*-- ... -->	Megjegyzést definiál

Feladat

Készítsen egy tetszőleges költő verseit tartalmazó oldalt!

- Az oldal főcíme a költő neve legyen
- Legalább három verset tartalmazzon:
  - A verscímek alacsonyabb szintű címek legyenek
  - Versszakonként egyetlen bekezdést alkalmazzon, a sorok végén sortöréssel
  - A verseket egy vízszintes vonal is válassza el egymástól

CSS feladatok

- Alkalmazzon a fenti oldalra háttér-, és szegélyformázásokat, állítson be megfelelő térközöket.
- Készítsen szamárfüles bekezdést!

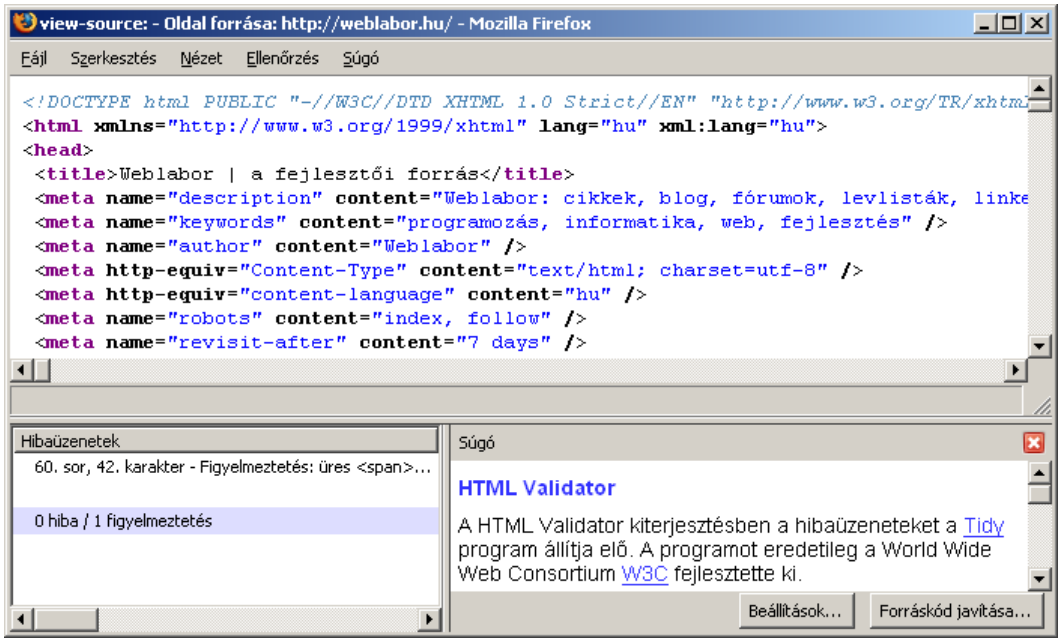
3.4. Hogy nézzük meg egy oldal HTML kódját?

Gyakran előfordul, hogy a weben böngészve megtetszik egy oldal, és szeretnénk megnézni a forrását. (A szerző véleménye szerint ez az egyik legjobb módszer a tanulásra, hiszen ekkor nem külső, hanem belső motiváló erő hat.) Hogyan tehetjük ezt meg?

Keressük meg a böngészőnk *Nézet* menüjét, majd *Forrás*, vagy *Oldal forrása* (vagy valami hasonló nevű) menüpontot.

A szerző javasolja a fejlesztéshez a *Firefox* nevű böngészőt, amely eleve webfejlesztők számára lett kifejlesztve, és több ezer kiterjesztése (*plug-in*) közül jó néhány a HTML forrás könnyen áttekinthető megjelenítését szolgálja. Következzen a teljesség igénye nélkül néhány kapcsolódó kiterjesztés.

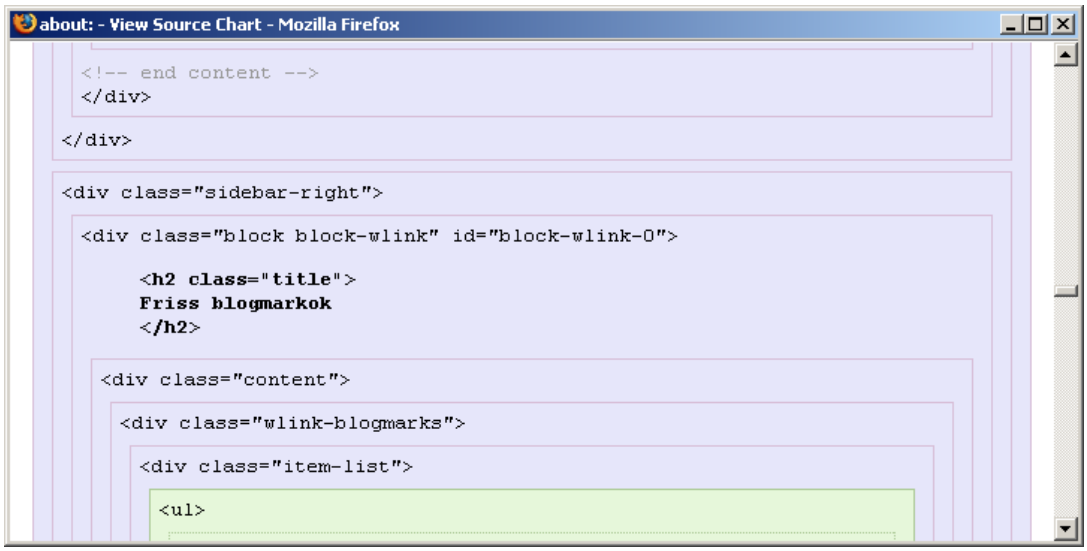
Tidy HTML Validator[27]



3.8. ábra: Forrás a Tidy HTML Validator megjelenítésével

A képen látható, ahogy a színkiemelés alatt a szintaktikai hibák és a hibák részletes leírása is megtekinthetők. Kis ügyességgel (Forráskód javítása gomb) akár a hibák egy részétől is megszabadulhatunk.

View Source Chart[28]



3.9. ábra: View Source Chart működése

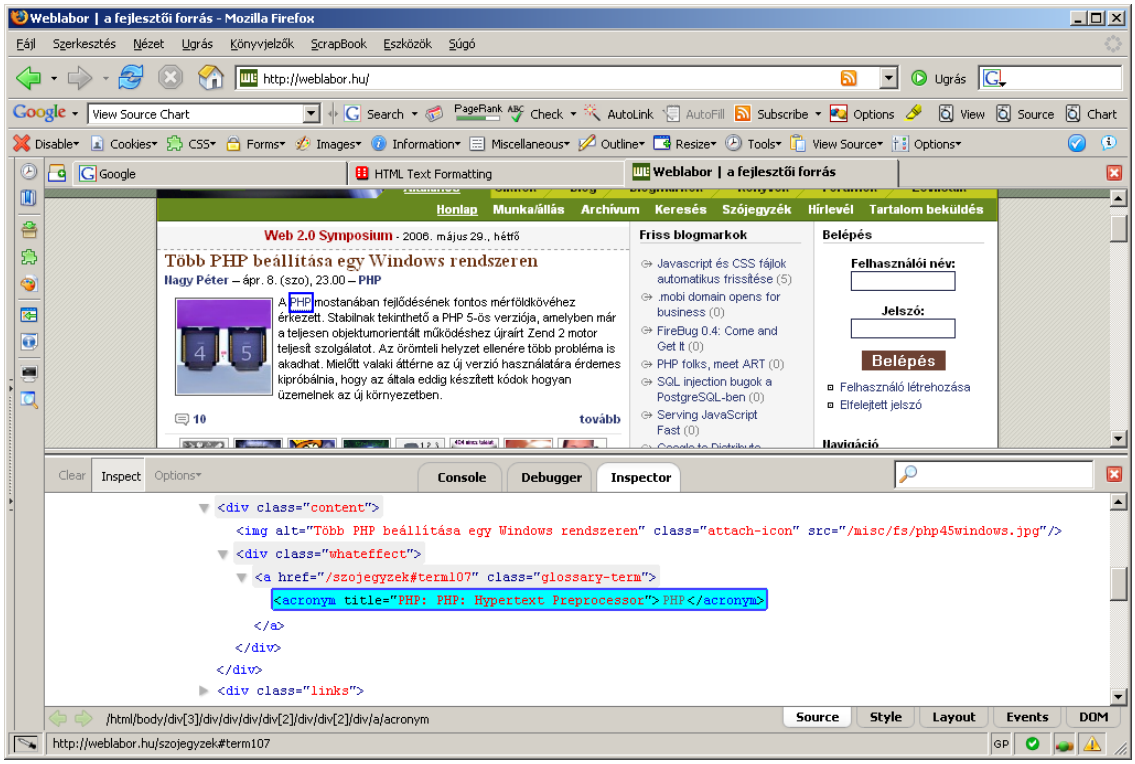
Ennek a kiterjesztésnek fő szolgáltatása a kód színezéssel történő megjelenítése. Ráadásul az ábrán látható módon az egymásba ágyazási hierarchia követhető a vonalak alapján, és ezek a dobozok kattintásra kinyílnak-becsukódnak.

---

[27] http://users.skynet.be/mgueury/mozilla/

[28] http://jennifermadden.com/scripts/ViewRenderedSource.html

FireBug[29]



3.10. ábra: FireBug

A kiterjesztés az oldalt és a forráskódját teljesen szimultán mutatja, érdemes megfigyelni az egérkurzorral irányítható kék dobozokat.

3.5. Formázás

A HTML definiál néhány formázásra szolgáló elemet, mint pl. félkövér vagy dőlt szöveg formázásához.

Szövegformázó tagok

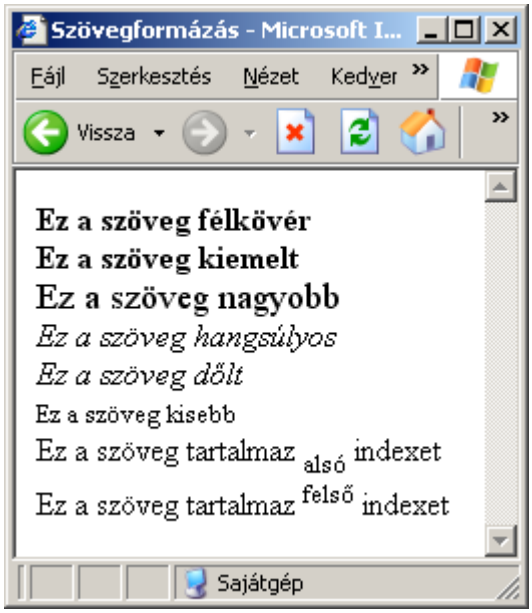
Tag	Leírás
<*b*>	Félkövér szöveget definiál
<*big*>	Nagyobb szöveget definiál
<*em*>	Kiemeli a szöveget
<*i*>	Dőlt szöveget definiál
<*small*>	Kisebb szöveget definiál
<*strong*>	Erősebb kiemelés

[29] http://www.joehewitt.com/software/firebug/

<sub>	Alsó indexet definiál
<sup>	Felső indexet definiál
<ins>	Beszúrt szöveget jelöl
<del>	Törölt szöveget jelöl

A következő példa mutatja a tagok hatását:

```
<b>Ez a szöveg félkövér</b>
<br>
<strong>Ez a szöveg
  kiemelt</strong>
<br>
<big>Ez a szöveg nagyobb</big>
<br>
<em>Ez a szöveg hangsúlyos</em>
<br>
<i>Ez a szöveg dőlt</i>
<br>
<small>Ez a szöveg
  kisebb</small>
<br>
Ez a szöveg tartalmaz
  <sub>alsó</sub> indexet
<br>
Ez a szöveg tartalmaz
  <sup>felső</sup> indexet
```

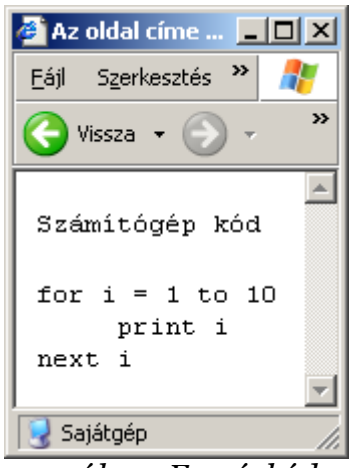


3.11. ábra: Szövegformázások

Meg kell jegyezni, hogy azokat a tagokat érdemes előtérbe helyeznünk, amelyek nem a kinézetre, hanem a jelentésre utalnak. Éppen ezért kerülendő pl. a *b* és *i* tagok használata, helyette a *strong* és *em* tagok javasolhatók, amit persze CSS segítségével kinézetben is testre szabhatunk.

Számítógép kimenetet definiáló tagok

Tag	Leírás
<code>	Forráskódot definiál
<pre>	Előformázott szöveget definiál: az elválasztó (*white space*) karaktereket nem a HTML-ben szokásos, hanem direkt módon értelmezi



3.12. ábra: Forráskód

Idézet, kiemelés és definíciós tagok

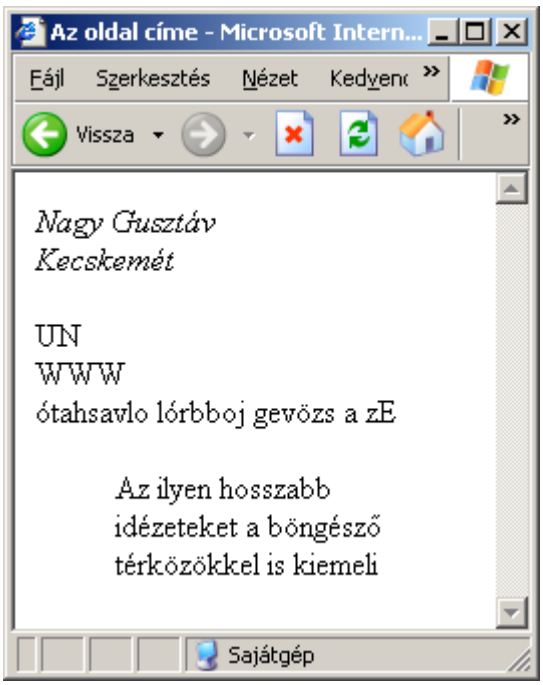
Tag	Leírás
<abbr>	Rövidítést definiál
<acronym>	Mozaikszót definiál
<address>	Cím elemet definiál
<bdo>	Szöveg írásirányt határoz meg
<blockquote>	Hosszú (akár több bekezdéses) idézetet jelöl
<q>	Rövid (általában bekezdésen belüli) idézetet jelöl
<cite>	Idézetet jelöl
<dfn>	Definíciót jelöl

```
<address>
  Nagy Gusztáv<br>
  Kecskemét
</address>
<br>
```

```
<abbr title=
  "United Nations">UN</abbr>
<br>
<acronym title="World Wide
  Web">WWW</acronym>
<br>

<bdo dir="rtl">
  Ez a szöveg jobbról
  olvasható
</bdo>

<blockquote>
  Az ilyen hosszabb idézeteket
  a böngésző térközökkel is
  kiemeli
</blockquote>
```

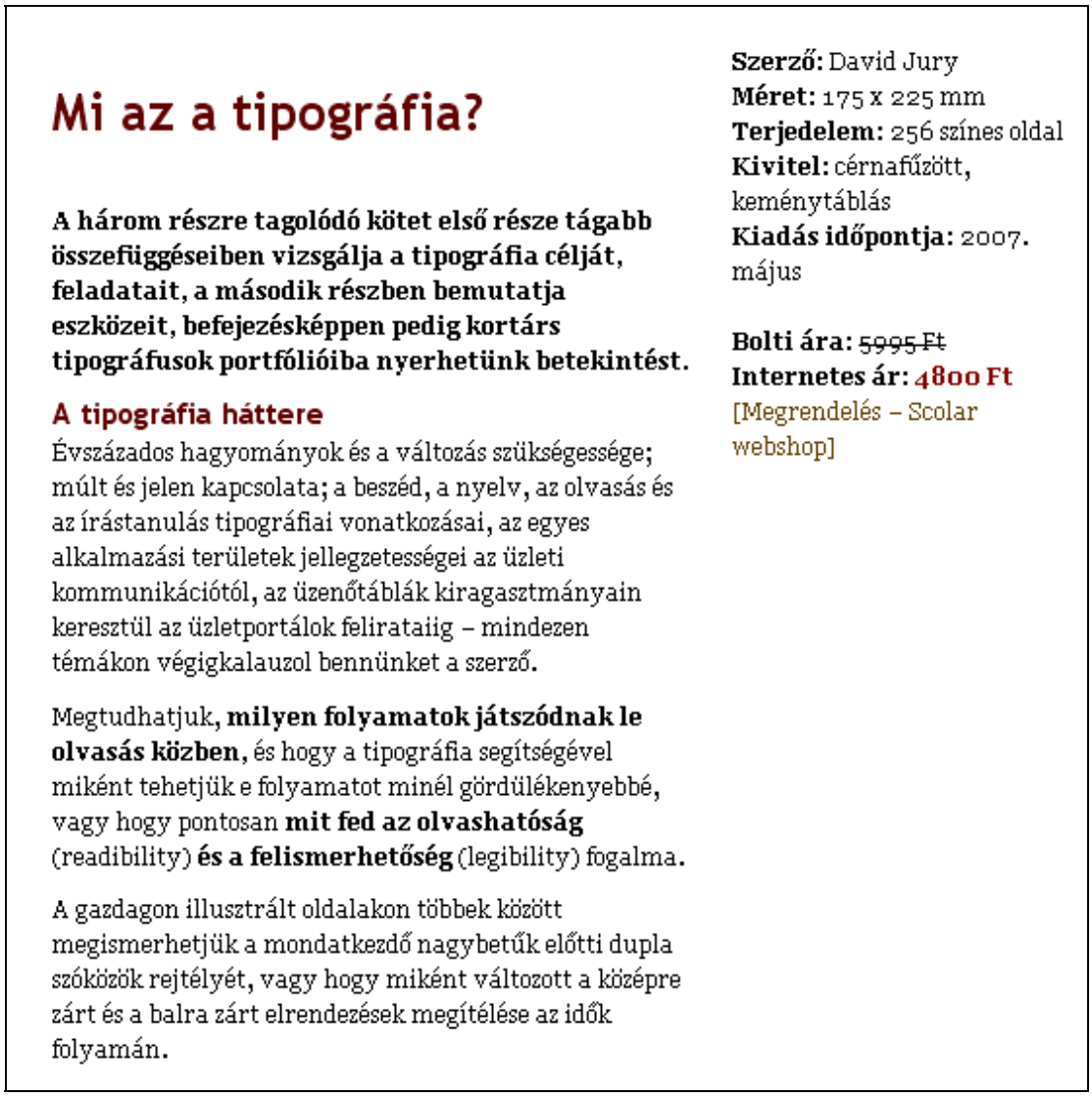


3.13. ábra: Idézet, definíció

Feladat

Készítsen HTML oldalakat, amelyek a következő két képen[30] látható HTML tartalomra a lehető legjobban hasonlítanak.

[30] A kép a http://design.scolar.hu/tipografia.html oldal felhasználásával készült



Mi az a tipográfia?

A három részre tagolódó kötet első része tágabb összefüggéseiben vizsgálja a tipográfia célját, feladatait, a második részben bemutatja eszközeit, befejezésképpen pedig kortárs tipográfusok portfólióiba nyerhetünk betekintést.

A tipográfia háttere

Évszázados hagyományok és a változás szükségessége; múlt és jelen kapcsolata; a beszéd, a nyelv, az olvasás és az írástanulás tipográfiai vonatkozásai, az egyes alkalmazási területek jellegzetességei az üzleti kommunikációtól, az üzenőtáblák kiragasztmányain keresztül az üzletportálok feliratáig – mindezen témákon végigkalauzol bennünket a szerző.

Megtudhatjuk, **milyen folyamatok játszódnak le olvasás közben**, és hogy a tipográfia segítségével miként tehetjük e folyamatot minél gördülékenyebbé, vagy hogy pontosan **mit fed az olvashatóság** (readibility) **és a felismerhetőség** (legibility) fogalma.

A gazdagon illusztrált oldalakon többek között megismerhetjük a mondatkezdő nagybetűk előtti dupla szóközök rejtélyét, vagy hogy miként változott a középre zárt és a balra zárt elrendezések megítélése az idők folyamán.

Szerző: David Jury
Méret: 175 x 225 mm
Terjedelem: 256 színes oldal
Kivitel: cérnafűzött, keménytáblás
Kiadás időpontja: 2007. május

Bolti ára: ~~5995 Ft~~
Internetes ár: **4800 Ft**
[Megrendelés – Scolar webshop]

CSS feladatok

- Készítsen rövidítéseket (*abbr*) tartalmazó oldalt, amely a tantárgy során eddig előfordult rövidítéseket tooltip-ként megjeleníti.
  CSS segítségével oldja meg, hogy az ilyen rövidítések sárga háttérszínnel és szaggatott piros szegéllyel jelenjenek meg.
- Készítsem idézeteket tartalmazó oldalt. Használjon a *blockquote* elemekhez idézőjelet ábrázoló képet!

3.6. Karakter entitások

Bizonyos karakterek (mint például a < és >) speciális jelentésűek a HTML-ben, ezért nem használhatók a folyó szövegben. Ha egy ilyen speciális karaktert akarunk megjeleníteni, akkor karakter entitást kell alkalmaznunk.

A karakter entitás három részből áll: *&* az elején, *;* a végén, a kettő között pedig egy entitás név, vagy # és számkód. Ha például a < jelet szeretnénk megjeleníteni, akkor a dokumentumba az *<* vagy a *<* karaktersorozatot kell gépelnünk.

Megjegyzés: az entitások kis-nagybetű érzékenyek!

Nem törhető szóköz

A gyakorlatban talán a legtöbbet alkalmazott karakter entitás a nem törhető szóköz. A HTML a több egymást követő elválasztó (ún. *white space*) karaktereket csak egy szóközként jeleníti meg. Ilyen esetekben szokás a * * entitást egymás után többször alkalmazni, ugyanis ezeket ténylegesen figyelembe veszi a böngésző. Ez azonban nem felel meg a HTML eredeti céljának, és a mai technikai lehetőségeknek sem. (Stílus formázások segítségével ezek a problémák sokkal elegánsabban oldhatók meg.)

Ennek az entitásnak eredetileg az a célja (és a szerző véleménye szerint csak ilyen esetben szabadna alkalmazni), hogy a több szóból álló kifejezések (például tulajdonnév) esetén a sor végén ne törje szét a böngészőnk a kifejezést, hanem mindenképpen egy sorba kerüljenek. Például a következő név mindig egy sorba, tördelés nélkül fog kerülni:

```
Nagy Gusztáv
```

Ékezetes karakterek

Az angol abc-ben nem szereplő karakterek (így a magyar nyelv ékezetes karakterei is) sokáig problémát okoztak a HTML szerkesztés során. Ezért korábban szokás volt az ékezetes karaktereket is entitások segítségével megadni. A mai napra azonban ezek a problémák lényegében megszűntek, ezért a szerző véleménye szerint teljesen indokolatlan az entitások alkalmazása. Helyette inkább a pontos karakterkódolásra érdemes figyelmet fordítani.

További karakter entitások

Leírás	Jelentés	Entitás név	Entitás számkód
	Nem törhető szóköz	* *	* *
<	Kisebb, mint	*<*	*<*
>	Nagyobb, mint	*>*	*>*
&	És	*&*	*&*
"	Idézőjel	*"*	*"*
'	Aposztróf	*'*	*'*
§	Paragrafus	*§*	*§*
©	Copyright	*©*	*©*

Az entitások teljes listáját a HTML referenciában érdemes keresni.

3.7. Linkek

A HTML linkeket (hivatkozásokat) használ arra, hogy az oldalunkhoz más tartalmakat kapcsolhassunk.

A horgony (<*a*>) tag és a *href* tulajdonság

Egy horgony hivatkozni tud egy tetszőleges webes erőforrásra, pl. egy HTML oldalra, egy képre, zenére stb.

A horgony szintaxisa a következő:

```
<a href="url">Megjelenő szöveg</a>
```

A *href* tulajdonságoz rendelt érték határozza meg, hogy a böngésző hogyan reagáljon a link kiválasztására. (Itt nem csak kattintás jöhet szóba, hiszen billentyűzetről is lehet linket kiválasztani a TAB segítségével, és akár gyorsbillentyű (*accesskey*) is rendelhető egy horgonyhoz.) A kezdő és a záró tag közötti szöveg (vagy akár bonyolultabb tartalom) lesz kattintható, és (alapértelmezetten) kék színnel aláhúzott link.

A következő példa egy linket definiál a Weblabor honlapjára:

```
<p>A <a href="http://weblabor.hu/">Weblabor</a> honlapja.</p>
```

A *target* tulajdonság

Alapértelmezetten egy link kiválasztása esetén az új oldal az aktuális helyett jelenik meg. Ez azonban módosítható.

A következő link egy új ablakban nyitja meg a Weblabor honlapját:

```
<a href="http://weblabor.hu/" target="_blank">Weblabor</a>
```

Ez a tag mára már elvesztette a jelentőségét, az XHTML már nem is teszi lehetővé a tulajdonság használatát. Sokan felteszik a kérdést: Miért? Itt elsősorban azt a szemléletet kell látni, hogy a felhasználó dönthesse el, hogy mi hol jelenjen meg. Nem szerencsés, ha erről a webfejlesztő helyette dönthet.

A *name* tulajdonság

Egy hivatkozás alapértelmezetten a HTML oldal tetejét jelenti. Néha azonban praktikus, ha egy oldalon belül is pontosítani tudjuk a link célját. Erre szolgál ez a tulajdonság.

A következő módon tudunk létrehozni egy ugrási célpontot:

```
<a name="term107">PHP</a>
```

Ez az elem vizuálisan nem fog megjelenni (pl. aláhúzással), hiszen ez a kapcsolat végpontja lehet, és nem a kezdőpontja.

Ha erre a pontra akarunk hivatkozni egy linkkel, akkor a következő módon kell alkalmaznunk:

```
<a href="http://weblabor.hu/szojegyzek#term107">
```

Természetesen akár az oldalon belül is lehet ilyen linkeket alkalmazni:

```
<a href="#tipp">Ugrás</a>
```

Hasznos tippek

Ha egy alkönyvtárra akarunk hivatkozni, az URL végére tegyük ki a / karaktert. Így a webkiszolgáló egy felesleges próbálkozást megspórolva gyorsabban fogja megtalálni a keresett könyvtár tartalmát. (Először a könyvtárnevet állománynévként próbálja értelmezni, ha nincs mögötte / karakter.)

Hosszú oldal esetén tegyünk az elejére egy tartalomjegyzéket a fontosabb fejezetekre mutató belső linkekkel. Ezen kívül szokás az oldal végén (vagy akár több helyen is) az oldal tetejére (elejére) mutató linket alkalmazni (erre a *href="#"* használható). Ennek a megoldásnak az a hátránya, hogy a vissza gomb hatására is az oldalon belül fog a felhasználó ugrálni.

Feladat

A korábban elkészített verseket tartalmazó oldalt bővítse ki a következőkkel:

- A költő neve alá szúrjon be egy „tartalomjegyzéket", amely az összes vers címét tartalmazza, és egyben oldalon belüli linkként is működik
- Minden vers után helyezzen el egy *tartalomjegyzék* feliratú linket, amivel a tartalomjegyzékre ugorhatunk vissza

CSS feladat

- Készítsen oldalt, amely belső és külső hivatkozásokat is tartalmaz. A külső hivatkozásokat jelöljük meg az ilyenkor szokásos háttérképpel! (Lehetőleg a képet is rajzoljuk!)

Ma találtam rá egy szuper oldalra, amely Webes szabványok címet viseli. Ha ez az oldal korábban indult volna, valószínűleg bele se kezdek a Web programozás jegyzet írásába :-)

3.8. Keretek

A keretek segítségével egynél több HTML oldalt is meg tudunk jeleníteni egyetlen böngésző ablakban.

A keretek néhány évvel ezelőttig nagy népszerűségnek örvendtek. Segítségükkel pusztán kliens oldali eszközökkel (HTML, CSS) összetett oldalakat lehet létrehozni. Mára azonban már egyre kevesebb helyen indokolt, és egyre kevesebb helyen is használják. (Érdemes belegondolni, hogy nagy portálok szinte egyáltalán nem alkalmazzák.) A teljesség igénye nélkül néhány hátrány, mielőtt belekezdenénk a keretek használatába:

- A fejlesztőnek több dokumentumot kell karbantartani
- Nagyon nehéz kinyomtatni egy keretes oldalt
- Nem lehet könyvjelzőt rakni egy állapothoz, hiszen az URL-ben csak a főoldal címe szerepel
- Az előzőhöz hasonlóan nem lehet egy oldalra hivatkozni
- Ha – esztétikai okokból – letiltjuk a keret görgetősávját, a tervezettnél kisebb méretű megjelenítés esetén a keret kilógó részei lényegében elérhetetlenek lesznek

A sort lehetne folytatni a szerver oldali programozás nehézségeivel.

A *frameset* tag

A <*frameset*> tag határozza meg, hogy az ablakot hogyan akarjuk felosztani keretekre. Egy *frameset* esetén vagy vízszintes, vagy függőleges irányban oszthatjuk fel az ablakot.

Megjegyzés: *frameset* esetén nincs szükség a *body* elemre.

A *frame* tag

A <*frame*> tag azt határozza meg, hogy egy-egy ablakrész milyen HTML állományt jelenítsen meg.

A következő példa két oszlopra bontja az ablakot. Az első keret a böngésző ablak 25%-át, a második 75%-át fogja elfoglalni. A *src* tulajdonságokban megadott oldalak lesznek az oldal keretei.

```
<frameset cols="25%,75%">
   <frame src="frame_a.htm">
   <frame src="frame_b.htm">
</frameset>
```

A százalékos megadás helyett pixeles mennyiség is alkalmazható, és – legfeljebb egyszer – használhatjuk a * karaktert a fennmaradó hely kitöltéséhez.

A *noframes* tag

Ha igazán korrektek akarunk lenni, akkor alkalmazhatjuk a <*noframes*> tagot, ami akkor jut szerephez, ha a megjelenítő eszköz nem tudja kezelni a kereteket. Itt azonban nem a – korábban elterjedt – „töltse le az XY böngészőt” típusú üzenetekre érdemes gondolni, hanem az oldal keretek nélkül működő változatára. A szerző tapasztalata és tudomása szerint eddig nagyon kevés oldal vette a fáradtságot, hogy a keretes változat mellett keretek nélkül is tökéletes funkcionalitással bíró változatot készítsen. Akik mégis vették a fáradtságot, hamarosan rájöttek, hogy a látogatók zöme a keret nélküli változatot részesíti előnyben, így a kérdés eldőlt.

Nézzünk egy minimális példát az alkalmazására:

```
<html>
  <frameset cols="25%,50%,25%">
    <frame src="frame_a.htm">
    <frame src="frame_b.htm">
    <frame src="frame_c.htm">

    <noframes>
    <body>Ez a böngésző nem jeleníti meg a kereteket!</body>
    </noframes>

  </frameset>
</html>
```

Az *iframe* tag

Az *iframe* tag dobozszerű keretet definiál. Ezt a tagot is többnyire érdemes elkerülni. Igen speciális esetekben engedhető meg.

Hasznos tippek

Ha a keretnek van látható szegélye, akkor a felhasználó vonszolással át tudja méretezni a kereteket. Ha ezt szeretnénk elkerülni, adjuk meg a *frame* tagnak *noresize="noresize"* tulajdonságot.

Egymásba ágyazás

A keretek egymásba ágyazásának segítségével akár összetettebb szerkezetek is kialakíthatók. A következő példában a *frame_a.htm* tartalma az ablak felső felét foglalja el, az alsó felében pedig vízszintesen 25%:75% arányban felosztva a másik két oldal osztozik:

```
<html>
  <frameset rows="50%,50%">
    <frame src="frame_a.htm">
    <frameset cols="25%,75%">
      <frame src="frame_b.htm">
      <frame src="frame_c.htm">
    </frameset>
  </frameset>
</html>
```

Navigációs keret

Navigációs keret alkalmazása esetén a link segítségével cserélni kívánt keretet névvel kell ellátni, hogy a link hivatkozása egyértelmű lehessen. (E nélkül a link a saját keretét cserélné ki.)

A főoldal kódjában a *name* tulajdonsággal adhatunk nevet a jobb oldali keretnek:

```
<html>
  <frameset cols="120,*">
    <frame src="tryhtml_contents.htm">
    <frame src="frame_a.htm" name="showframe">
  </frameset>
</html>
```

A bal oldali navigációs keretben a linkek *target* tulajdonságában kell ezt a keretet megadni:

```
<a href ="frame_a.htm" target ="showframe">Frame a</a><br>
<a href ="frame_b.htm" target ="showframe">Frame b</a><br>
<a href ="frame_c.htm" target ="showframe">Frame c</a>
```

Ügyes megoldás az is, ha a navigációs keret HTML kódjában megadjuk a linkek esetén alapértelmezett cél keret nevét a *head* elem *base* tagjának *href* tulajdonságával. Az előző példa így módosulna:

```
<html>
  <head><base href="showframe"></head>
<a href ="frame_a.htm">Frame a</a><br>
<a href ="frame_b.htm">Frame b</a><br>
<a href ="frame_c.htm">Frame c</a>
```

Összefoglaló táblázat

Tag	Leírás
<frame>	keretet definiál
<frameset>	keret-csoportot definiál
<noframes>	a keretet nem ismerő böngésző esetén megjelenő tartalom

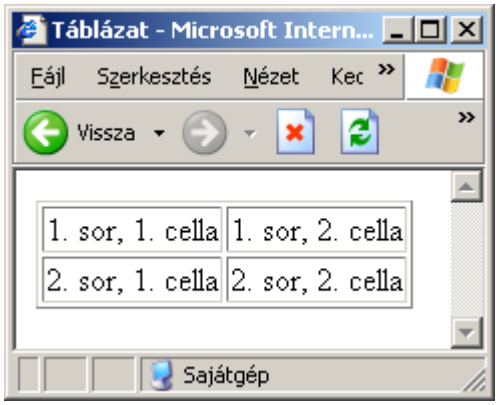
<iframe> soron belüli keretet definiál

3.9. Táblázatok

Táblázatokat a *<table>* tag segítségével lehet létrehozni. Egy tábla sorokat tartalmaz (*<tr>* tag), és minden sor cellákat (*<td>* tag). A tábla cellái szöveget, képet, bekezdést, listát, űrlapokat, újabb táblázatokat is tartalmazhatnak.

Nézzünk egy egyszerű, 2•2 cellás táblázatot:

```
<table border="1">
  <tr>
    <td>1. sor, 1. cella</td>
    <td>1. sor, 2. cella </td>
  </tr>
  <tr>
    <td>2. sor, 1. cella </td>
    <td>2. sor, 2. cella </td>
  </tr>
</table>
```



3.14. ábra: Táblázat

Táblázat szegélyek

Alapértelmezetten a táblázatok szegélyek nélkül jelennek meg. Van azonban lehetőség arra, hogy szegélyek is megjelenjenek az oldalon: az előző példában is látható *border* tulajdonsággal lehet beállítani a szegély szélességét. (A szám képpontban értendő.)

Táblázat fejléc

A táblázat első sorába szokás fejléc információkat elhelyezni. Ez magyarázza az alatta található értékek jelentését. Ebben az esetben az első sort celláit *<th>* tagokkal kell megadni:

```
<table border="1">
  <tr>
    <th>1. fejléc</th>
    <th>2. fejléc</th>
  </tr>
  <tr>
    <td>1. sor, 1. cella</td>
    <td>1. sor, 2. cella</td>
  </tr>
  <tr>
    <td>2. sor, 1. cella</td>
    <td>2. sor, 2. cella</td>
  </tr>
</table>
```

3.15. ábra: Táblázat fejléccel

Természetesen fejléc információt az első oszlopban is lehet alkalmazni, ekkor a sorok első celláját kell *th* taggal megadni.

Üres cellák

A tartalom nélküli (üres) cellák a legtöbb böngésző esetén nem jelennek meg. Például az utolsó cella tartalmának törlése esetén a következőt láthatjuk:

3.16. ábra: Üres cella

Szokás ilyenkor egy nem törhető szóközt (* *) tenni a cellába, ami kikényszeríti a szegély megjelenítését.

Táblázat cím

Méltatlanul keveset használt elem a *<caption>*, amivel a táblázat címét tudjuk korrekt módon megadni. Lehetőség van annak kiválasztására is, hogy a négy lehetséges oldal közül hol jelenjen meg a cím. A *caption* elemnek a *table* első elemének kell lennie, ha használjuk:

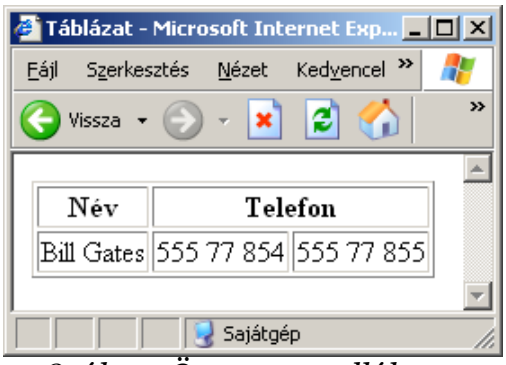
```
<table border="1">
  <caption>Táblázat címe</caption>
  <tr>
    <th>1. fejléc</th>
    <th>2. fejléc</th>
  </tr>
```

3.17. ábra: Táblázat címmel

Cellák összevonása

A *colspan* és *rowspan* tulajdonságok segítségével cellákat lehet egyesíteni:

```
<table border="1">
  <tr>
    <th>Név</th>
    <th colspan="2">Telefon</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>555 77 854</td>
    <td>555 77 855</td>
  </tr>
</table>
```



3.18. ábra: Összevont cellák

Tippek

A *<thead>*,*<tbody>* és *<tfoot>* elemek a céljuk szerint nagyon hasznos elemek lennének, de sajnos a böngészők igen változó módon támogatják őket, ezért a gyakorlatban nem is szokás alkalmazni.

A táblázat cellák további táblázatokat is tartalmazhatnak, amivel összetett szerkezetek alakíthatók ki.

Összefoglaló táblázat

Tag	Leírás
<table>	táblázatot definiál
<th>	fejlécet definiál
<tr>	sort definiál
<td>	cellát definiál
<caption>	címet definiál
<thead>	fejlécet definiál
<tbody>	táblázat törzset definiál
<tfoot>	láblécet definiál

Feladat

Készítsen HTML oldalt, amelyik a lehető leginkább hasonlít a következő oldalra:

Alapfok

Készség	Feladat típusa	Feladatok száma	Idő	Pontszám
A - szóbeli				
Beszédkészség	Személyes beszélgetés	5-6 kérdés	3 perc	45
	Beszélgetés kép(ek) alapján	5-6 gondolati egység	5-6 perc	
	Szituációs feladat		3-4 perc	
Hallás utáni értés	Jegyzetkészítés	5 címszó	kb. 20 perc	25
	Igaz-hamis - választásos feladat	10 állítás		20
B - írásbeli				
Íráskészség	Lyukas szöveg kiegészítése adott egységekkel	15 inforációs egység	15 perc	15
	Irányított fogalmazás (levél)*	4 szempont	75 perc	30
Olv. szöveg értése	Kérdésekre válaszadás*	10 inforációs egység		30
	Információközvetítés magyarul*	5 inforációs egység		15

CSS feladat

- Készítsen egy egyszerűbb táblázatos oldalt.
- A táblázat kinézetét próbálja minél inkább a mai szokásoknak megfelelően szerkeszteni, pl. zebracsíkos háttérrel.

TABLE DESIGNS

Design Name	Author	Country	Comment	Download
TagBox	TagBox	Deutschland	Table design based on the fresh TagBox style.	Download
Maniac Merchants	Marten Willberg	Germany	A Georgia headline with a green-blue body.	Download
Acuity Design Style	Acuity Internet Marketing	Brazil	Black Style, with smooth grey and green elements.	Download

3.10. Listák

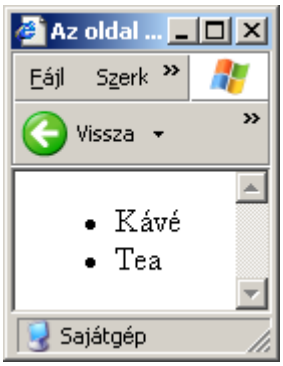
A HTML támogatja a számozott, felsorolt és definíció-listák létrehozását.

Felsorolt lista

A felsorolt listák olyan elemeket tartalmaznak, amelyeket nem kell számozással azonosítanunk, ehelyett egy felsorolási szimbólum (alapértelmezetten egy fekete karika) jelzi vizuálisan a listaelemek kezdetét.

A felsorolt lista az *<ul>* elemmel írható le, a lista elem pedig *<li>* elemmel.

```
<ul>
  <li>Kávé</li>
  <li>Tea</li>
</ul>
```



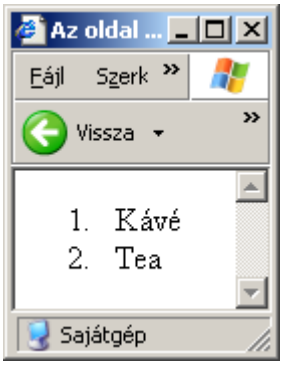
3.19. ábra: Lista

Számozott lista

A számozott listák elemei (többnyire) számmal azonosítottak.

A számozott listát *<ol>* taggal kell létrehozni, a lista elemek az előzőhöz hasonlóan *li*-vel definiálhatók.

```
<ol>
  <li>Kávé</li>
  <li>Tea</li>
</ol>
```



3.20. ábra: Lista

Definíciós lista

A definíciós lista nem csupán egy felsorolás, hanem a definiált elemek és a definícióik sorozata. Egy definícióhoz akár több elem is megadható.

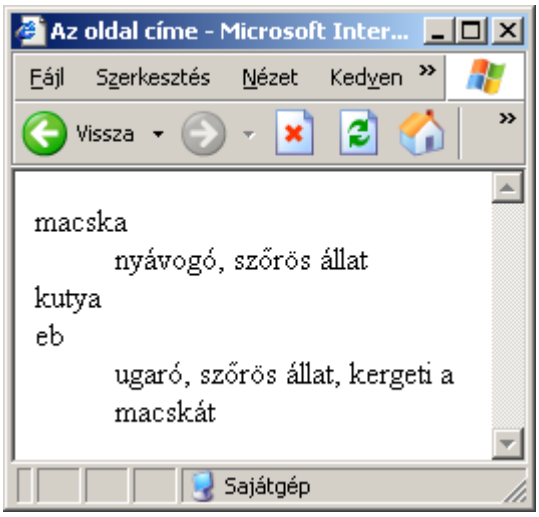
A definíciós lista <*dl*> taggal, a definiált elem <*dt*> taggal, maga a definíció pedig <*dd*> taggal definiálható.

```
<dl>

  <dt>macska</dt>
  <dd>nyávogó, szőrös
    állat</dd>

  <dt>kutya</dt>
  <dt>eb</dt>
  <dd>ugaró, szőrös állat,
    kergeti a macskát</dd>

</dl>
```



3.21. ábra: Definíciók

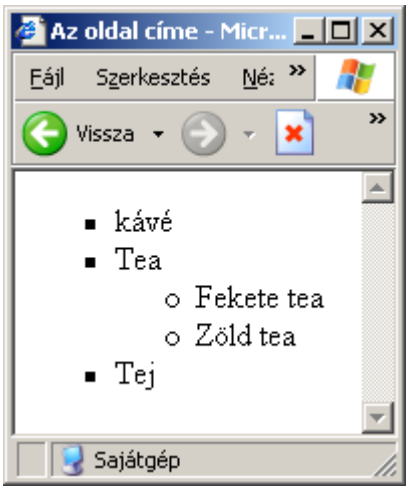
További példák

Felsorolt és számozott listák esetén a szimbólumok választhatók. A következő lista az angol abc kisbetűit írja ki:

```
<ol type="a">
 <li>alma</li>
 <li>banán</li>
 <li>citrom</li>
</ol>
```

A listák egymásba is ágyazhatók, így hierarchikus szerkezeteket is kialakíthatunk.

```
<ul type="square">
  <li>kávé</li>
  <li>Tea
    <ul>
      <li>Fekete tea</li>
      <li>Zöld tea</li>
    </ul>
  </li>
  <li>Tej</li>
</ul>
```



3.22. ábra: Egymásba ágyazás

Összefoglaló táblázat

Tag	Leírás
* *	számozott listát definiál
* *	felsorolást definiál
* *	lista elemet definiál
<dl>	definíciós listát definiál
<dt>	definiált elem
<dd>	definíció

Feladat

- Készítsen HTML oldalt, amely egy számozott listába ágyazott felsorolt listákkal demonstrálja a felsorolt listák összes lehetőségét.

CSS feladatok

- Készítsen egyedi kinézetet listákhoz! (Pl. használjon egyedi szimbólumot, háttér-képként beillesztve.)
- Készítsen vízszintes és függőleges elrendezésű menüket!

3.11. Űrlapok

Az űrlapokat arra használhatjuk, hogy különböző módokon lehetőséget adjunk a látogatónak visszajelzésre, vagyis adatok megadására.

A <*form*> elem más elemeket tartalmaz. Ezek az űrlap elemek teszik lehetővé az adatbevitelt.

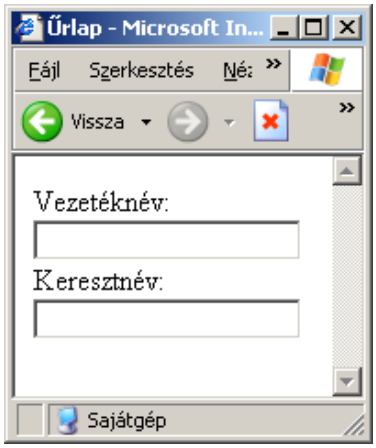
Az *input* tag

A leggyakrabban használt elem az <*input*>. A *type* tulajdonságával állítható be, hogy pontosan milyen adatbeviteli módot szeretnénk.

Szöveges mezők

A szöveges mezők lehetővé teszik, hogy betűkből, számokból, írásjelekből álló karaktersorozatot lehessen begépelni.

```
<form>
  Vezetéknév:
  <input type="text" name="vezeteknev">
  <br>
  Keresztnév:
  <input type="text" name="keresztnev">
</form>
```



3.23. ábra: Szöveges mezők

Érdemes megfigyelni, hogy maga a *form* elem vizuálisan nem látható, csak a benne elhelyezett elemek. Ha (ahogy ebben a példában is) nem adjuk meg a mezők szélességét, a legtöbb böngésző alapértelmezetten 20 karakter szélesen jeleníti meg. Ez azonban nem korlátozza a ténylegesen begépelhető szöveg hosszát.

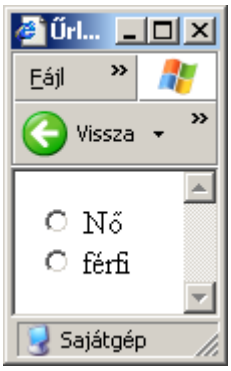
Jelszavak begépeléséhez *password* típusú (*type="password"*) beviteli mezőt szokás létrehozni. Ez viselkedésében csak annyiban tér el a *text* típustól, hogy a begépelt szöveg helyett * karakterek jelennek meg.

Megjegyzés: Az elrejtés csak a képernyőre vonatkozik, a hálózaton egyszerű adatként utazik a jelszó.

Rádiógombok

A rádiógombokat akkor használhatjuk, ha a látogatónak néhány választható elem közül kell választási lehetőséget adni. Az elemek közül mindig csak az egyik lehet aktív. Érdemes megfigyelni a következő listában, hogy a *name* tulajdonság azonossága rendeli a rádiógombokat logikailag egy csoporttá, vagyis egymást kizáró választási lehetőségekké. (Tehát ebből a szempontból sem a vizuális elrendezés számít!)

```
<form>
  <input type="radio" name="nem"
    value="no">Nő</input>
  <br>
  <input type="radio" name="nem"
    value="ferfi">férfi</input>
</form>
```



3.24. ábra: Rádiógombok

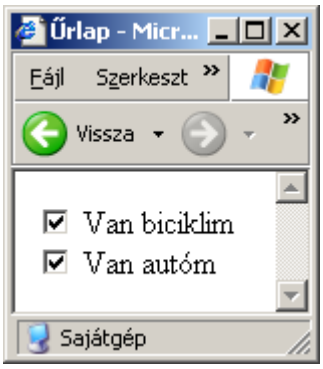
Ha valamelyik választási lehetőséget alapértelmezettnek tekintjük, akkor ezzel könnyíthetjük is az űrlap kitöltését:

```
  <input type="radio" name="nem" value="no"
    checked="checked">Nő</input>
```

Jelölőnégyzetek

A jelölőnégyzetek arra szolgálnak, hogy a felhasználó egy vagy több elemet is ki tudjon választani a rendelkezésre álló lehetőségek közül. Más megközelítésben úgy is lehetne fogalmazni, hogy egy jelölőnégyzet ki-, vagy bekapcsolt állapotban lehet, függetlenül más beviteli elemektől.

```
<form>
  <input type="checkbox" name="bicikli">
  Van biciklim
  <br>
  <input type="checkbox" name="auto">
  Van autóm
</form>
```



3.25. ábra: Jelölőnégyzetek

Itt is van lehetőségünk az alapértelmezetten ki nem választott állapot helyett bejelölve megjeleníteni a jelölőnégyzetet:

```
<input type="checkbox" checked="checked" name="bicikli">
```

A *label* elem

Érdemes megfigyelni, hogy rádiógomb és jelölőnégyzet esetén a kattintható terület a kör illetve négyzet alakú területre korlátozódik. Az elemek mellett megjelenő szövegek a böngésző számára függetlenek a jelölő elemektől, csupán a vizuális helyzet jelzi nekünk az összefüggést.

A *label* elem használatával ez a függetlenség megszüntethető: a szöveget a jelölőelemmel aktív kapcsolatba hozhatjuk. Ez azt jelenti, hogy lehet kattintani a szövegre is.

```
<form>
  <input type="radio" name="nem" value="no" id="no">
  <label for="no">Nő</label>
  <br>
  <input type="radio" name="nem" value="ferfi" id="ferfi">
  <label for="ferfi">férfi</label>
</form>
```

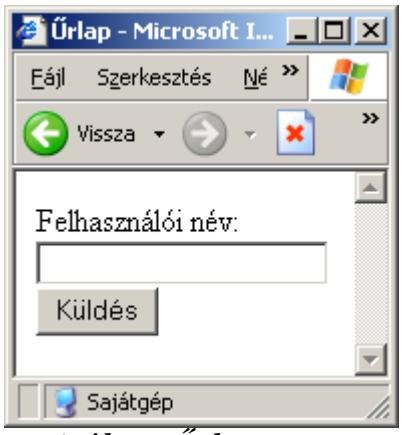
Az előző verzióhoz képest fontos kiegészítés, hogy a *value* mellett az *id* is megkapta az azonosító szövegeket, mivel a *label* tag *for* tulajdonsága az *id* alapján azonosítja az elemeket.

Űrlap adatok elküldése

Az esetek jelentős részében a felhasználó azért tölt ki egy űrlapot, hogy adatokat tudjon a szerver felé küldeni valamilyen hatás érdekében.

Az eddigi példákból két fontos rész kimaradt. Először is a felhasználó számára szokás biztosítani egy küldés (vagy valami hasonló feliratú) gombot, hogy erre kattintva kezdeményezhesse az adatok elküldését. Másrészt a *form* tag – egyébként kötelezően kitöltendő – *action* tulajdonsága határozza meg, hogy melyik oldalnak kell a kérést feldolgoznia.

```
<form name="input"
    action="feldolgoz.php"
    method="get">
  Felhasználói név:
  <input type="text" name="nev">
  <input type="submit" value="Küldés">
</form>
```



3.26. ábra: Űrlap

Ha a felhasználó begépeli a nevet, majd kattint a *Küldés* gombra, akkor a szöveg továbbításra kerül a *feldolgoz.php* számára.

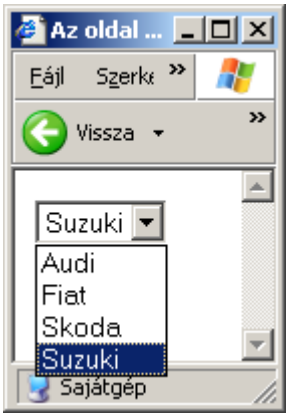
Nyomógombokat a *button* taggal is létre lehet hozni.

Lenyíló lista

Bizonyos esetekben rádiógombok helyett célszerű inkább a lenyíló listák alkalmazása. (Itt elsősorban terjedelmi, áttekinthetőségi okokra kell gondolni.)

Két tag használatára lesz szükség: először is a *select* tag adja meg a lista kereteit, míg az *option* tagok a választható elemeket.

```
<form>
  <select name="autok">
    <option value="audi">Audi
    <option value="fiat">Fiat
    <option value="skoda">Skoda
    <option value="suzuki"
       selected="selected">Suzuki
  </select>
</form>
```



3.27. ábra: Lista

A *selected* tulajdonság lehetővé teszi az alapértelmezett elem kijelölését. Ha ez nem szerepel a forrásban, akkor az első elem lesz a kiválasztott betöltődéskor.

Lehetőség van olyan lista létrehozására is, ahol egyszerre akár több elem is kiválasztható.

Több soros szöveges mezők

Lehetőség van hosszabb szöveg begépelését, szerkesztését lehetővé tevő beviteli mezőt is létrehozni. Erre szolgál a *textarea* elem. A következő példán a méretek megadásán túl a kezdőszöveg is definiált:

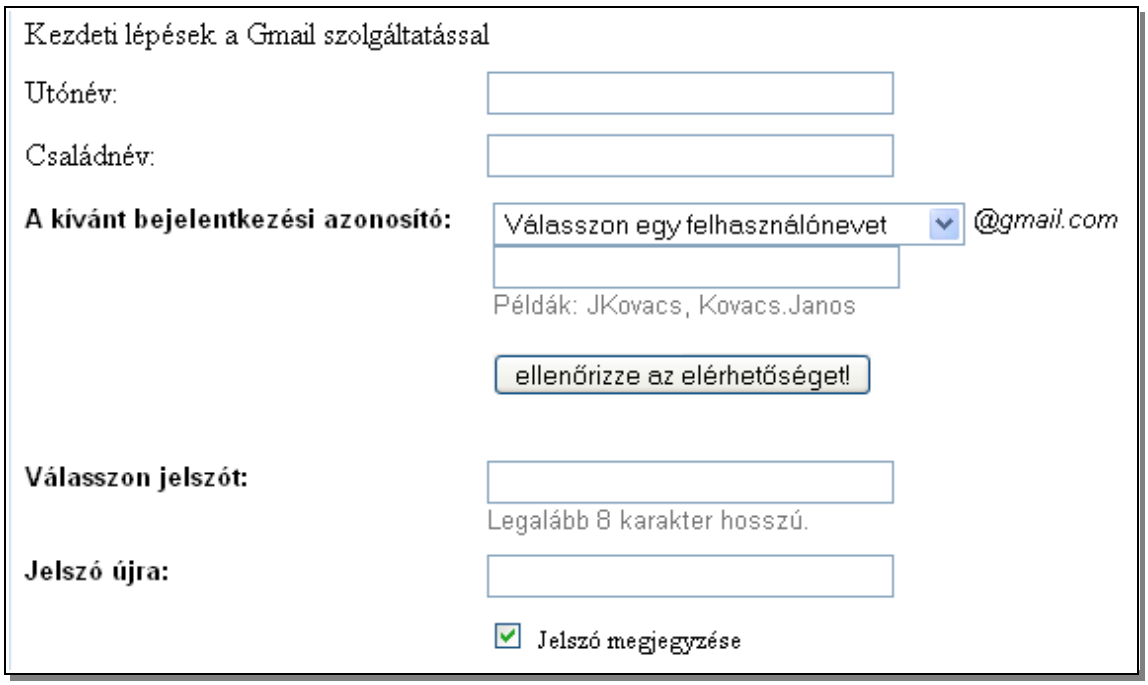
```
<textarea rows="10" cols="30">Kezdőszöveg</textarea>
```

Összefoglaló táblázat

Tag	Leírás
<*form*>	űrlapot definiál
<*input*>	beviteli mezőt definiál
<*textarea*>	többsoros beviteli mezőt definiál
<*label*>	címkét definiál
<*select*>	lenyíló listát definiál
<*option*>	lenyíló lista elemet definiál
<*button*>	nyomógombot definiál

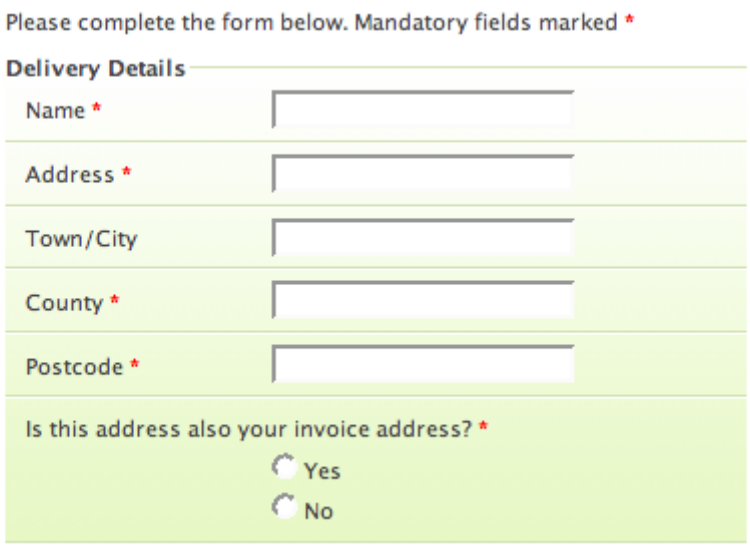
Feladat

Készítsen egy oldalt, ami a Google Mail szolgáltatásának regisztrációját utánozza.



CSS feladat

Készítsen az alábbihoz hasonló űrlapot!



További források

- http://www.webaim.org/techniques/forms/controls.php
- http://www.websemantics.co.uk/tutorials/accessible_forms/
- http://www.alistapart.com/articles/prettyaccessibleforms
- http://www.lukew.com/resources/articles/WebForms_LukeW.pdf

3.12. Képek

A HTML nyelvben az *<img>* tag segítségével tudunk képeket definiálni. Ez az elem üres, és nincs záró tagja sem (hasonlóan a *br* és *hr* elemekhez).

A kép megjelenítéséhez először is meg kell adni a *src* tulajdonságot, vagyis meg kell adni a kép állomány helyét és nevét. A szintaxis a következő:

```
<img src="url">
```

Az *url* lehet abszolút vagy relatív megadású is. Abszolút:

```
<img src="http://valami.hu/kep.gif">
```

Abszolút, a *base href*-ben megadott helytől, ennek hiányában a szerver gyökerében keres:

```
<img src="/kep.gif">
```

Relatív, a HTML könyvtárában keresi:

```
<img src="kep.gif">
```

Az *alt* tulajdonság

Az alt tulajdonság alternatív szöveg definiálását teszi lehetővé. A szabvány szerint tehát ennek a szövegnek akkor kell a böngészőben láthatóvá válni, ha a kép valamilyen oknál fogva nem jeleníthető meg (pl. még nem töltődött le, nem érhető el vagy eleve ki van kapcsolva a képek letöltése).

Megjegyzés: a Microsoft Internet Explorer akkor is megjeleníti ezt a szöveget, ha az egérkurzort visszük a kép fölé, de ez eltér a HTML eredeti céljától.

```
<img src="hajo.gif" alt="1 árbócos vitorlás hajó">
```

Méret megadása

A böngésző ugyan a méret megadása nélkül is meg tudja jeleníteni a képet, mégis célszerű a *width* és *height* tulajdonságokat megadni. Lassú kapcsolat vagy sok nagy képesetén kimondottan zavaró lehet, amikor egy újabb kép letöltődésekor – az ekkor ismertté vált méret adatok alapján – a félig megjelent oldal „ugrik”.

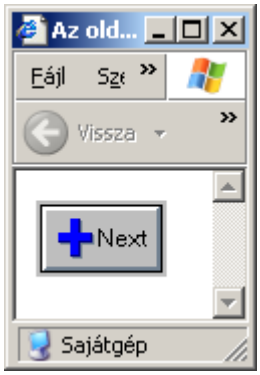
```
<img src="hajo.gif" width="144" height="50">
```

Természetesen a kép fizikai méretétől eltérő méretek is megadhatók, ekkor kicsinyítés, nagyítás, vagy akár torzítás is lehet az eredmény.

Kép használata linkként

Link aktív felületéhez szöveg mellett vagy helyett kép is rendelhető. Erre mutat példát a következő kód:

```
<p>
   <a href="lastpage.htm">
      <img border="0" src="kovetkezo.gif"
        width="65" height="38">
   </a>
</p>
```

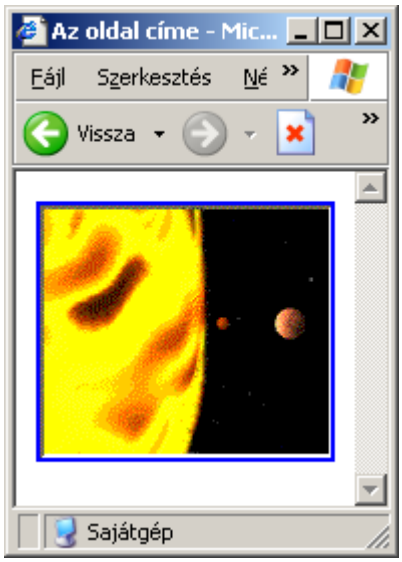


3.28. ábra: Kép linkként

Képtérkép

Még érdekesebb lehetőség a kliens oldali képtérképek alkalmazása. Ekkor a kép egyes területeihez más-más link kapcsolható. Példánkban a Naphoz, valamint a Merkúr és Vénusz bolygóhoz más-más linket rendelünk, míg a maradék terület nem reagál a kattintásra.

```
<img src="bolygok.gif"
  width="145" height="126"
  usemap="#bolygoterkep">
<map id="bolygoterkep"
  name="bolygoterkep">
  <area shape="rect"
    coords="0,0,82,126"
    alt="Nap" href="sun.htm">
  <area shape="circle"
    coords="90,58,3"
    alt="Merkúr" href="mercur.htm">
  <area shape="circle"
    coords="124,58,8"
    alt="Vénusz" href="venus.htm">
</map>
```



3.29. ábra: Képtérkép

Érdemes megfigyelni először is a *usemap* tulajdonság használatát. Ennek értéke köti össze a képet a területi felosztást leíró *map* elemmel. A *map* elem akárhány *area* elemet tartalmazva írhatja le a linkek jellemzőit.

Megjegyzés: Az előbb felvázolt lehetőséget kliens oldali képtérképnek nevezzük. Szerver oldali képtérkép is létezik, érdemes azonban a felhasználó kényelme érdekében a kliens oldali lehetőséget alkalmazni. (A szerver oldali megoldás hátránya, hogy a kép egyes részei közt a felhasználó nem tud különbséget tenni, a böngésző csak a koordinátákat küldi el a szerver számára.)

Hasznos tippek

Az *align* tulajdonság kép igazítását határozza meg. Ezt a tulajdonságot azonban a CSS lehetőségei miatt felesleges használni.

Az *img* elem helye határozza meg, hogy a megjelenített oldalon hol fog szerepelni. Például ha két bekezdés (p) elem között szerepel, akkor a bekezdésektől jól elkülöníthetően jelenik meg. Viszont ha a bekezdés belsejében van, akkor a bekezdés részeként a szöveggel akár körbefuttathatóvá is válik. (Maga a körbefuttatás szintén CSS-el végzendő.)

Összefoglaló táblázat

Tag	Leírás

<*img*>	kép elemet definiál
<*map*>	képtérképet definiál
<*area*>	területet definiál

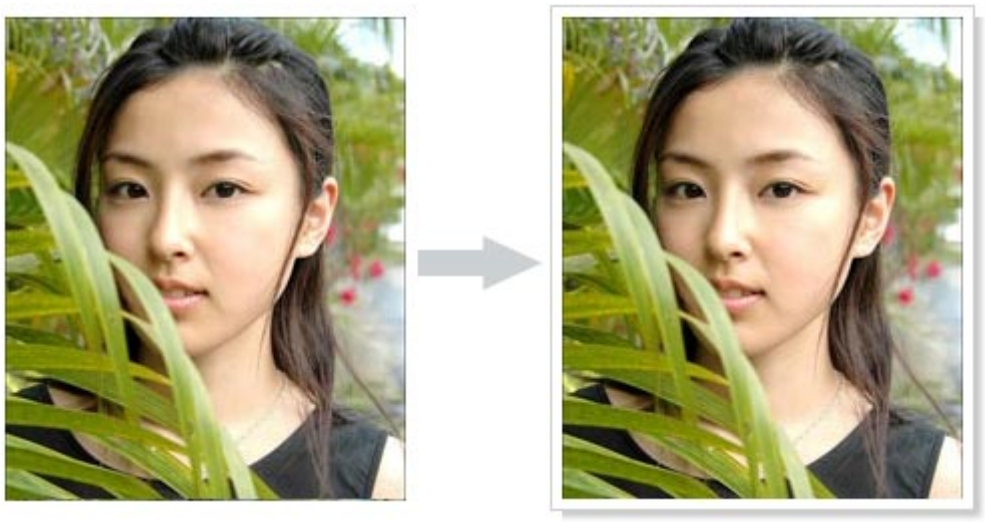
Feladat

Készítsen egy több oldalból álló képgaléria honlapot!

Az index oldal 3x4-es táblázatba szervezve tartalmazza a bélyegképeket, alattuk címmel. A bélyegképekre kattintva egy olyan oldalra jussunk, ahol a kép címe és a kép jelenik meg.

CSS feladat

- A képekhez készítsen árnyék hatást, szegéllyel.



3.13. Oldal háttere

A *body* tag rendelkezik két tulajdonsággal, amelyek az egész oldal hátterét határozzák meg.

A *bgcolor* tulajdonság az oldal háttérszínét határozza meg. A színt háromféle módon lehet megadni: hexadecimális számként, RGB értékként vagy szín-névvel:

```
<body bgcolor="#000000">
<body bgcolor="rgb(0,0,0)">
<body bgcolor="black">
```

Mindhárom példa feketére állítja az oldal háttérszínét.

A *background* tulajdonság az oldalhoz háttérképet határoz meg. A paraméter értéke lehet relatívan vagy abszolút módon megadott URL.

```
<body background="hatter.gif">
<body background="http://honlapom.hu/hatter.gif">
```

Hasznos tippek

Háttérkép és szöveg alkalmazása esetén figyelni kell arra, hogy a szöveg színe eléggé elüssön a háttérétől (elég nagy legyen a kontraszt). Háttérképként viszonylag homogén (közel egyszínű) képet érdemes alkalmazni, és a háttérszínt is hasonlóra beállítani, hogy a kép lassú letöltődése esetén se legyenek olvashatósági problémák.

CSS segítségével sokkal több és jobb lehetőségünk lesz a háttér beállítására, érdemes azt előnyben részesíteni.

3.14. Színek

A színeket többféle módon közelíthetjük meg. Itt most a monitorok képmegjelenítésénél használatos, éppen ezért a webfejlesztésben is általános RGB megközelítést alkalmazzuk.

Az *RGB* színek három összetevőből állnak, mint piros (*red*), zöld (*green*), és kék (*blue*). Mindhárom érték egy bájton tárolt előjel nélküli számként határozható meg, vagyis értéke 0 és 255 között (hexában 0 és FF között) lehet. Ezzel a módszerrel tehát a 3 bájton ábrázolható 16 millió szín közül bármelyik kiválasztható.

Ezen kívül 16 alapszín névvel is rendelkezik (*aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white,* és *yellow*).

3.15. Szabványosság

A HTML nyelv a kezdetektől fogva szigorú szabályokra épült. A *Microsoft Internet Explorer* és a *Netscape Navigator*[31] harcának egyik melléktermeke, hogy a böngészők felismernek, értelmeznek olyan HTML oldalakat is, amelyek nem felelnek meg a szabványnak. Sőt a webfejlesztők ezekre a pontatlanságokra rá is szoktak, és ami az informatikaoktatás egyik szégyenfoltja: sok „szakíró” is úgy használja a szabványtól elrugaszkodott téves gyakorlatokat, mintha a szabványok létéről sem hallott volna még.

A HTML nyelv minden verziója, változata egy úgynevezett *Document Type Definition* (DTD) segítségével definiált. A böngészők számára segítség, ha a dokumentum elején pontosan leírjuk, hogy melyik verzióhoz tartjuk magunkat.

A HTML 4.01 változata[32] 3 féle DTD-vel érvényesíthető.

A legpontosabb (*strict*) változat nem engedélyezi az elavult részek (többnyire formázó elemek és tulajdonságok) használatát, valamint a kereteket. A következő elemet kell a dokumentum első elemeként elhelyezni:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Az átmeneti (*transitional)* DTD megengedi az elavult részek használatát is.

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

[31] A Netscape 2007 utolsó heteiben – hosszú haldoklás után – befejezte pályáját.

[32] A HTML 5-ös verziójának kidolgozása 2007 végén előrelépett ugyan, de még mindig nem jutott el a hivatalos ajánlásig, ami után egyáltalán számon kérhető lenne a böngészők gyártóitól, és egyáltalán alkalmazható lenne a gyakorlati fejlesztésben.

Keretek használata esetén a következő DTD-t kell megjelölni:

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

Az oldalunk HTML szabványosságának ellenőrzésére több lehetőségünk van.

- *http://validator.w3.org/* online használata
- Firefox böngészőbe beépülő Tidy HTML Validator[33]

Feladat

A fejezet eddigi példáit ellenőrizze, és szükség esetén korrigálja a W3C validátorát felhasználva.

3.16. Stílusok

A HTML 4.01 minden formázási információt elkülönített stíluslapokra[34] bíz.

A stílus információkat háromféle módon rendelhetjük hozzá a HTML dokumentumunkhoz.

Külső stíluslap

A külső stíluslap alkalmazása a legideálisabb eset. Ekkor a HTML állomány az oldal informatív részér tartalmazza, a külső CSS állomány pedig összegyűjtve a megjelenítésre vonatkozó formázásokat.

A HTML oldalhoz a *head* elemben elhelyezett <*link*> tag segítségével csatolhatunk külső CSS állományt:

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

Más megközelítés, de szintén alkalmazható:

```
<head>
  <style type="text/css">
    @import url("stilus.css");
  </style>
  ...
</head>
```

Külső stíluslap fájlt nem HTML oldalanként külön-külön, hanem akár egy egész portálhoz egyetlen (vagy néhány) fájlként szokás készíteni. Így a további oldalak látogatása esetén nincs szükség a CSS fájl újbóli letöltésére.

[33] http://users.skynet.be/mgueury/mozilla/

[34] Ma a legfontosabb, de nem az egyetlen stílusdefiníciós lehetőség a CSS. Elvileg más megoldások is vannak, pl. DSSSL, XSL.

Beágyazott stíluslap

Beágyazott stíluslapot kész oldalnál akkor szokás alkalmazni, ha az oldal egyedi (a külső stíluslapot nem lehetne másik oldalhoz felhasználni), vagy nagyon egyszerű stílusról van szó. Persze a dizájn kialakításának fázisában is jó megoldás lehet.

Beágyazott stíluslapot a *head* elemben szereplő <*style*> tag segítségével definiálhatunk.

```
<head>
  <style type="text/css">
    body {background-color: red}
    p {margin-left: 20px}
  </style>
  ...
</head>
```

Soron belüli stílus

Soron belüli stílust esetleg akkor szokás alkalmazni, ha olyan egyedi stílusról van szó, amelyik sehol máshol nem fordul elő.

Megjegyzés: A szerző véleménye szerint még ilyen esetekben sem érdemes ezt alkalmazni, a CSS alapos megismerésével és ügyes szervezéssel ezt el lehet kerülni.

Soron belüli stílus alkalmazásához a kiválasztott elemet kell *style* tulajdonsággal ellátni. A következő példa csak e bekezdés megjelenítését változtatja meg:

```
<p style="color: red; margin-left: 20px">Ez egy bekezdés</p>
```

A *div* és *span* tagok

Ha az ajánlásokhoz ragaszkodunk, és tényleg minden formázást külső CSS fájl segítségével adunk meg, akkor bizonyos formázó tagok helyett más tagokat kell alkalmaznunk. Nem mindig van szükség rájuk, hiszen az egyébként már szereplő tagok is formázhatók.

A *div* és *span* tagok célja az, hogy rá lehessen akasztani valamilyen CSS formázást, ha nincs más alkalmas tag (vagy az túl erőltetett lenne) a HTML kódban.

A két tag között egyetlen különbség, hogy a *div* blokk szintű (mint pl. a *p* vagy *table*), míg a *span* soron belüli (mint pl. a *strong* vagy *a*) elem.

Összefoglaló táblázat

Tag	Leírás
<*style*>	stílus definíciót hoz létre
<*link*>	külső stíluslapot kapcsol az oldalhoz
<*div*>	blokk szintű elemet definiál
<*span*>	soron belüli elemet definiál

3.17. Fejrész

A *head* elem olyan információkat tartalmaz, amelyek a HTML dokumentum egészére vonatkoznak. A fejrész a <*base*>, <*link*>, <*meta*>, <*title*>, <*style*> és <*script*> tagokat tartalmazhatja.

A *base* elem az oldal relatív linkjeinek működésére lesz hatással. Ha nincs a *base* elem *href* tulajdonsága megjelölve, akkor a relatív hivatkozások az aktuális HTML fájl könyvtárához képest lesznek értelmezve. A *href* megadása esetén a megadott URL lesz az indulási hely.

A *meta*[35] elem meta-információt definiál az oldalhoz. Néhány példa:

```
<meta name="keywords" content="HTML, CSS, XHTML, JavaScript" />
```

A kereső robotok számára fontos kulcsszavakat emeli ki.

```
<meta name="description" content="Web programozás jegyzet" />
```

Rövid összefoglaló az oldaltól.

```
<meta name="revised" content="Hege Refsnes, 6/10/99" />
```

Az utolsó módosítás napja.

Összefoglaló táblázat

Tag	Leírás
<*head*>	a dokumentum egészére vonatkozó információkat definiál
<*title*>	a böngésző címsorában megjelenő szöveget definiálja
<*base*>	relatív linkek számára bázis hivatkozás
<*link*>	külső fájlok kapcsolása
<*meta*>	meta-információkat definiál

3.18. Szkriptek

Kliens oldali szkriptek segítségével az oldal interaktívabbá tehető.

A <*script*> elem segítségével tudunk a HTML oldalba szkripteket elhelyezni. Ennek *type* tulajdonságában kell megadni, hogy milyen szkriptnyelvet szeretnénk használni.

Megjegyzés: A JavaScript egyetlen korábbi említésre méltó vetélytársa a Visual Basic Script volt. A szerző véleménye szerint a sokféle, egymással sem kompatibilis Basic rendszerével a Microsoft öngólt rúgott.

[35] A keresőoldalak egy korábbi generációja lényegében a *meta* tagokra építette a keresési szolgáltatását. Ezt a fejlesztők egy idő után arra használták, hogy hamis képet mutassanak magukról, és ezzel jobb helyet érjenek el a találati listákon. A Google nagy ötlete volt, hogy nem erre a könnyen manipulálható információra épít, így ma a *meta* tagok jelentősége ma jóval kisebb, mint ahogy azt a régi források hirdetik.

```
<html>
  <body>
    <script type="text/javascript">
      document.write("Hello Világ!")
    </script>
  </body>
</html>
```

Az oldalt böngészőben megjelenítve a *Hello Világ!* szöveg jelenik meg a képernyőn.

A *noscript* tag

Ha a böngésző valamilyen ok miatt nem tudja végrehajtani a *script* elem tartalmát (pl. a JavaScript ki van kapcsolva), akkor a *noscript* elem tartalma fog megjelenni.

```
<script type="text/javascript">
  document.write("A böngésző támogatja a JavaScriptet.")
</script>
<noscript>
  A JavaScript nem támogatott.
</noscript>
```

Általában nem javasolt a használata. (Az okokról a JavaScript kapcsán lesz szó.)

Az *object* és *param* tagok

Az <*object*> tag beágyazott objektumot definiál. Az *object* tipikusan multimédia tartalom beágyazására használható, de a W3C célja szerint az *applet* és *img* elemek kiváltására is alkalmas.

A következő példa egy *pelda.mov* állomány lejátszását kezdeményezi.

```
<object data="pelda.mov" type="video/quicktime"
    title="példavideó" width="200" height="200">
  <param name="pluginspage" value="http://quicktime.apple.com">
  <param name="autoplay" value="true">
</object>
```

A *param* tag segítségével futásidejű paramétereket adhatunk át a beágyazott objektumnak.

Az *object* tagok akár egymásba is ágyazhatók, a belső *object* tag akkor lesz figyelembe véve, ha a külső *object* nem futtatható. (Pl. letiltott *Flash* objektum esetén egy állókép jelenjen meg.

Összefoglaló táblázat

Tag	Leírás
<*script*>	szkriptet definiál
<*noscript*>	ha nem fut le a szkript, ennek tartalma jelenik meg
<*object*>	beágyazott objektumot definiál
<*param*>	beágyazott objektum számára paramétert definiál

3.19. Általános tulajdonságok

A HTML tagoknak különböző tulajdonságaik lehetnek. Az eddig bemutatott tagok speciális tulajdonságairól már a korábbiakban olvashattunk. Ebben a fejezetben az általánosan használható tulajdonságok kerülnek a középpontba.

Alap tulajdonságok

Nem használhatók a *base, head, html, meta, param, script, style,* és *title* elemekben.

Tulajdonság	Leírás
class	az elem osztálya
id	az elem egyedi azonosítója
style	soron belüli stílus definíció
title	rövid segítség definiálása

Nyelvi tulajdonságok

Nem használhatók a *base, br, frame, frameset, hr, iframe, param,* és *script* elemekben.

Tulajdonság	Leírás
dir	az írás irányát definiálja
lang	a nyelvet adja meg

Billentyűzet tulajdonságok

Tulajdonság	Leírás
accesskey	gyorsbillentyűt rendel az elemhez
tabindex	tabulátorral történő elemváltás sorrendjét befolyásolja

3.20. Esemény tulajdonságok

A böngésző programok lehetőséget adnak arra, hogy különböző (többnyire felhasználói) események esetén szkriptek segítségével programozható tevékenységet lehessen definiálni. Például egy HTML elemen való kattintás hatására az oldal egy része megváltozik.

Ablak események

Egyedül a *body* és a *frameset* elemek esetén használhatók.

Tulajdonság	Leírás
onload	akkor fut le a szkript, ha az oldal betöltődött

onunload	akkor fut le a szkript, ha a felhasználó az oldalt bezárja, vagy más oldalra lép tovább

Űrlap elemek eseményei

Egyedül űrlap elemek esetén használhatók.

Tulajdonság	Leírás
onchange	akkor fut le a szkript, ha az elem megváltozott
onsubmit	akkor fut le a szkript, ha az űrlap elküldését kéri a felhasználó
onreset	akkor fut le a szkript, ha az űrlap kezdőállapotba áll
onselect	akkor fut le a szkript, ha az elemet kiválasztja a felhasználó
onblur	akkor fut le a szkript, ha az elem elveszti a fókuszt
onfocus	akkor fut le a szkript, ha az elem megkapja a fókuszt

Billentyűzet események

Nem használható a *base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style,* és *title* elemekben.

Tulajdonság	Leírás
onkeydown	akkor fut le a szkript, ha egy billentyűt lenyomott a felhasználó
onkeypress	akkor fut le a szkript, ha egy billentyűt lenyomott és felengedett a felhasználó
onkeyup	akkor fut le a szkript, ha egy billentyűt felengedett a felhasználó

Egér események

Nem használható a *base, bdo, br, frame, frameset, head, html, iframe, meta, param, script, style,* és *title* elemekben.

Tulajdonság	Leírás
onclick	akkor fut le a szkript, ha egérklikkelés történt
ondblclick	akkor fut le a szkript, ha dupla egérklikkelés történt
onmousedown	akkor fut le a szkript, ha az egérgombot lenyomta a felhasználó
onmousemove	akkor fut le a szkript, ha a felhasználó az adott elem felett mozgatja a kurzort
onmouseout	akkor fut le a szkript, ha a felhasználó az adott elemről elmozgatja

	a kurzort
onmouseover	akkor fut le a szkript, ha a felhasználó az adott elemre mozgatja a kurzort
onmouseup	akkor fut le a szkript, ha az egérgombot felengedi a felhasználó

3.21. URL-kódolás

Az URL egy erőforrások megcímzésére alkalmas eszköz kötött karakterkészlettel. Bizonyos karakterek (pl. ékezetes betűk, egyes írásjelek) közvetlenül nem szerepelhetnek az URL-ben, de kódolt formában már igen.

A kódolt karakterek egy % jellel és a két számjeggyel leírt hexadecimális értékükkel írhatók le. Például a szóköz kódolva *%20*.

Űrlapok adatainak GET metódussal való elküldése esetén az űrlap adatok az URL-ben kerülnek továbbításra. Ilyen esetben is fontos szerepet kap a kódolás, hiszen egy begépelt űrlapmezőben bármilyen karakter előfordulhat.

3.22. Szemantikus HTML

A szemantikus HTML nem más, mint a HTML tagok jelentésének betartása, rendeltetésszerű használata. Például, egy bekezdést nem használunk fel menü kialakításához.

Azért fontos ez, mert a keresőgépek, automatikus katalógusok, HTML értelmezők ez alapján tudják kategorizálni az oldal tartalmait. Például egy felolvasóprogram egy címsort ki tud emelni felolvasás közben is, vagy egy keresőgép tud idézetekre keresni egy bizonyos szerzőtől, ha jól meg van formázva a HTML.

A gyakrabban előforduló hibákkal szemben néhány jó megközelítés:

- A címsorokat a <*h**> elemekkel adjuk meg, úgy ahogy a tartalom strukturálódik; a <h1>-el kezdjük, ez az oldal címe, majd jönnek a továbbiak
- Az oldalnak egyedi címet adunk a fejlécben (*title*): az egész webhely címe + oldal címe).
- A bekezdések szövegeit <*p*> elemekbe tagoljuk, az új bekezdésnél nem csak a kurzort visszük új sorba egy <*br*> elemmel, hanem valóban jelezzük a bekezdés végét és egy új kezdetét.
- A különböző listákat a lista elemekkel definiáljuk. (<*ul*>, <*ol*>, <*dl*>). Talán nem triviális, de a menüket is *ul* elemmel érdemes megadni. (A menüpontok is egyfajta listát jelentenek.)

3.23. Ellenőrző kérdések

Melyik a helyes HTML dokumentum felépítés?

a. *<html><head>...</head><title>Cím</title><body>A dokumentum*

szövegtörzse</body></html>

b. *<html><head><title>Cím</title></head><body>A dokumentum szövegtörzse</body></html>*

c. *<html><head>...</head><title>Cím</title></html><body>A dokumentum szövegtörzse</body>*

Hogyan jelenik meg a következő szövegrész?

```
<p>Ez itt
egy próba              szöveg </p>
```

a. Ez itt egy próba szöveg

b. Ez itt
egy próba szöveg

c. Ez itt
egy próba
szöveg

Melyik a legnagyobb méretű fejléc?

a. *h1*

b. *h6*

c. *h7*

Melyik tag segítségével készíthetünk sorszámozott listát?

a. *<dl>*

b. *<ol>*

c. *<ul>*

d. *<list>*

Melyik tag segítségével készíthetünk egyszerű felsorolás listát?

a. *<ul>*

b. *<dl>*

c. *<ol>*

d. *<list>*

Melyik hiperhivatkozás formája megfelelő,

ha egy weboldalra mutató linket szeretnénk létrehozni ?

a. *<a>http://www.gamf.hu*

b. *GAMF Honlap *

c. *GAMF Honlap *

d. *GAMF Honlap *

Hogyan tehetünk úgy linkké egy e-mail címet,

hogy rákattintva az alapértelmezett levelezőprogram induljon el?

a. * szöveg *

b. *<mail>gipszjakab@nemtudom.hol.hu</mail>*

c. * szöveg *

d. *<mail href="gipszjakab@nemtudom.hol.hu"></mail>*

Melyik tagok szerepelhetnek táblázat megadásánál?

a. *<table><tr><tt>*

b. *<thead><dt><tr>*

c. *<table><tr><td>*

d. *<table><head><caption>*

Mi történik, ha egy táblázatban nem állítunk be szélességeket?

a. automatikusan beállítja a böngésző a cellák tartalma szerint

b. hibát okoz

c. a lap teljes szélességével lesz egyenlő a táblázat szélessége

Melyik megoldás használható kép beillesztésére?

a. *<image src="image.gif">*

b. *<img href="image.gif>*

c. *<img src="image.gif">*

d. *<img scr="image.gif">*

Mi a szerepe az alt paraméternek?

Pl.

```
<img src="kids.jpg" alt="gyerekek">
```

a. ezzel adhatjuk meg a böngészőnek, hogy először egy kisebb méretű képet töltsön be.

b. ezzel adhatunk nevet a képünknek, és később ezzel a névvel hivatkozhatunk rá

c. a képek letöltésére nem alkalmas böngészőknek kínál szöveges magyarázatot.

Melyik helyes?

A *value* paraméter

a. csak számok esetén értelmezhető, csak számformátum elfogadását teszi lehetővé

b. a helyes érték/szöveg azonosítására használják

c. a megjelenítéskor látható alaphelyzet beállítására, aktív gombok feliratozására, alapértékek kiválasztására szolgál

Ha egyszerű egy soros szöveget szeretnék bekérni

a szöveges űrlapelemet melyik módon írhatom helyesen?

a *<input type="text">*

b *<input type="textarea">*

c *<textinput>*

A *submit* vagy *reset* típusú gombokon

hogyan tüntethetek fel tetszőleges szöveget?

a. *<input type="reset"> szöveg</input>*

b. *<input type="submit" szöveg>*

c. *<input type="submit" value="szöveg">*

Melyik a helyes forma?

a. *<input type="checkbox">.....*

b. *<checkbox> .....</checkbox>*

c. *<input type="check">*

Melyik helyes?

a. *<textarea>.....</textarea>*

b. *<input type="textarea">*

c. *<input type="text" option="area">*

Milyen módon érhetem el,

hogy a magyarul írt ékezetes betűket a használt böngésző meg tudja jeleníteni?

a. *<meta http-equiv="Content-Type" lang="hu">*

b. *<meta name="description" content="hu">*

c. *<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">*

A következő sornak mi lesz a hatása?

```
<meta http-equiv="Refresh" content="5;url=http://www.gamf.hu/">
```

a. semmi, ez csak információ a keresőprogramok számára

b. kiírja az utolsó frissítés dátumát

c. 5 másodperc múlva automatikusan betölti a megadott url-ről a dokumentumot

3.24. Feladatok

Oldal elemzés

Keressen meg egy tetszőleges portált (pl. Index, Origo), és a Web Developer segítségével kapcsoljon ki minden Javascriptet és CSS-t. Vizsgálja meg, hogyan épül fel az oldal HTML forrása.

Oldal reprodukció

Keressen meg egy másik portált, a fentihez hasonlóan állítsa elő az oldal tisztán HTML változatát, majd a forráskód megnézése nélkül próbálja reprodukálni az oldalt.

3.25. További források

A HTML alaposabb megismeréséhez, fejlesztés közbeni napi használathoz a legalkalmasabb a HTMLinfo HTML referenciáját alkalmazni. Címe:

http://htmlinfo.polyhistor.hu/html_ref/main.htm

4. XML

Ebben az alfejezetben röviden áttekintjük az XML alapjait, hogy az XHTML újdonságait megérthessük.

Mi az XML?

- *eXtensible Markup Language*
- a HTML-hez hasonló jelölőnyelv
- adatok leírására szolgál
- nincsenek előre definiált tagok, azok minden egyes XML alkalmazás esetén egyedileg definiálhatók
- a nyelvtani szabályai *Document Type Definition* (DTD) vagy *XML Schema* segítségével írhatók le
- az XML formátum önleíró
- a W3C ajánlása (jelenleg 1.0)

Az XML nem a HTML helyett jött létre, hiszen az XML céljaiban sokkal nagyobb területet fed le. Az XML tetszőleges célú szöveges adatok leírásával, míg a HTML a weben megjelenítendő tartalommal foglalkozik.

4.1. Mire használjuk az XML-t?

A HTML webre szánt tartalmak leírására szolgál. Ehhez képest az XML tetszőleges más esetekben is alkalmazható. Tulajdonképpen bárhol, ahol adatok struktúrált leírására, továbbítására, átadására van szükség.

Adatcsere

Mivel az XML teljesen platform és alkalmazás-független módon írja le az adatokat, az első komoly elterjedési területe a **heterogén rendszerek közötti adatcsere** megvalósítása volt.

Példa: Képzeljünk el egy olyan vállalatot, amelyik önálló informatikai infrastruktúrával, vállalatirányítási rendszerrel rendelkező kisebb cégeket kebelez be. A bekebelezés folyamatának végcélja lehet a bekebelezett cég információs rendszerének teljes integrációja, de ez sokszor eléggé nehézkes lehet, és nem is biztos, hogy célszerű. Sokszor célravezetőbb megoldás lehet, ha „csupán” a rendszerek közötti aktív adatcserét teremtjük meg. Erre pedig a legegyszerűbb megoldás, ha definiálunk egy XML nyelvtant az adatcsere definiálására, és a két rendszerhez csupán egy XML csatolófelületet adunk hozzá. Így a két rendszer költséges újraírása helyett jóval kisebb költséggel megoldható az integráció.

Természetesen integrációs törekvések más esetben is előfordulhatnak. Jellemző eset még az is, hogy egy vállalaton belül hosszú évek vagy évtizedek óta hibátlanul működő rendszerek egymástól elkülönülten működnek, de a vállalatvezetés új igényei szükségessé teszik, hogy a független rendszerek közötti adatcsere megvalósulhasson.

B2B

Egy másik, mára nagyon fontossá váló terület a **B2B** (*Business To Business*) kapcsolatok megvalósítása. Ma minden komoly kereskedő cégnek van webes megjelenése, de egyre nagyobb az igény az üzleti partnerek közötti szorosabb kommunikáció megvalósu-

lásának is. Itt az XML arra használható, hogy az üzleti partnereket (természetesen nem emberi, hanem automatizált, szoftveres megoldásokkal) minél gyorsabban megtaláljuk. A kommunikáció nyelve itt is az XML lesz.

Példa: Képzeljünk el egy könyvesboltot, amelyik nem csak egy, hanem több nagykereskedővel áll kapcsolatban, és így akár ugyanazt a könyvet is más-más nagykereskedőtől más-más áron, szállítási határidővel és egyéb feltételekkel lehet megszerezni. Tegyük fel, hogy a nagykereskedők lehetővé teszik a vásárlóik (a könyvesboltok) számára, hogy XML alapon (például egy webszolgáltatás formájában) bármikor le tudják kérdezni a rendelhető mennyiséget, és a további paramétereket. Ekkor egy megfelelően elkészített alkalmazás képes lehet arra, hogy a könyvesbolt alkalmazottjának nagyvonalú instrukciói alapján megtalálja az ideális beszerzési módot és a szállítót, majd a megrendelést szintén az XML alkalmazásával leadja.

Ez akár teljes emberi munkafolyamatokat is kiválthat, például a népszerű könyvek pillanatnyi eladásai alapján látszik, hogy 2-3 napon belül egy könyv készlete ki fog merülni, így a szoftver automatikusan olyan szállítót keres, amelyik 2 napon belül vállalja a leszállítást (és esetleg a korábbi szállítási tapasztalatok alapján ez valószínűleg meg is fog történni). Lehet, hogy lenne valamivel olcsóbb szállító is, de ha csak egy hét határidővel vállalja a szállítást, akkor nem az kapja a megrendelést.

A példák sorát sokáig lehetne folytatni, ennyi azonban elegendő ahhoz, hogy lássuk: az XML alapú adatcserében hatalmas lehetőségek vannak.

Adatok megosztása

Az XML-ben tárolt adatok hardver-, és szoftver független adatmegosztást tesznek lehetővé.

Sokkal könnyebbé teszi olyan adatok létrehozását, amelyekkel különböző alkalmazások is dolgozni tudnak.

Az XML-t használhatjuk arra is, hogy adatokat tároljunk fájlokban vagy adatbázisokban. Alkalmazásokat lehet írni hogy elmentsen és visszaolvasson információkat az adatbázisból, és általános alkalmazásokat használhatunk arra, hogy adatokat jelenítsen meg.

Más ügyfelek és alkalmazások hozzá tudnak férni az XML fájljainkhoz, mint adatforráshoz.

Saját nyelv létrehozása

Az XML arra lett kitalálva, hogy saját nyelveket definiálhassunk általa. Saját, egyedi alkalmazásunkhoz saját, a céljainknak megfelelő nyelveket hozhatunk létre.

Ilyen értelemben az XML-t talán pontosabb lenne „nyelv” helyett „meta-nyelvnek” nevezni.

4.2. XML szintaxis

Az XML szintaxisa nagyon egyszerű, és nagyon szigorú. Könnyen megtanulható, és könnyen használható.

Nézzünk egy egyszerű példát, egy jegyzettömb bejegyzést:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Az első sor a dokumentum formátumát adja meg: a (jelenleg egyetlen létező) verziószámot, és a karakterkódolást. (Érdemes belegondolni, hogy ez a sor a 7-bites ASCII kódolással mindig leírható, a további sorok olvasásánál már ismert lesz a karakterkódolás.

A <*note*> és </*note*> tagok határolják az egész dokumentumot kitöltő gyökér elemet. A közbülső négy sor a bejegyzés adatait tartalmazza.

Jól formázott dokumentumok

XML dokumentumról akkor beszélhetünk, ha minimum jól formázott (*well formed*):

- Ahogy a példában is látszik, minden tagnak van záró párja.
- A tagok kisbetű-nagybetű érzékenyek, és az egymásba ágyazás is korrekt.
- Az XML dokumentumnak egy gyökér eleme van (a példában *note*).
- A tulajdonság értékei mindig idézőjelek között szerepelnek, és mindig megadottak.

Az elválasztó (*white space*) karaktereknek lehet szerepe (az XML alkalmazástól függ). Itt tehát – eltérően a HTML-től – a több elválasztó karakter nem ugyanazt jelenti, mintha csak egyet tartalmazna.

Érvényes dokumentum

Az esetek egy részében az érvényesség (*validation*) egy további kritérium. (Érvényes dokumentum csak jól formázott lehet.) Az egyik (és a régebbi, többször alkalmazott) lehetőség a DTD (*Document Type Definition*) segítségével való érvényesítés. A következő XML dokumentum egy külső DTD állományra hivatkozik:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "InternalNote.dtd">
<note>
...
```

Nagyon röviden megfogalmazva a DTD azt tartalmazza, hogy a dokumentumban milyen tagok szerepelhetnek, milyen módon ágyazhatók egymásba, milyen módon ismétlődhetnek, milyen tulajdonságaik lehetnek a tagoknak, ezekből melyek kötelezőek, stb.

A következő példa nem külső állományban, hanem helyben definiálja a DTD-t, és mindössze a lehetséges tagokkal, valamint a gyökérelem nevével foglalkozik:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

5. XHTML

Az XHTML (*EXtensible HyperText Markup Language* – kiterjeszthető HTML) a HTML pontosabb és „tisztább” verziója. A HTML 4.01 leváltására[36] hozták létre. Az XHTML az XML egy alkalmazása, vagyis XML eszközökkel is feldolgozható. Ugyanakkor a csak HTML-t ismerő böngészőknek és egyéb programokat sem kell kidobni, hiszen az XHTML lényegében felülről kompatibilis a HTML 4.01-el. Az XHTML tehát a HTML-re és XML-re egyaránt épít.

A W3C az XHTML 1.0-ás ajánlását 2000-ben tette közzé. Jelenleg legfrissebb verzió az 1.1-es, és dolgoznak a 2-esen.

5.1. XHTML a gyakorlathoz képest

Az XHTML igazából nem sok mindenben tér el a szabványos HTML nyelvtől. A weben látható oldalak többségéhez képest nagyobb az eltérés, hiszen a legtöbb weblap eleve nem szabványos.

Egymásba ágyazás

Az elemek csak megfelelően egymásba ágyazva és mindig lezárva lehetnek.

A következő példában az egymásba ágyazással van probléma:

```
<b><i>Ez egy félkövér és dőlt szöveg</b></i>
```

A helyes megoldás:

```
<b><i>Ez egy félkövér és dőlt szöveg</i></b>
```

A tagok neveit kisbetűvel kell írni

A tagok neveit kisbetűvel kell írni, ezért a következő hibás:

```
<BODY>
  <P>Ez egy bekezdés.</P>
</BODY>
```

A helyes változat:

```
<body>
  <p>Ez egy bekezdés.</p>
</body>
```

Elemek lezárása

A HTML elemeket le kell zárni. Hibás:

```
<p>Ez egy bekezdés.
<p>Ez egy másik bekezdés.
```

A helyes:

```
<p>Ez egy bekezdés.</p>
<p>Ez egy másik bekezdés.</p>
```

[36] A szakma a mai napig is megosztott. Sokakat nem zavar, hogy 5-10 éves szabványokat, és ráadásul rosszul használnak. Egyesek a HTML 5-ös, mások az XHTML irányában látják a kiutat.

Záró pár nélküli elemek

Záró pár nélküli elemek esetén is le kell zárni. Itt azonban egy rövidített írásmód is alkalmazható:

```
Sortörés<br />
Elválasztó vonal<hr />
Kép <img src="happy.gif" alt="Happy face" />
```

Megjegyzés: a / előtti szóköz csak a visszafele kompatibilitás miatt kell, az XHTML nem igényli. (Ez azt jelenti, hogy a csak HTML-t ismerő böngészők is meg tudják jeleníteni az oldalt.)

A tulajdonság-értékeket mindig idézőjelbe kell tenni

Ez praktikus volt HTML-nél is, ha a tulajdonság értéke pl. szóközt tartalmazott. XHTML-ben azonban mindig kötelező:

```
<table width="100%">
```

A tulajdonságok kötelezően tartalmaznak értéket is

Ha a HTML nem írt elő értéket (vagyis a tulajdonság lényegében logikai jelzőként működik), akkor az érték megegyezik a tulajdonság nevével.

```
<input checked="checked" />
<input readonly="readonly" />
<input disabled="disabled" />
<option selected="selected" />
<frame noresize="noresize" />
```

A *name* tulajdonság helyett az *id* használandó

Hibás tehát:

```
<img src="picture.gif" name="kep" />
```

Helyes:

```
<img src="picture.gif" id="kep" />
```

Megjegyzés: A régi böngészőkkel való kompatibilitás miatt mindkettőt is szokták alkalmazni:

```
<img src="picture.gif" id="kep" name="kep" />
```

5.2. Dokumentumtípus-deklaráció

Bár a HTML is előírja a DTD meghatározását, ezt alig néhány oldalon használják ténylegesen. Ezen kívül a gyökér elem névterét is meg szokás határozni.

```
<!DOCTYPE dokumentum-típus>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Cím</title>
  </head>
  <body>
    Az oldal törzse
   </body>
</html>
```

A DTD pontosan meghatározza, hogy milyen szintaktikai szabályoknak felel meg a dokumentum.

5.2.1 XHTML 1.0 DTD-k

Transitional

Ez a típus a legmegengedőbb, legszabadabb, csak azért hozták létre, hogy egy kisebb, közbülső lépést tudjanak tenni a régi szokásokhoz képest.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Frameset

Egy másik ideiglenes változat kereteket használó oldalakhoz.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Strict

A legszigorúbb változat, lényegében az újabb verziók ezt viszik tovább.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

5.2.2 XHTML 1.1 DTD

A jelenleg használható legfrissebb verzió. Itt már csak egyféle verzió van:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Ebből is következik, hogy az 1.1-es verzió nem támogatja a keretek használatát.

5.3. Visszafelé kompatibilitás

Attól, hogy webfejlesztőként törekszünk a szabványok betartására, a látogatók egy része még akár évekig régebbi böngészőt fog alkalmazni. Érdemes tehát arra is figyelni, hogy az oldalaink (lehetőleg) a csak HTML-t ismerő böngészők által is értelmezhetők legyenek.

Megjegyzés: A HTML nyelv sikerének egyik oka az volt, hogy a régebbi böngészők általában minden gond nélkül értelmezték az esetleg újabb verziójú HTML oldalakat is, legfeljebb az újabb tagokat és tulajdonságokat nem vették figyelembe. Az XHTML-re való váltás ennél valamivel több figyelmet igényel.

A következő felsorolás összefoglalja a lényegesebb információkat:

- önálló tag esetén szóköznek kell a / jelet megelőzni
- önálló tag nem mindig alkalmazható (pl. bekezdés esetén nem)

- külső stíluslapot, szkript fájlt kell alkalmazni, ha nem használható karaktereket tartalmaz (pl. <, két -)
- *lang* és *xml:lang* is megadható
- A karakterkódolást elsősorban a HTTP *content-type* megadásával határozzuk meg. Ha ez nem megoldható, akkor mind az XML deklarációban, mind *meta* elemként érdemes megadni.

5.4. Feladat

Keressen néhány weboldalt, amely XHTML 1.1 szerint nem szabványos, és alakítsa át azokat szabványossá. (Használja a Firefox *HTML Validator* kiegészítőjét!)

5.5. További források

Magyar nyelven a HTMLInfo oldalán[37] olvasható a W3C ajánlásának fordítása.

Ugyanitt megtalálható az XML igen alapos definíciója[38] is.

[37] http://htmlinfo.polyhistor.hu/xhtml1se/cover.html

[38] http://htmlinfo.polyhistor.hu/xml_ref/rec-xml.html

6. CSS

6.1. Bevezetés

Mi a CSS?

- A CSS a *Cascading Style Sheets* rövidítése
- A stílusok a HTML megjelenítési elemei és attribútumai helyett használhatók, azoknál jóval több lehetőséget biztosítva
- A stílusok meghatározzák, hogy hogyan jelenjenek meg vizuálisan a HTML elemek
- A stíluslapok segítségével könnyen szét lehet választani az oldal tartalmát annak kinézetétől (a dizájntól)
- A stílusokat általában külön állományban tároljuk (*.css* kiterjesztéssel)
- Külső stíluslapokkal gyorsítani tudjuk a munkavégzést
- Több stílus is hatással lehet egy elem megjelenésére

Kedvcsináló

Mielőtt a CSS alapos megismeréséhez kezdenénk, mindenképpen célszerű a *CSS Zen Garden*[39] oldalát meglátogatni. Ez az oldal azzal a céllal jött létre, hogy a CSS-el szembeni ellenérzéseket lerombolja, és bemutassa, milyen óriási lehetőségek vannak a dizájner kezében, ha a CSS-t komolyan és szakszerűen használja. Ezen az oldalon ugyanazt a HTML oldalt láthatjuk sok-sok különböző dizájnná formálva – csupán a CSS állomány (és természetesen a dekorációs képek) cseréjével. A teljesség igénye nélkül néhány képernyőkép:

6.1. ábra: CSS Zen Garden példa

6.2. ábra: CSS Zen Garden példa

[39] Magyar oldal: http://www.csszengarden.com/tr/magyar/

6.3. ábra: CSS Zen Garden példa

6.4. ábra: CSS Zen Garden példa

A stílusok használatával megoldható nehézségek

A HTML tagok eredetileg arra lettek megalkotva, hogy a dokumentum tartalmát definiálják. Amikor egy címet, bekezdést vagy táblázatot akarunk létrehozni, akkor használhatjuk a *h1*, *p* és *table* tagokat. A tényleges megjelenítés a böngészőre van bízva, eldöntheti, hogy mit hogyan jelenítsen meg a tagok, az ablakméret, a felhasználó beállításai alapján.

Később a két nagy böngésző (a *Netscape Navigator* és az *Internet Explorer*) újabb és újabb HTML tagokat és tulajdonságokat adott a böngésző által felismert HTML nyelvhez (pl. a *font* tag, vagy a *color* tulajdonság). Így belekeveredtek a HTML nyelvbe a megjelenítést befolyásoló elemek. (Mellesleg a böngészők csak részben és később támogatták a vetélytárs újításait.)

A problémát a *World Wide Web Consortium*[40] (W3C), egy non-profit, szabványokat alkotó szervezet oldotta meg. A HTML 4.0-ás verziójával és a vele párhuzamosan fejlesztett CSS segítségével létrejött egy jól használható eszközpáros a webfejlesztők részére.

Mára minden jelentősebb böngésző támogatja a CSS-t, bár a támogatottság mértékében vannak eltérések.

Megjegyzés: A Microsoft (és több más, magát a saját területén monopolhelyzetben tudó cég) módszere, hogy a független szabványokat (a szerző véleménye szerint) tudatosan és szándékosan figyelmen kívül hagyja azt remélve, hogy a versenytársak helyzetét ezzel lehetetlenné teszi. Egészen a Firefox megjelenéséig és a felhasználók körében való viharos sebességű elterjedéséig ez a taktika sikeresnek látszott. A jövőt megjósolni azonban ezen a területen is nagyon nehéz.

A stíluslapokkal munkát spórolhatunk

A stíluslapok azt definiálják, hogy hogyan jelenjenek meg az egyes HTML elemek. A stíluslapokat külön *.css* kiterjesztésű állományban szokás elhelyezni. Így könnyedén lehet ugyanazt a megjelenítést adni a honlap összes oldalához, mindössze egyetlen CSS állomány szerkesztésével. Ha bármit változtatni kell a dizájnon, lényegében csak ezt az egyetlen oldalt kell módosítanunk.

[40] http://www.w3.org/

Egy elemre több stílusdefiníció is hatással van

A stílusok egy vagy több elemre, vagy akár az egész oldalra is hatással lehetnek (ez utóbbi a *body* elem formázásával). Megfordítva: egy elemre hatással lehet a soron belüli stílusa, a *head* elembeli formázás és akár külső CSS állomány is. Sőt egy HTML oldalhoz akár több külső CSS állományt is rendelhetünk, és egy CSS állományt is több különböző HTML állományhoz rendelhetünk.

Lépcsős elrendezés

Melyik stílus fog érvényesülni, ha több stílust is definiálunk ugyanahhoz a HTML elemhez?

A következő négy beállítás érvényesül egyre nagyobb prioritással (tehát ütközés esetén a későbbi felülírja az előzőt).

1. a böngésző alapbeállítása
2. külső stíluslap
3. *head* elemben definiált stílus
4. soron belüli stílus

Tehát a soron belüli stílus a legmagasabb prioritású, tehát felülír minden alacsonyabb szintű formázást.

6.2. A CSS nyelvtana

A nyelvtan három elemet különböztet meg: kiválasztó, tulajdonság és érték:

```
kiválasztó {tulajdonság: érték}
```

A kiválasztó legegyszerűbb esetben egy HTML tag, a tulajdonság azt határozza meg, hogy milyen jellemzőt akarunk módosítani, míg az érték a változást határozza meg. A tulajdonságot és az értéket egy kettősponttal kell egymástól elválasztani, és a kettőt együtt kapcsos zárójelbe tenni.

```
body {color: black}
```

Ha az érték több szóból áll, idézőjelbe kell tenni:

```
p {font-family: "sans serif"}
```

Ha egy kiválasztó esetén többféle tulajdonságot is módosítani szeretnénk, könnyedén megtehetjük, mindössze pontosvesszővel kell elválasztani a tulajdonság-érték párokat.

```
p {text-align:center; color:red}
```

A stílusdefiníciók jobb olvashatósága érdekében inkább több sorba érdemes tagolni a sort:

```
p {
  text-align: center;
  color: black;
  font-family: arial
}
```

Felsorolás

Egyszerre akár több kiválasztóra is érvényesíthetjük a formázást. Ekkor a kiválasztókat vesszővel elválasztott listaként kell felsorolni. A példában minden címet zölden szeretnénk megjeleníteni:

```
h1,h2,h3,h4,h5,h6 {
  color: green
}
```

Osztály kiválasztó

Osztály kiválasztó segítségével más-más módon tudjuk megjeleníteni az egyes osztályokba sorolt elemek tartalmát. A példában a két különböző osztályhoz tartozó bekezdések más-más formázást kapnak:

```
p.right {text-align: right}
p.center {text-align: center}
```

Ez a két stílus a következő két bekezdés megjelenésére hatással lesz:

```
<p class="right">
  Ez egy jobbra igazított bekezdés.
</p>
<p class="center">
  Ez egy középre igazított bekezdés.
</p>
```

A *p* HTML tagoknak nem kötelező megadni *class* tulajdonságot, vagy akár más is lehet a *class* értéke, de ezekben az esetekben a fenti stílusoknak nem lesz hatása a bekezdésekre.

Az osztály szintű kiválasztást nem kötelező taghoz kötni, lehet tagoktól független, általános osztály szelektort is definiálni. A példa minden *center* osztályú elemet középre igazít. (Már amelyik HTML elemnél egyáltalán van értelme a középre igazításnak.)

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

A formázás minden *center* osztályú tagot középre igazít:

```
<h1 class="center">
  Ez egy középre igazított cím.
</h1>
<p class="center">
  Ez egy középre igazított bekezdés.
</p>
```

Azonosító alapú kiválasztás

A HTML elemeknek megadhatjuk az egyedi *id* tulajdonságot. Így az egyedi *id*-vel rendelkező elemhez speciális formázást határozhatunk meg. CSS-ben a # segítségével tudunk elemet *id* alapján kiválasztani.

A következő példában a *menu* azonosítójú elem betűszínét zöldre állítjuk:

```
#menu {color: green}
```

A *para1* azonosítójú bekezdést középre igazítva és piros színnel definiáljuk:

```
p#para1 {
  text-align: center;
  color: red
}
```

Megjegyzés

A CSS fájlba azért szoktunk megjegyzéseket tenni, hogy a későbbi megértést és módosítást könnyebbé tegye. CSS megjegyzésként egyedül a C nyelvből ismert /*...*/megjegyzés használható:

```
/* Ez itt egy megjegyzés */
p {
  text-align: center;
/* Ez itt egy másik megjegyzés */
  color: black;
  font-family: arial
}
```

A CSS lépcsős formázása

Bár a gyakorlatban még nem terjedt el, nagyobb CSS állományok esetén az áttekinthetőséget jelentősen javítani tudja a CSS forrás (a C nyelven megszokotthoz hasonló) behúzása. Például:

```
body {...}
  p {...}
    p.balra {...}
  table {...}
    table#egyik td {...}
```

Természetesen ez a leírási mód nem egyértelmű, többféleképpen is lehet a logikai csoportosítást szervezni.

6.3. A *background* tulajdonságok

Egy elem hátterét háttérszín és háttérkép segítségével változatosan formázhatjuk. A háttérkép ismétlődhet is függőlegesen, vízszintesen, és lehet a böngészőablakhoz vagy a görgetett tartalomhoz ragasztott.

6.3.1 Háttérszín

A *background-color* tulajdonság segítségével meghatározhatjuk az elemek háttérszínét. A korábban megismert színmegadási módok közül itt is tetszőlegesen választhatunk. További lehetőség a *transparent* megadása, ami átlátszó háttért fog jelenteni.

```
body {background-color: yellow}
h1 {background-color: #00ff00}
h2 {background-color: transparent}
p {background-color: rgb(250,0,255)}
```

6.3.2 Háttérkép

Az egyszínű háttér helyett akár látványos háttérképet is elhelyezhetünk az elemek hátte-reként a *background-image* tulajdonság segítségével.

```
background-image: url('bgdesert.jpg')
```

Ismétlődés

A háttérkép alapértelmezés szerint kitapétázva jelenik meg, ha az elem mérete ezt szük-ségessé teszi. Természetesen ez megváltoztatható, mind vízszintes, mind függőleges irányban megtilthatjuk az ismétlődést. A következő példában csak *y* irányban ismétlődik a háttérkép:

```
background-image: url('bgdesert.jpg');
background-repeat: repeat-y;
```

A mindkét irányú ismétlődés kikapcsolásához a *no-repeat* értéket kell adnunk.

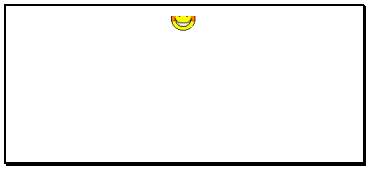
Pozíció

A háttérkép pozíciója is megváltoztatható: az alapértelmezett bal felső sarok helyett máshol is lehet a háttérkép.

Megjegyzés: Ennek még akkor is van jelentősége, ha a teljes hátteret kitapétázva látjuk, ugyanis a kezdő tapétát van értelme máshol és máshol megadni.

A következő példában az ablak felső részén, középen jelenik meg a smiley-nk.

```
background-image: url('smiley.gif');
background-repeat: no-repeat;
background-position: top center;
```



Az érték megadásnál először a függőleges, majd a vízszintes pozíciót kell megadnunk. A 3-3 konstans (*top, center, bottom* és *left, center, right*) mellett százalékos és pixeles po-zícionálás is lehetséges.

Megjegyzés: Még a kettő közül az egyiket is elhagyhatjuk, ha az egyetlen megadott érték egyértelművé teszi a fejlesztő szándékát.

Háttérkép ragasztva

Elsősorban hosszabb görgethető tartalom esetén van jelentősége annak, hogy a háttér-kép nem csak a görgetősávval együtt mozogva, hanem fixen pozicionálva is kérhető. Sok érdekes megoldás érhető el ezzel az apró trükkel.

```
background-image: url('smiley.gif');
background-repeat: no-repeat;
background-attachment: fixed;
```

Mindent bele

Ha többféle háttértulajdonságot is állítunk, tömörebb írásmódot eredményez az összevont *background* tulajdonság. Így egyszerre akár mindent is beállíthatunk.

```
background: #00ff00 url('smiley.gif') no-repeat fixed center;
```

6.4. Szövegek megjelenítése

A CSS szöveg tulajdonságai segítségével a szövegek vizuális megjelenítését lehet testre szabni.

A szöveg színe

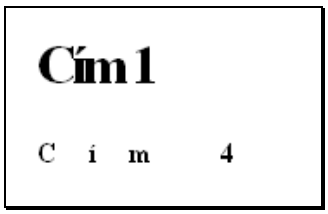
A szöveg színét a *color* tulajdonság határozza meg:

```
h1 {color: #00ff00}
h2 {color: #dda0dd}
p {color: rgb(0,0,255)}
```

Betűtávolság

A betűk közötti távolság a *letter-spacing* tulajdonsággal módosítható. A pozitív érték ritkítást eredményez, a negatív pedig sűrítést:

```
h1 {letter-spacing: -3px}
h4 {letter-spacing: 0.5cm}
```



Szótávolság

A betűtávolsághoz hasonló módon adható meg a *word-spacing* tulajdonság segítségével.

A szöveg igazítása

A következő példa bemutatja, hogyan lehet a szöveget balra, középre, jobbra vagy sorkizártan igazítani:

```
h1 {text-align: center}
h2 {text-align: left}
h3 {text-align: right}
p {text-align: justify}
```

Természetesen az igazítás meghatározásának csak blokk szintű elem esetén van értelme. Az alapértelmezett a balra igazítás.

Megjegyzés: A sorkizárt igazítással érdemes csínján bánni, mivel az automatikus elválasztás hiánya miatt a szóközök nagyon csúnyán megnyúlhatnak. Keskeny szövegblokk esetén különösen kerülendő.

A szöveg dekorációja

A következő példa bemutatja, hogy hogyan lehet a szövegünket fölé húzott, áthúzott, alá-húzott vonallal megjeleníteni:

```
h1 {text-decoration: overline}
h2 {text-decoration: line-through}
h3 {text-decoration: underline}
a {text-decoration: none}
```

A szöveg behúzása

A bekezdés első sorát a következő módon tudjuk 1 cm-el behúzni:

```
p {text-indent: 1cm}
```

Kis-, és nagybetű formázás

A következő példa a nagybetűs, kisbetűs, majd kis-kapitális formázást mutatja be:

```
p.uppercase {text-transform: uppercase}
p.lowercase {text-transform: lowercase}
p.capitalize {text-transform: capitalize}
```

Elválasztó karakterek értelmezése

Ahogy már a korábbiakban volt szó róla, a HTML oldalakon az ún. elválasztó karakterek (*white-spaces*) számától és típusától (szóköz, tabulátor vagy újsor) függetlenül mindig egy szóköznek számítanak. Ez alól az egyetlen kivétel a *pre* tag alkalmazásával érhető el.

Praktikusabb azonban, hogy az elválasztó karakterek értelmezését ennél finomabban tudjuk szabályozni a *white-space* tulajdonság segítségével. Az alapértelmezett kikapcsolt (*none*) mellett lehetőség van az előformázott értelmezés (*pre*) és a több sorra tördelést megtiltó (*nowrap*) beállításra.

Az eddigi lehetőségeken túl az írásirányt is beállíthatjuk a *direction* tulajdonsággal.

6.5. Betűk formázása

Betűtípus megadása

A következő példában a *h3* címeknek és a bekezdéseknek más-más betűtípust használunk.

```
h3 {font-family: times}
p {font-family: courier}
p.sansserif {font-family: sans-serif}
```

A betűtípusok közül érdemes általánosan használt, a képernyőn jól olvasható típusokat megadni. Kevésbé elterjedt betűtípus esetén érdemes elterjedtebb alternatívákat is felsorolni, hogy a böngésző nem ismert típus esetén is tudjon hasonlót választani.

Betűméret

A betűk méretét a *font-size* tulajdonsággal állíthatjuk be. A megadásnál kötelező mértékegységet is alkalmazni, és általában érdemes relatív megadási módot alkalmazni, hogy a felhasználó a saját igényei szerint tudja azt kicsinyíteni vagy nagyítani.

```
p {font-size: 1,1em}
h1 {font-size: 130%}
```

A számszerű megadáson túl szövegesen is megadhatjuk a méretet: *xx-small, x-small, small, medium, large, x-large, xx-large, smaller* és *larger*. Az így megadott méretek is relatívak.

Betűstílus

A dőlt betűstílus alkalmazására láthatunk egy példát:

```
p {font-style: italic}
```

Betűvastagság

A betűk vastagsága a *font-weight* tulajdonsággal befolyásolható. Szöveges konstansok (*normal, bold, bolder* és *lighter*) mellett 100, 200, ... 900 értékek használhatók.

```
p.normal {font-weight: normal}
p.vastagabb {font-weight: bold}
p.legvastagabb {font-weight: 900}
```

Betűformázások összevonása

Az összes betűfornázás összevonható akár egyetlen deklarációvá:

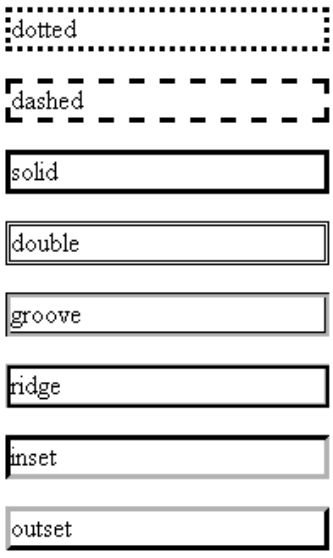
```
p {font: italic small-caps 900 12px arial}
```

6.6. Szegélyek

A CSS *border* tulajdonságaival szegélyeket rajzolhatunk az elemek köré. A következő példa az összes lehetséges vonalstílust bemutatja.

Megjegyzés: Nem minden böngésző az előírásoknak megfelelően rajzolja a szegélystílust.

```
p.dotted {border-style: dotted}
p.dashed {border-style: dashed}
p.solid  {border-style: solid}
p.double {border-style: double}
p.groove {border-style: groove}
p.ridge  {border-style: ridge}
p.inset  {border-style: inset}
p.outset {border-style: outset}
```



Szegélyszín

A szegélyek színét is beállíthatjuk a *border-color* tulajdonsággal.

```
p.egy {
  border-style: solid;
  border-color: #0000ff
}
p.ketto {
  border-style: solid;
  border-color: #ff0000 #0000ff
}
p.haron {
  border-style: solid;
  border-color: #ff0000 #00ff00 #0000ff
}
p.negy {
  border-style: solid;
  border-color: #ff0000 #00ff00 #0000ff rgb(250,0,255)
}
```

A szegély vastagsága

A szegély vastagságát a *border-width* tulajdonsággal állíthatjuk be. A következő példa folyamatos vonallal szegélyezi a bekezdést, de a szegély vastagságát megnöveli az alapértelmezett 1px-hez képest:

```
p {
  border-style: solid;
  border-width: 15px
}
```

A négy oldal szegélyeit nem csak egyszerre, hanem akár külön-külön is lehet állítani, például a bal oldalit:

```
p {
  border-style: solid;
  border-left-width: 15px
}
```

Több érték egyszerre

Ha több oldal szegély vastagságát más-más értékekre, de egyszerre szeretnénk állítani, arra is van lehetőségünk.

Ha négy értéket adunk meg, akkor a felső szegélytől indulva, az óramutató járásának megfelelő sorrendben (felső, jobb oldali, alsó és bal oldali) szegényre fog vonatkozni.

Két érték megadása esetén az első szám a felső és alsó szegély, a második a bal és jobb oldali szegélyt befolyásolja.

Közös deklaráció

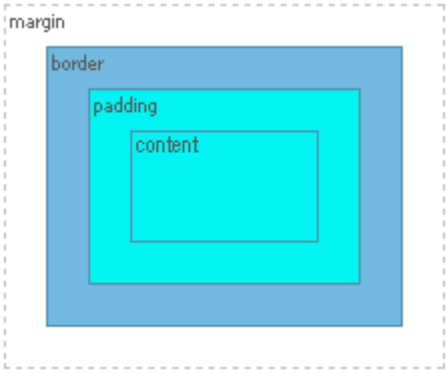
Az eddigi szegély tulajdonságok (hasonlóan a korábbi tulajdonságcsoportokhoz) összevonhatók a border tulajdonság segítségével. Például az alsó szegély több tulajdonságának beállítása:

```
p {border-bottom: medium solid #ff0000}
```

6.7. Térközök a szegélyen belül és kívül

A CSS szintaktikailag két nagyon hasonló tulajdonság-csoportot tartalmaz. A *margin* tulajdonsággal a szegélyen (*border*) kívüli, a *padding* tulajdonsággal pedig a tartalom és a szegély közötti belső margót lehet beállítani. A szintaktikai hasonlóság miatt ebben az alfejezetben csak a (külső) margó szintaxisa fog következni, de minden példa hasonlóan leírható lenne a belső margóra is.

A következő összefoglaló ábra mutatja az alapfogalmakat:



Margók esetén is van lehetőségünk, hogy egyszerre mind a négy oldal értékét, akár különbözőre is beállíthassuk, vagy csak egyetlen oldalét változtassuk meg. Csak a bal oldali margót definiálja:

```
p.leftmargin {margin-left: 2cm}
```

Minden oldalét definiálja, de más-más értékkel:

```
p {margin: 2cm 5px 2em 5%}
```

Minden oldali margót egységesen nulláz:

```
p {margin: 0}
```

Megjegyzés: Az egyes böngészők bizonyos alapértelmezett margóbeállításokat eltérően értelmezhetnek. Az ebből eredő kellemetlenségek és bosszúságok elkerülése érdekében a szerző szokása, hogy egy új oldal CSS állományát valahogy így kezdi:

```
body, p, h1, h2, h3, table, tr, th, td, img {
  margin: 0;
  padding: 0;
}
```

6.8. Listák

A CSS lista formázásai segítségével a lista előtti térközök, a lista elem vagy lista kép állítható be.

Lista jelölők

Felsorolt listák esetén az egyes listaelemek sorrendisége nem jelent információt, ezért minden listaelem előtt ugyanazt a jelölőt szokás alkalmazni. Érdemes megfigyelni a következő példában, hogy a formázás a lista (*ul*) és nem az egyes listaelemek (*li*) tekintetében történik.

```
ul.disc {list-style-type: disc}
ul.circle {list-style-type: circle}
ul.square {list-style-type: square}
ul.none {list-style-type: none}
```

- Kávé
- Tea

- Kávé
- Tea

- Kávé
- Tea

Kávé
Tea

Számozott listák esetén jóval több lehetőségünk van, bár ezek közül is csak néhány tartozik a gyakrabban használtak közé.

```
ol.decimal {list-style-type: decimal}
ol.lroman {list-style-type: lower-roman}
ol.uroman {list-style-type: upper-roman}
ol.lalpha {list-style-type: lower-alpha}
ol.ualpha {list-style-type: upper-alpha}
```

1. Kávé
2. Tea

i. Kávé
ii. Tea

I. Kávé
II. Tea

a. Kávé
b. Tea

A. Kávé
B. Tea

Az előre adott lehetőségeket magunk is tovább bővíthetjök azzal, hogy tetszőleges képet alkalmazhatunk listajelölőnek:

```
list-style-image: url('arrow.gif')
```

- Kávé
- Tea

A listajelölő pozíciója

A listajelölő a szöveg belsejében, vagy az előtt is szerepelhet:

```
ul.inside {
  list-style-position: inside
}
ul.outside {
  list-style-position: outside
}
```

- Egyik listaelem
- Másik, hosszabb listaelem

- Egyik listaelem
- Másik, hosszabb listaelem

Közös deklaráció

A listaelemek esetén is használható a már jól megszokott, rövidebb írásmódot lehetővé tevő közös deklaráció. Erre is láthatunk egy példát:

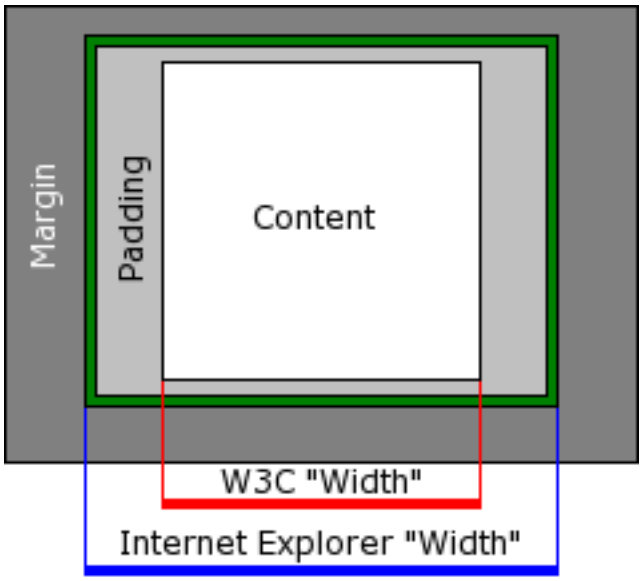
```
list-style: square inside url('arrow.gif')
```

6.9. Méretek

Az elemek szélesség (*width*) és magasság (*height*) tulajdonsága segítségével az elem mérete befolyásolható.

Elsősorban doboz-jellegű elemeknél van értelme használni: például egy kép méretét, vagy a navigációs sáv szélességét gyakran állítjuk be ilyen módon.

Itt kell megemlítenünk, hogy az Internet Explorer 6.0 és 7.0 változatában a *width/height* és *padding* tulajdonságok tekintetében egy bosszantó hiba van. A következő ábra[41] mutatja a probléma lényegét (a vízszintes kiterjedésre koncentrálva):



A W3C szándéka szerint a *width* és a *padding* megadása esetén a *width* a tartalom (*content*) tényleges szélességét jelenti. Ezzel szemben az Explorer a *width* értékét csökkenti a *padding* értékével, és a *content* mérete így kisebb lesz. Nézzünk egy példát:

```
div#doboz {
  width: 200px;
  padding: 10px;
}
```

A dobozban levő szöveg az Explorerben 180, míg a többi böngészőben 200 pixel széles lesz.

6.10. Megjelenítés

A megjelenítés módja

A *display* tulajdonság egy elem más elemekhez viszonyított megjelenését befolyásolja. A tulajdonság három legalapvetőbb értéke a *block*, *inline* és *none*.

[41] forrás: http://en.wikipedia.org/wiki/Internet_Explorer_box_model_bug

Egyes HTML elemeknek (pl. *h1..h6, p, ul, ol, div*) a *block* az alapértelmezése, míg másoknak (pl. *small, img, span*) az *inline*.

Nézzünk egy egyszerű példát a bekezdések *inline*-ként való használatára, és a *div* elemek eltüntetésére:

```
p {display: inline}
div {display: none}
```

Első bekezdés. Második bekedés.

```
<p>Első bekezdés.</p>
<p>Második bekedés.</p>
<div>Nem látható div
elem.</div>
```

Egy másik – gyakorlati szempontból érdekes – példa, ami szerint az oldal főmenüjét felsorolt listával is szokás létrehozni, és a CSS szintjén *inline* listaelemekkel megoldani a menüpontok egymás mellé kerülését. Ezt a példát a következőkben még alaposabban megvizsgáljuk.

Felmerülhet a kérdés, hogy mi értelme van egy elem láthatóságát kikapcsolni. Sok érdekes eset közül csak egy faszerkezetű navigációs elemet nézzünk meg közelebbről. (A Windows Intéző bal oldali panelére érdemes gondolni.) Itt praktikus, ha egyes faelemeket, vagy akár nagyobb részeket is be lehet csukni, hogy a navigáció áttekinthetőbb legyen. Ezt JavaScripttel támogatva egyszerűen meg lehet tenni: kattintás hatására egy bizonyos elem display tulajdonságát kell *none* vagy *block* értékre váltani.

6.10.1 A lebegtetés

Hagyományosan képeknél használt, bár más elemeknél is praktikus lehetőség a *float* tulajdonság alkalmazása. Ennek segítségével a soron belül megjelenő elemet ki tudjuk emelni a sorfolytonosságból, és a környező tartalom körül tudja futni az elemet.

Nézzünk egy képes példát először CSS nélkül. A kép a bekezdés belsejében, sorfolytonosan szerepel (pont úgy, mintha egy nagy méretű betű lenne a szövegben:

```
<p>
Ez egy szöveg. Ez egy szöveg.
Ez egy szöveg. Ez egy szöveg.
Ez egy szöveg. Ez egy szöveg.

<img src="logocss.gif"
width="95" height="84" />
Ez egy szöveg. Ez egy szöveg.
Ez egy szöveg. Ez egy szöveg.
Ez egy szöveg. Ez egy szöveg.
Ez egy szöveg. Ez egy szöveg.
```

Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez

egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy szöveg. Ez egy

Ha a kép elem megkapja a *float: right* formázását, a kép jobb oldalt jelenik meg, körbefuttatva a következő tartalommal:

Ha a kép és a körbefuttatott tartalom közt nagyobb helyet szeretnénk kihagyni, akkor ezt a kép *margin* tulajdonságaival tehetjük meg. Ha a képhez feliratot is szeretnénk társítani, akkor ez egy apró trükkel megvalósítható. A megoldás lényege, hogy nem közvetlenül a képet lebegtetjük, hanem egy *div* elem segítségével egységbe zárjuk a képet és a feliratát, és ezt a csoportosító *div* elemet lebegtetjük.

```
div {
  float:right;
  width:120px;
  margin:0 0 15px 20px;
  padding:15px;
  border:1px solid black;
  text-align:center;
}
```

```
<body>
  <div>
    <img src="logocss.gif"
      width="95" height="84"
      /><br />
    A CSS szuper!
  </div>
  <p>Ez egy szöveg. Ez egy
szöveg. ...
```

Az eddigiek alapján egy iniciálé lebegtetése egyszerű feladat. Tipp: a bekezdés első betűjét tegyük egy *span* elembe, hogy tudjunk formázást hozzákapcsolni.

Horizontális menü

Nézzük most meg a korábban beígért menüs példát. A menü egyszerű HTML lista:

```
<ul>
  <li><a href="#">Menü 1</a></li>
  <li><a href="#">Menü 2</a></li>
  <li><a href="#">Menü 3</a></li>
</ul>
```

A következő eredményt szeretnénk látni:

Megjegyzés: A képen nem látszik, hogy az egérkurzot a *Menü 1* felett van.

Nézzük meg közelebbről a példa fontosabb pontjait.

```
ul {
  float:left;
  width:100%;
```

A menü az alatta levő szövegtől elválik, birtokolja a teljes vízszintes sávját.

```
  padding:0;
  margin:0;
  list-style-type:none;
}
```

Csak maga a szöveg felesleges térközök és jelölő nélkül.

```
a {
  float:left;
  width:6em;
```

Fix szélességű gombok érhetők el. Érdemes azonban vigyázni ezzel a módszerrel, mert ha a szöveg nem fér ki, a dizájn szétesik.

```
  text-decoration:none;
```

Ne legyen aláhúzva a link.

```
  color:white;
  background-color:purple;
  padding:0.2em 0.6em;
  border-right:1px solid white;
}
```

A menün kívüli háttérszín és a jobb oldali szegély színe itt meg kell, hogy egyezzen.

```
a:hover { background-color:#ff3300}
```

Az egérkurzorra színváltással reagál a menüpont. Érdemes megfigyelni, hogy a link a teljes *li* területét elfoglalja, ezért a teljes *li* elem linkként működik.

```
li {display:inline}
```

Ez a sor oldja meg, hogy a menüpontok egymás mellé kerüljenek.

Felesleges táblázatok nélküli oldalkialakítás

A webfejlesztés elmúlt éveiben sok zsákutcát megjártak a szakma művelői. A keretek indokolatlan használatánál talán már csak a táblázatos oldalkialakítás okozott több bonyodalmat és felesleges munkát.

Régi "jól bevált", és sokak által még a mai napig is legjobbnak ítélt módszer a menük oldalra, fejléc felülre, lábléc alulra stb. pozicionálásához a táblázatok használata. A módszer rövidesen elkezdett burjánzani, megjelentek a 2-3-4 szinten egymásba ágyazott táblázatok, az egyesített cellák, a csak térköz kialakításához létrehozott sorok és oszlopok – vagy ami még ennél is elvetemültebb ötlet – a "távtartó gif-ek". Aki már megpróbált egyszer egy ilyen szerkezetű oldalt megérteni, esetleg a dizájnt megváltoztatni, az lehet, hogy néhány ősz hajszállal "gazdagodott". A CSS 2-es verziója óta semmi szükség az ilyen elavult és értelmetlen technikákra.

Példaként nézzünk egy alap oldalelrendezést fejléccel, lábléccel és baloldali (például menü kialakítására alkalmas) sávval. A következőt szeretnénk elérni:

A HTML szerkezet kialakításánál alapvetően a fentről lefelé, azon belül balról jobbra haladó tervezést érdemes követni. (Természetesen összetettebb esetben ez a sorrend nem ilyen egyszerű, és legtöbb esetben többféle megoldás is adható. Másrészt az is egy fontos szempont, hogy a lényegi információtól haladjunk a kevésbé lényeges felé.) Az oldal HTML szerkezete:

```
<body>
  <div class="container">
    <div class="header">
      <h1 class="header">Praesent...</h1>
    </div>
    <div class="left">
      <p>Phasellus wisi nulla...</p>
    </div>
    <div class="content">
      <h2>Aenean nummy odio orci</h2>
      <p>Phasellus wisi nulla...</p>
      <p>Adipiscing elit praesent...</p>
    </div>
    <div class="footer">Praesent...</div>
  </div>
</body>
```

A *container* nevet gyakran alkalmazzák az oldal fő tárolójának azonosításához. Érdemes még azt is megfigyelni, hogy a *left* és *content* doboz nincsenek egy közös dobozba összefogva, bár bizonyos esetekben ez is szükséges lehet.

```
div.container {
  width: 100%;
  margin: 0px;
  border: 1px solid gray;
  line-height: 150%;
}
```

A *container* szélessége alapvetően az egész oldal szélességét határozza meg. Látszik, hogy az oldal ki fogja tölteni a teljes ablakszélességet. A *margin* és *border* tulajdonságok már ismerősek, csak erre a dobozra lesznek hatással, míg a *line-height* öröklődni fog a tartalmazott dobozok irányába. Ehhez hasonlóan színeket, betűtípusokat szokás ilyen módon, egységesen megadni.

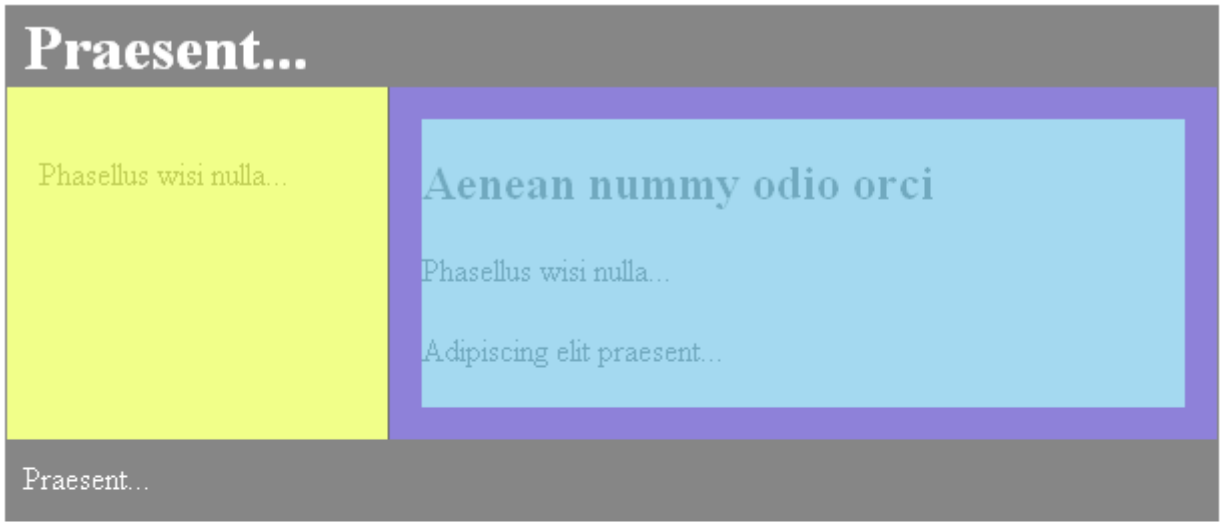
```
div.header,div.footer {
  padding: 0.5em;
  color: white;
  background-color: gray;
  clear: left;
}
```

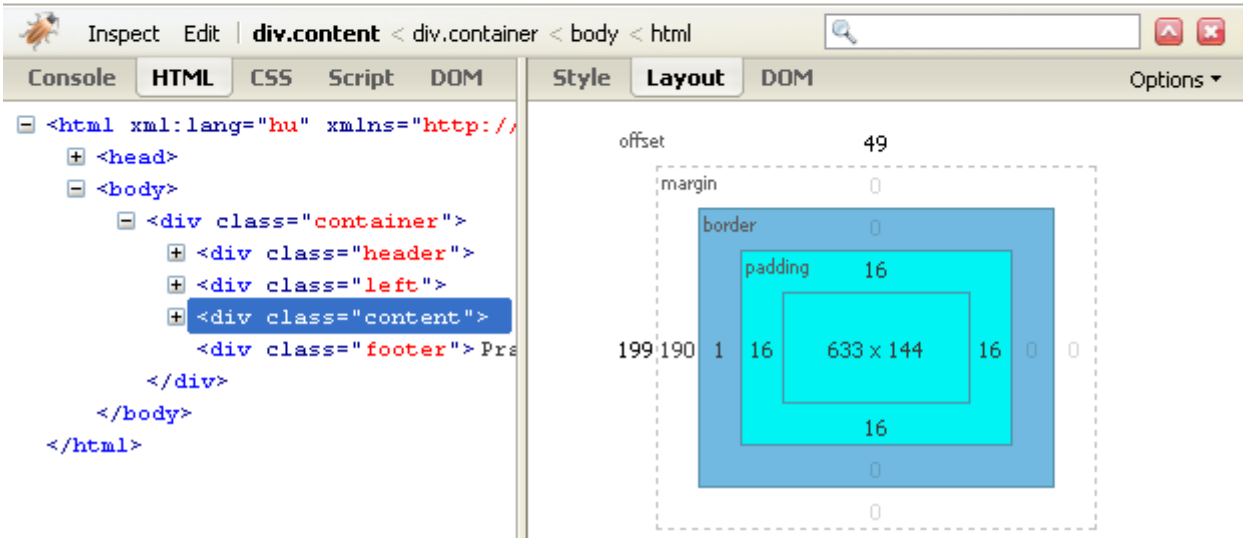
Az utolsó sor megakadályozza, hogy a két doboz bal oldalán lebegő (*float*) elem legyen.

```
h1.header {
  padding: 0;
  margin: 0;
}
div.left {
  float: left;
  width: 160px;
  margin: 0;
  padding: 1em;
}
div.content {
  margin-left: 190px;
  border-left: 1px solid gray;
  padding: 1em;
}
```

A példa talán legérdekesebb részéhez értünk: a *left* és *content* egymás mellé helyezéséhez. Alap tulajdonságokkal ez a két doboz egymás alá kerülne, de a *left* elem *float* formázása lehetővé teszi, hogy a forráskódban utána következő *content* doboz ne alatta, hanem mellette (is) jelenjen meg. Ezen kívül még fontos, hogy a *content* elem bal margója (*margin-left*) is be lett állítva, így a *left* és *content* dobozok soha nem fogják egymást zavarni.

A megértéshez érdemes a működő példát a FireBug kiegészítő Layout nézetével is megnézni. Az ábra azt a pillanatot mutatja, amikor a *content* doboz van kijelölve (ez a bal alsó részen jól látszik). Megfigyelhető a 190 pixel széles, sárgával színezett bal oldali *margin* terület, és a konkrét számérték is a jobb alsó Layout elemben.





Megjegyzés: A táblázatos oldalkialakítási módszer után a tisztán CSS-re építő megoldás logikája furcsa, nehézkes lehet. Hosszú távon azonban busásan meg fogja hálálni a befektetett energia.

Az előző példánál maradva talán az szokta a legtöbb nehézséget okozni, hogy a táblázat celláinál megszokott háttérbeállítások itt nem úgy működnek, hiszen itt nem két egyforma magasságú celláról van szó. Ilyen jellegű probléma esetén van egy egyszerű megoldás: az egymás mellé kerülő dobozokat egy közös tartalmazó dobozba helyezzük, és a háttér ügyes beállításával el lehet azt a hatást érni, amit szeretnénk. Tehát nem az egymás melletti dobozok, hanem az őket közösen tartalmazó doboz hátterét kell beállítanunk.

6.10.2 Pozicionálási sémák

A *position* tulajdonság segítségével az alapértelmezett *static* beállítás helyett relatív és abszolút pozicionálást is kérhetünk.

Relatív pozíció

Az elemeket alapértelmezett (*static*) helyzetüktől el tudjuk mozgatni a relatív pozicionálás segítségével vízszintes és függőleges irányban. A relatív eltolás mértékét a *left, right, top* és *bottom* tulajdonságokkal határozhatjuk meg. Az így eltolt elem “eredeti” helye üresen marad, oda más elem nem fog becsúszni. (Ez lényeges eltérés a lebegtetett elemekhez képest.)

A következő árnyékkal ellátott cím példa szemantikusan ugyan nem szerencsés, de jól szemlélteti a relatív pozicionálás lehetőségeit. A megoldás lényege, hogy a cím szövege két példányban szerepel a HTML forrásban, de vizuálisan majdnem egymást elfedve, és más színnel jelennek meg. Nézzük a példát:

```
<h1 class="shadow">Főcím</h2>
<h1 class="main">Főcím</h2>
<p>
  Bekezdés. Bekezdés.
  Bekezdés...
</p>
```

```
h1 {
  font-size:  2em;
}
h1.main
{
  color:     black;
  position:  relative;
  top:       -1.9em;
  left:      -0.1em;
}
h1.shadow {
  color:     #bbb;
}
```

A megoldás hátránya is jól látszik a képen: a címet követő bekezdés nem követi közvetlenül a címet. Természetesen ez a hátrány további trükkökkel kiküszöbölhető.

Abszolút pozíció

A relatív pozíció esetén csak eltoltuk az elemet a helyéről, de az eltolás a környezetére nem volt más hatással. Ezzel szemben az abszolút módom pozicionált elem nem tart fenn egy területet. A megadott pozíciók itt a tartalmazott dobozon belüli abszolút pozíciók, és a következő elemek számára nem is léteznek.

Az előző példa így megoldható abszolút pozicionálással is a korábbi hátrány nélkül.

```
h1 {
  font-size: 2em;
  margin:    0px;
}
h1.main {
  color:     black;
  top:       6px;
  left:      6px;
  position:  absolute;
}
h1.shadow {
  color:     #bbb;
  margin:    2px;
}
```

Fix pozíció

A fix pozíció az előző speciális esete. Az elem itt is kikerül a pozicionálási sémából, de itt nem a tartalmazott doboz, hanem a látótér (képernyő) a viszonyítási pont.

Nagyszerűen alkalmazható, pl. a hagyományos keretek (*frame*-k) kiváltására[42].

Megjegyzés: Az Explorer nem teljes körűen támogatja.

6.10.3 Láthatóság

A *visibility* tulajdonság segítségével az elem alapértelmezett láthatósága (*visible* érték) helyett el is rejthetjük azt (*hidden* érték).

Példaként megemlíthetjük, hogy egy bonyolultabb adminisztrációs felületet könnyebben áttekinthetővé tehet egy több füles elrendezés, ahol a felhasználó tetszőleges fülre kattintva csak az adott részletekkel akar foglalkozni. Ekkor a hagyományos, minden fül-kattintás esetén újratöltődő megoldás helyett JavaScript segítségével látványosabb megoldást is alkalmazhatunk. A megoldás alapja, hogy ugyanazon a helyen több doboz helyezkedik el, de mindig csak egy látható, a többi nem. A fület megvalósító elemek kattintás eseménye esetén azt a dobozt kell láthatóvá tenni, amelyik az adott fülhöz tartozik, míg az előző aktuális elemet láthatatlanná kell tenni.

Példaként érdemes megnézni a következő példát: http://stilbuero.de/jquery/tabs/

6.10.4 Z-index

Legtöbb esetben az elemek nem takarják egymást. Ha mégis, akkor alapértelmezetten a (HTML-ben) későbbi elem takarja el a korábbit. Ha ez nem megfelelő, akkor a *z-index* értékek meghatározásával manipulálhatjuk a vizuális takarást.

6.11. Látszólagos osztályok

A kiválasztók között speciális feladatot látnak el a látszólagos osztályok. Ezek ugyanis olyan elemekre vonatkoznak, amelyek nem fizikailag, hanem az adott környezetben, szituációban válnak különlegessé.

6.11.1 Linkek viselkedése

Valószínűleg legismertebb látszólagos osztályok a linkekhez kapcsolódnak. Hagyományosan más-más színnel szokás jelezni az egyszerű linkeket, a már meglátogatott linkeket, az éppen az egérkurzor alatt levő és a kattintás közben levő linkeket:

```
a:link    {color: #FF0000}
a:visited {color: #00FF00}
a:hover   {color: #FF00FF}
a:active  {color: #0000FF}
```

Ma talán a *:hover* látszólagos kiválasztóval találkozhatunk a legtöbbször, és nem is csupán a betűszínek, hanem akár komolyabb „viselkedés” is megvalósítható vele.

[42] Egy nagyon szép megoldás:
http://www.456bereastreet.com/archive/200609/css_frames_v2_fullheight/

6.11.2 Első gyermek

A *:first-child* látszólagos osztály egy adott elem első gyermekét képes kiválasztani. Nézzük a következő példát:

```
a:first-child {
  text-decoration:none
}
```

Ennek hatására minden első gyermekként előforduló *a* elemre érvényes lesz a fenti formázás. Például:

```
<p>Nézze meg a
  <a href="http://www.gamf.hu">GAMF</a>
  és az
  <a href="http://informatika.gamf.hu">
    Informatika tanszék</a>
  honlapját!</p>
```

Nézze meg a GAMF és az Informatika tanszék honlapját!

Ha az adott elemben (itt *p*) nem link (*a*) az első gyermek elem, akkor az adott környezetben ennek a formázásnak semmilyen hatása nem lesz. A következő verzióban a *strong* elem átvette az első gyermek szerepét:

```
<p>
  <strong>Nézze meg</strong> a
  <a href="http://www.gamf.hu">GAMF</a>
...
```

Nézze meg a GAMF és az Informatika tanszék honlapját!

6.11.3 Első betű és első sor

A *:first-letter* kiválasztó segítségével az első betű, míg a *:first-line* segítségével az első sor kaphat speciális formázást. Nézzünk példaként egy iniciálét, valamint egy félkövér első sort:

```
<p>
  Tetszőleges szövegű bekezdés
</p>
```

TETSZŐLEGES szövegű bekezdés

```
p:first-letter {
  color: #ff0000;
  font-size:300%;
  vertical-align: top;
  float: left;
}
p:first-line {
  font-variant: small-caps;
}
```

6.12. Média típusok

Ha már egy weboldalt többféle környezetben használhatunk. Egy weboldalt megnézhetünk képernyőn, kéziszámítógépen, mobiltelefonon, vagy akár kinyomtatva is.

Bizonyos formázások csak egyes médiák esetén értelmezhetők, ezért természetesen hasznos lehetőség a médiatípusok szerint eltérő formázás alkalmazása.

Nézzünk példát a betűtípusok eltérő kezelésére. Nyomtatásban és képernyőn más-más méretet és betűtípust alkalmazhatunk a következő stílusokkal:

```
@media screen {
  p.test {font-family: verdana, sans-serif; font-size: 14px}
}

@media print {
  p.test {font-family: times, serif; font-size: 10px}
}

@media screen, print {
  p.test {font-weight: bold}
}
```

Másik gyakran alkalmazott lehetőség, amikor a HTML kódhoz külön-külön stíluslap állományokat készítünk:

```
<link rel="stylesheet" type="text/css"
  href="screen.css" media="screen" />
<link rel="stylesheet" type="text/css"
  href="print.css" media="print" />
```

A fontosabb médiatípusok:

all	minden eszköz
aural	felolvasó szoftver
handheld	kézi megjelenítő
print	lapozható (nyomtatás)
projection	vetítés (mint egy kirakati reklám, vagy az S5[43] módszer)
screen	képernyő

6.13. Validátor

A CSS kód ellenőrzésére az online is használható W3C CSS Validation Service[44] egy nagyon jó eszköz. Arra azért figyelni kell, hogy csak érvényes HTML kóddal érdemes a CSS érvényességét vizsgálni.

[43] http://meyerweb.com/eric/tools/s5/

[44] http://jigsaw.w3.org/css-validator/

6.14. Esettanulmány

6.14.1 Happy Holidays

Ebben a fejezetben egy olyan komplex oldal bemutatására kerül sor, amelyik jól illusztrálja a dizájn fontosabb részeit.

A szerző az ingyenesen használható webhely-gyűjtemények körül legjobbnak az Open-Designs[45] gyűjteményét tartja. A következő bemutatott példa is innen származik.

A bemutatás során érdemes megfigyelni, hogy hasonló módszerrel érdemes általában is egy dizájn felépítését megismerni. A legfontosabb felhasznált eszközök az editoron kívül a Firefox két kiegészítője: a *FireBug* és a *Web Developer*.

Ez a dizájn[46] sokféle szempontból korszerűnek, divatosnak tekinthető. Többféle oldalelrendezést is tartalmaz. Elsősorban a kezdőoldalra fogunk koncentrálni. Nézzük meg először a kész oldalt:

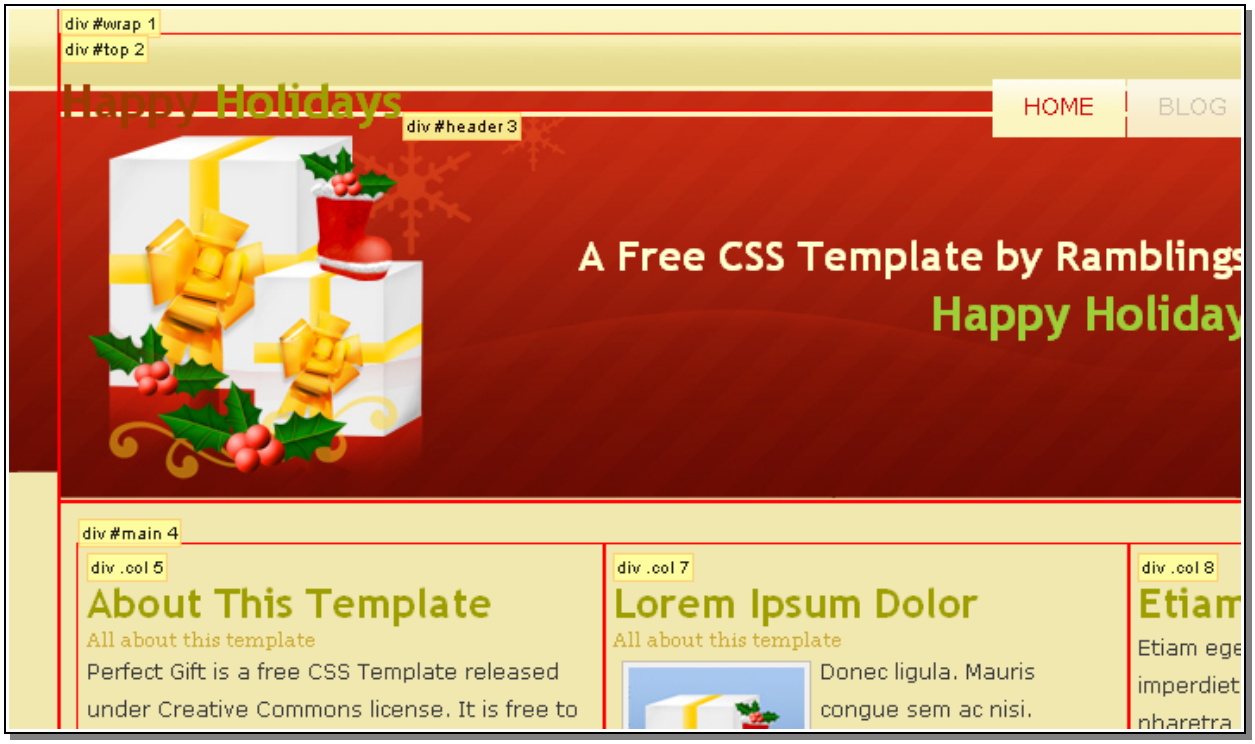
Szemet gyönyörködtető, jól áttekinthető kezdőoldal kialakítására nagyszerűen megfelel.

[45] http://www.opendesigns.org/

[46] Letöltve: http://www.opendesigns.org/preview/?template=1181, eredeti forrás: http://www.ramblingsoul.com/

Mivel az oldalkialakítás váza alapvetően egymásba ágyazott *div*-ekből áll, érdemes a *div*-ek felépítésével kezdeni. Ennek áttekintésére egyik leggyorsabb módszer a *Web Developer* kiegészítő *Information/Display div order* menüpontját megnézni.

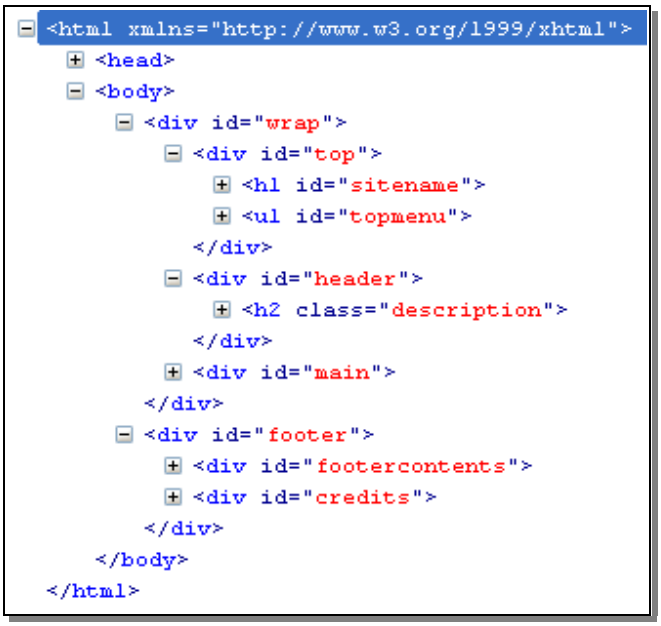
(Itt a nyomtatás korlátai miatt először csak a bal felső sarkot tudjuk megnézni. Egy komolyabb elemzéshez érdemes a teljes oldalt ilyen módon kinyomtatni.)



A kiegészítő piros vonalakkal berajzolja az oldalon található *div*-ek határait, és a bal felső sarkában néhány információt is megmutat:

1. a *div* szót
2. a *div* elem *id*-jét
3. a *div* HTML-en belüli sorrendjét

Ha ehhez a nézethez hozzávesszük a *FireBug* kiegészítő HTML faszerkezetű nézetét, egy igen jó áttekintést kaphatunk az oldal szerkezeti felépítéséről.



Nézzük meg az elemek néhány fontosabb CSS formázását. A sorrendnél a fa szerint a szülőktől a gyermekekig érdemes haladni.

body

A háttérképe egy vízszintesen ismétlődő kép, ami felső szöveges sor (*Happy ... FORMS*) és a piros sáv háttere lesz.

Háttérszíne az oldal törzsének háttérszíne lesz majd. Ez azt is jelenti, hogy a *#main* ezt nem fogja semmivel felülírni.

```
body {
  background:#F0E7AC url(images/bg.jpg) repeat-x scroll
  center top;
}
```

A *body #wrap* és *#footer* részre oszlik.

#wrap

Ez egy „összefogó” elem, az oldal nagy része (*#top, #header, #main*) bele tartozik.

A *width: 940px;* az oldal aktív részének szélességét határozza meg.

#top

A felső szöveges sort (*Happy ... FORMS*) tartalmazza. A *height:45px;* a sáv magasságát határozza meg. Tartalmazza a *#sitename* (ami egy h1-es cím) és *#topmenu* elemeket.

#topmenu

Felsorolt, jobbra lebegtetett lista a menühöz.

```
#topmenu {
  display:block;
  float:right;
  list-style-image:none;
  list-style-position:outside;
  list-style-type:none;
  min-width:400px;
  padding-top:11px;
}
```

#header

A fenti piros sáv, benne egy h2-es címmel. A háttérképe 940x228 pixeles.

A h2-es cím formázása:

```
#header .description {
  color:#FFFFCC;
  display:block;
  font-size:24px;
  padding:55px 25px 25px 300px;
}
```

#main

Az oldal fő tartalmi része. A fenti ábrán 3 „hasábot” tartalmaz, amelyek formázása:

```
#main .col {
  float:left;
  padding:5px;
  width:32%;
}
```

A három doboz egymás mellé csúszik, a 32%-os méretek ezt éppen lehetővé teszik. Az „összeérést” a *padding: 5px* zárja ki.

A dizájn tartalmaz (*blog.html*) egy más típusú oldalelrendezést is, ahol a *#content* és *#sidebar* részek 2:1 arányban a tartalmat és az oldalsó sávot tartalmazzák. Ezek formázása:

```
#content {
  float:left;
  padding:5px;
  width:65%;
}

#sidebar {
  float:right;
  padding:5px;
  width:32%;
}
```

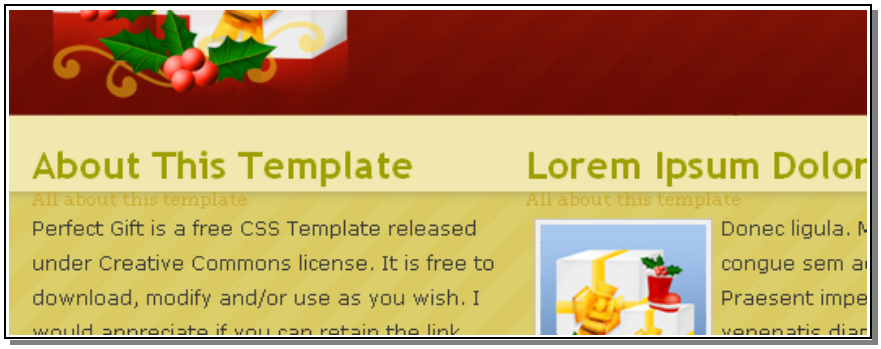
A *#main* mindkét esetben egy üres blokkal záródik:

```
<div class="clear"/>

.clear {
  clear:both;
}
```

Ennek célja, hogy a következő *#footer* kezdetéig tartson a *#main*. (Ez azért nem triviális, mert ha egy doboznak csak lebegtetett gyermekei vannak, akkor a háttér nem fog mind-

egyik „alatt” kitartani.) A *.clear* elem nélkül a *#footer* (és lényegében annak háttere) felcsúszna:



#footer

A *#footercontents* (csíkos hátterű) és a *#credits* elemeket tartalmazza.

6.15. Ellenőrző kérdések

Minek a rövidítése a CSS?

- Colorful Style Sheets
- Cascading Style Sheets
- Computer Style Sheets
- Creative Style Sheets

Hogyan tudunk külső stíluslapot HTML oldalhoz kapcsolni?

- *<stylesheet>mystyle.css</stylesheet>*
- *<style src="mystyle.css">*
- *<link rel="stylesheet" type="text/css" href="mystyle.css">*

A HTML dokumentum melyik részében kell a külső stíluslapra hivatkozni?

- A *body* tagban
- A dokumentum elején
- A dokumentum végén
- A *head* tagban

Milyen HTML tulajdonsággal lehet soron belüli stílusmegadást alkalmazni?

- *font*
- *styles*
- *style*

- *class*

Melyik helyes CSS szintaxis?

- *{body;color:black}*
- *body {color: black}*
- *{body:color=black(body}*
- *body:color=black*

Hogyan lehet megjegyzést beszúrni a CSS fájlba?

- *// this is a comment*
- *// this is a comment //*
- */* this is a comment */*
- *' this is a comment*

Melyik tulajdonság határozza meg a háttérszínt?

- *color*
- *background-color*
- *bgcolor*

Hogyan tudunk minden h1 tagnak háttérszínt beállítani?

- *h1 {background-color:#FFFFFF}*
- *all.h1 {background-color:#FFFFFF}*
- *h1.all {background-color:#FFFFFF}*

Hogyan lehet egy tag szövegszínét beállítani?

- *text-color=...*
- *text-color: ...*
- *fgcolor: ...*
- *color: ...*

Milyen tulajdonsággal állítható be a szöveg mérete?

- *text-size*
- *text-style*
- *font-size*
- *font-style*

Melyik helyes, ha minden p tagot félkövéren akarunk megjeleníteni?

- *p {font-weight:bold}*
- *<p style="text-size:bold">*
- *p {text-size:bold}*
- *<p style="font-size:bold">*

Hogyan tudjuk a linkeket aláhúzás nélkül megjeleníteni?

- *a {decoration:no underline}*
- *a {text-decoration:no underline}*
- *a {underline:none}*
- *a {text-decoration:none}*

Hogyan tudjuk a szöveg betűtípusát beállítani?

- *font=*
- *font-family:*
- *f:*

Hogyan tudunk szöveget félkövéren megjeleníteni?

- *font-weight:bold*
- *font:b*
- *style:bold*

Hogyan tudjuk beállítani egy tag bal margóját?

- *margin-left:*
- *indent:*
- *text-indent:*
- *margin:*

Hogyan tudjuk beállítani, hogy a lista elemek négyzettel jelenjenek meg?

- *list-type: square*
- *list-style-type: square*
- *type: square*
- *type: 2*

6.16. Feladatok

Elemzés

Keressen például az OpenDesigns oldalán (X)HTML és CSS szerint szabványos oldalakat, a fenti esettanulmányhoz hasonlóan elemezze, és ízlés szerint módosítsa a felépítését.

Szabványosítás

Ha az előző keresés folyamán talált nem szabványos oldalt, akkor módosítsa úgy a CSS-t, hogy szabványossá váljon!

Teljes dizájn készítése

Keressünk egy olyan oldalt a weben, amelyik tetszetős a számunkra. Az *Alt+PrintScrn* billentyűkombinációval helyezzük vágólapra, majd egy képszerkesztő programmal mentsük el. (Használhatjuk a Firefox *Save As Image*[47] kiegészítőjét is.)

Készítsünk teljes HTML+CSS dizájnt a kép feldarabolt részeit is felhasználva, ami a lehető legjobban hasonlít a fényképre, ugyanakkor legyen szabványos is!

6.17. További források

Magyar nyelven messze a legfontosabb forrás a Weblabor ide vonatkozó cikksorozata több mint 10 cikkel. Az első cikk címe: *http://weblabor.hu/cikkek/cssalapjai1*. A cikksorozaton kívül további fórumtémák, hírek, blogmarkok foglalkoznak a legújabb technikákkal.

Ezen kívül összefoglaló oldalként a *http://css.lap.hu* gyűjteményét érdemes megemlíteni.

[47] https://addons.mozilla.org/en-US/firefox/addon/3408

7. JavaScript

A JavaScript ma az egyetlen jelentős kliens oldali szkriptnyelv. Weboldalak milliói használják űrlapok helyes kitöltésének ellenőrzéséhez, sütik kezeléséhez, a felhasználói élmény növeléséhez. Sajnos sok olyan oldal van, ahol az alkalmazott JavaScript kód öncélú, sőt meglehetősen zavaró az oldal használója számára. Végül meg kell még említeni, hogy az AJAX és hasonló technikák ismét erősítik a JavaScript hasznos felhasználását, a felhasználói élmény növelését.

7.1. Bevezetés a JavaScript nyelvbe

Mi a JavaScript?

- Szkriptnyelv (tehát értelmezett, a böngésző értelmezi)
- Interaktivitást (működést) ad a HTML oldalhoz
- HTML oldalba építhető, vagy attól jobban elszeparálható kód
- Események kezelésére alkalmazható
- A neve ellenére nincs szoros kapcsolatba a Java nyelvvel
- C++ és Java szintaxisra alapoz
- A szabvány leírását az ECMAScript specifikáció tartalmazza

Kód beillesztése a HTML oldalba

A HTML *script* tagja használható arra, hogy az oldalhoz JavaScript kódot kapcsoljunk. A *script* tag tartalma közvetlenül nem jelenik meg a böngésző ablakában, hanem a böngésző fogja azt futtatni. Például a következő kód a HTML aktuális helyére a *Hello Világ!* szöveget jeleníti meg:

```
<body>
  <script type="text/javascript">
    document.write("Hello Világ!")
  </script>
</body>
```

Megjegyzés: Ha megnézzük a böngészőben az oldalunk forrását, akkor látható, hogy ott az eredeti, letöltött verzió látszik, és nem a JavaScript által „módosított”.

Jól látszik a példán, hogy JavaScriptben elfelejthetjük a sorokat lezáró pontosvesszőket.

A *document.write* módszerét a mai korszerű megközelítésben nem illik alkalmazni.

Nagyon egyszerű kódok kivételével nem szokás[48] a JavaScript kódot a *body* tagba tenni. Összetettebb működést inkább önálló függvények formájában szoktunk elhelyezni a *head* tagba, vagy még inkább külső JavaScript állományba. Nézzük meg ezek módjait:

[48] A könnyebb áttekinthetőség érdekében a demonstrációs példákban, vagy éppen a fejlesztés közben ezt a szabályt időnként áthágjuk.

```
<head>
  <script type="text/javascript">
    function message() {
      alert("This alert box was called with the onload event")
    }
  </script>
</head>
<body onload="message()">
...
</body>
```

A példán látszik, hogy a *body* tag betöltődésekor (*onload* esemény) fog lefutni a *message* nevű függvény, ami egy figyelmeztető dialógusablakot dob fel. Ha a *message* függvény kódját egy *message.js* állományban helyeztük volna, akkor a head részbe a következő kód lenne szükséges:

```
<script type="text/javascript" src="message.js"></script>
```

7.1.1 Változók

A változók olyan tárolók, amelyekbe adatokat helyezhetünk. A változó értékét meg tudjuk változtatni a szkript futása során. A változóra a nevével tudunk hivatkozni, és az értékét bármikor lekérdezni vagy módosítani.

A változó nevek (mint ahogy az egész JavaScript nyelv) kis-nagybetű érzékenyek, és betűvel vagy aláhúzás karakterrel kezdődnek.

Változó deklaráció

A *var* kulcsszó segítségével a következő módon tudunk változót létrehozni:

```
var strname = "Hege"
```

Létre lehet hozni változót *var* nélkül is:

```
strname = "Hege"
```

Ahogy a példákban is látszik, egyből értéket is adtunk a változóknak.

Változók élettartalma, láthatósága

Ha egy változót függvényen belül hozunk létre, a változó csak a függvényen belül érhető el. Ha kilépünk a függvényből, a változó megszűnik. Ezeket a változókat **lokális változó**knak nevezzük. Ha több különböző függvényben használjuk ugyanazt a változónevet, akkor azok egymástól független változók lesznek, mindegyik csak abban a függvényben látható, ahol létrehoztuk.

Ha egy változót a függvényeinken kívül deklarálunk, akkor az oldal minden függvényéből el tudjuk érni azt. A változó akkor jön létre, amikor deklaráljuk, és akkor szűnik meg, ha az oldalt bezárja a felhasználó.

Típusok

Egyszerű példák esetén nem túl sokat kell foglalkoznunk a típusok kérdésével, hiszen a JavaScript más szkriptnyelvekhez hasonlóan dinamikus típusrendszerű, vagyis a változó aktuális értéke határozza meg a típusát, ami bármikor megváltoztatható, pl. egy más típusú kifejezés értékadásával.

A változók típusa lehet objektum, primitív érték (*undefined*, *null*, *boolean*, *string*, *number*) vagy metódus (függvény objektum).

7.1.2 Elágazások

Egy egyszerű lineáris kód csak nagyon egyszerű feladatokat tud megoldani. Legtöbbször szükségünk lesz különböző vezérlési szerkezetekre, hogy a kódunk megoldhassa a feladatát.

Kód írása közben nagyon gyakran alkalmazunk elágazásokat, hogy különböző esetek között különbségeket tudjunk tenni.

A JavaScript a C-hez és Javához hasonlóan többféle elágazást ismer.

if utasítás

Akkor használjuk, ha egy adott feltétel beteljesülése esetén szeretnénk valamit végrehajtani. Szintaxis:

```
if (feltétel) {
  utasítás
}
```

A következő kód Jó reggelt kíván, ha még nincs 9 óra.

```
<script type="text/javascript">
  var d=new Date()
  var time=d.getHours()

  if (time<9) {
    document.write("<strong>Jó reggelt!</ strong >")
  }
</script>
```

A következő példa az ebédidőre figyelmeztet.

```
<script type="text/javascript">
  var d=new Date()
  var time=d.getHours()
  if (time==12) {
    document.write("<strong >Ebédidő!</ strong >")
  }
</script>
```

if-else utasítás

Akkor van szükségünk erre az utasításra, ha a feltételünk teljesülése mellett a nem teljesülés esetén is kell valamilyen feladatot ellátni. Szintaxis:

```
if (feltétel) {
  utasítás ha igaz
} else {
  utasítás ha hamis
}
```

Egészítsük ki a „Jó reggelt” példánkat:

```
<script type="text/javascript">
  var d=new Date()
  var time=d.getHours()

  if (time<9) {
    document.write("<strong>Jó reggelt!</ strong >")
  } else {
    document.write("Jó napot!")
  }
</script>
```

switch utasítás

Ez az utasítás akkor alkalmazható nagyszerűen, ha egy adott kifejezés különböző értékei esetén más-más feladatot kell a kódnak végrehajtani. Szintaxis:

```
switch(n) {
case 1:
  utasítás 1
  break
case 2:
  utasítás 2
  break
default:
  utasítás
}
```

Először az *n* kifejezés kiértékelése történik meg, majd a lehetséges *case* esetek között próbál a böngésző megegyezőt találni. Ha esetleg nincs, akkor a *default* címke fog aktiválódni. A *break* pedig azért szükséges, hogy a programunk ne tudjon a másik esetre rácsorogni.

A következő példa a hét napjától függően a munkahelyi hangulatot képes megjeleníteni.

```
<script type="text/javascript">
  var d=new Date()
  theDay=d.getDay()
  switch (theDay) {
  case 5:
    document.write("Végre péntek!")
    break
  case 6:
  case 0:
    document.write("Jó kis hétvége!")
    break
  default:
    document.write("Mikor lesz hétvége?")
  }
</script>
```

7.1.3 Operátorok

Aritmetikai operátorok

operátor	név	példa	eredmény
+	összeadás	x=2	4

		y=2 x+y	
-	kivonás	x=5 y=2 x-y	3
*	szorzás	x=5 y=4 x*y	20
/	osztás	15/5 5/2	3 2.5
%	maradék képzés	5%2 10%8 10%2	1 2 0
++	növelés 1-el	x=5 x++	x=6
--	csökkentés 1-el	x=5 x--	x=4

Értékadó operátorok

operátor	példa	eredmény
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Összehasonlító operátorok

operátor	név	példa	eredmény
==	egyenlő	5==8	hamis
===	egyenlő típus és érték	x=5 y="5" x==y x===y	 igaz hamis

!=	nem egyenlő	5!=8	igaz
>	nagyobb, mint	5>8	hamis
<	kisebb, mint	5<8	igaz
>=	nagyobb, vagy egyenlő, mint	5>=8	hamis
<=	kisebb, vagy egyenlő, mint	5<=8	igaz

Logikai operátorok

operátor	név	példa	eredmény
&&	és	x=6 y=3 (x < 10 && y > 1)	igaz
||	vagy	(x==5 || y==5)	hamis
!	nem	!(x==y)	igaz

Sztring operátor

Sztringek összefűzésére a + operátor egyszerűen alkalmazható:

```
txt1="Ez egy nagyon "
txt2="szép nap!"
txt3=txt1+txt2
```

Ekkor *txt3* értéke: *"Ez egy nagyon szép nap!"*.

Feltételes operátor

A háromoperandusú feltételes operátor egyszerű esetekben egy *if-else* utasítást is kiválthat. Szintaxis:

```
változónév=(feltétel)?kifejezéshaigaz:kifejezéshahamis
```

A *feltétel* teljesülése esetén *változónév* értéke *kifejezéshaigaz* értéke lesz, különben pedig *kifejezéshahamis* értéke.

A következő példa a Jó reggelt újabb verziója:

```
<script type="text/javascript">
  var d=new Date()
  var time=d.getHours()
  var message = "Jó " + ((time<9)?"reggelt":"napot") + "!"
  document.write(message)
</script>
```

7.1.4 Dialógusablakok

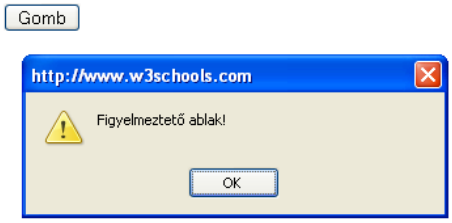
Időnként szükségünk lehet egy honlapon arra, hogy a felhasználó figyelmét ráirányítsuk valamilyen fontos információra egy dialógusablak segítségével. Ezek a dialógusablakok típusuktól függően más-más függvénnyel érhetők el.

Nézzük meg a fontosabb lehetőségeket példákon keresztül.

Üzenetablak

Az üzenetablak csupán egy egyszerű információt közöl, ezért az *alert* függvénynek is csak egy paramétert kell megadnunk. A felhasználó mindössze tudomásul veheti a közölt üzenetet.

```
<head>
  <script type="text/javascript">
  function disp_alert() {
    alert("Figyelmeztető ablak!")
  }
  </script>
</head>
<body>
  <form>
    <input type="button"
      onclick="disp_alert()"
      value="Gomb">
  </form>
</body>
```

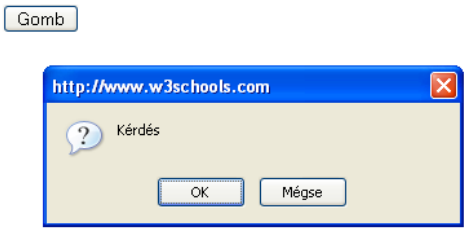


Kérdés

Bizonyos esetekben hasznos, ha a felhasználótól külön dialógusablakból kérünk választ egy eldöntendő kérdésre. Bizonyos esetekben ez praktikusabb, mint egy oldalon elhelyezett űrlap. Érdemes azonban arra is figyelni, hogy csak tényleg indokolt esetben alkalmazzunk üzenetablakokat!

A *confirm* függvény visszatérési értéke jelzi, hogy a felhasználó melyik gombot választotta.

```
<head>
  <script type="text/javascript">
    function disp_confirm() {
      if (confirm("Kérdés")) {
        document.write("OK")
      } else {
        document.write("Mégse")
      }
    }
  </script>
</head>
<body>
  <form>
    <input type="button"
      onclick="disp_confirm()"
      value="Gomb">
  </form>
```

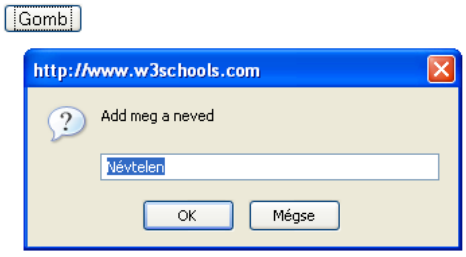


```
</body>
```

Egyszerű adatbevitel

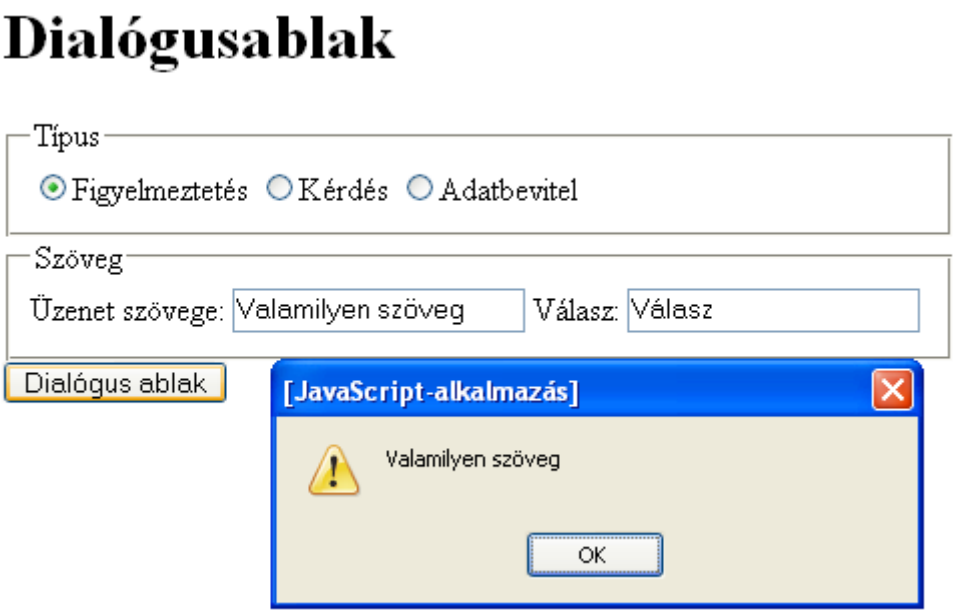
A *prompt* függvénnyel egy egysoros szöveg bevitelét kérhetjük a felhasználótól. A függvény első paramétere az üzenet szövege, a második pedig az alapértelmezett válasz (nem kötelező megadni). Itt is a függvény visszatérési értékén keresztül juthatunk a felhasználó válaszához. (Ha a felhasználó a *Mégse* gombot választotta, *null* a visszaadott érték.)

```
<head>
  <script type="text/javascript">
    function disp_prompt() {
      var name=prompt(
     "Add meg a neved","Névtelen")
      if (name!=null && name!=""){
        document.write(
            "Szia " + name + "!")
      }
    }
  </script>
</head>
<body>
  <form>
    <input type="button"
      onclick="disp_prompt()"
      value="Gomb">
  </form>
</body>
```



Dialógusablakok dinamikus létrehozása

A következő példa a dialógusablakok tesztelését teszi lehetővé. Nézzük először a működését. Hozzunk létre egy űrlapot, amelyik segítségével különböző dialógusablakok hozhatók létre, majd a válasz is értelmezhető lesz.



A *Típus* segítségével a 3 -féle dialógus-típus közül választhatunk, az *Üzenet szövege* kerül a dialógusablakba, a *Válasz* mező pedig a függvény visszatérési értékét jeleníti majd meg. Nézzük az űrlap kódját:

```
<h1>Dialógusablak</h1>

<form name="urlap" method="get" action="#">
  <fieldset>
    <legend>Típus</legend>
    <label><input checked="checked" type="radio" name="tipus"
value="alert" />Figyelmeztetés</label>
    <label><input type="radio" name="tipus" value="confirm"
            />Kérdés</label>
    <label><input type="radio" name="tipus" value="prompt"
            />Adatbevitel</label>
  </fieldset>

  <fieldset>
    <legend>Szöveg</legend>
    <label>Üzenet szövege: <input type="text" name="szoveg"
            value="" /></label>
    <label>Válasz: <input type="text" name="valasz"
            value="Válasz" /></label>
  </fieldset>

  <input onclick="uzenet();" type="button" name="gomb"
value="Dialógus ablak" ></input>
</form>
```

Az érdemi JavaScript kód a *head* részben található *uzenet* függvényben lesz:

```
function uzenet() {
  switch (true) {

    case document.urlap.tipus[0].checked:
      document.urlap.valasz.value =
          alert(document.urlap.szoveg.value)
      break

    case document.urlap.tipus[1].checked:
      document.urlap.valasz.value =
          confirm(document.urlap.szoveg.value)
      break

    case document.urlap.tipus[2].checked:
      document.urlap.valasz.value =
          prompt(document.urlap.szoveg.value)
      break
  }
}
```

Annak vizsgálására, hogy melyik rádiógomb is a kiválasztott, a rádiógomb *checked* tulajdonságának figyelését kell megoldanunk. Érdemes megfigyelni, hogy milyen logika szerint érjük el az egyes űrlapelemeket:

```
document.urlap.tipus[0].checked
```

A függvények válasza hasonlóan rugalmasan elhelyezhető a válasz mezőben:

```
document.urlap.valasz.value =
          alert(document.urlap.szoveg.value)
```

7.1.5 Függvények

A függvények az újra felhasználható kód egyszerű eszközei. Egy egyszer megírt függvényt a kód különböző helyein tetszőlegesen sokszor meghívhatjuk. (Egy jól megírt függvényt akár egymástól jelentősen eltérő weboldalakon is használhatunk.) Másrészt a függvények teszik lehetővé, hogy a HTML kódtól minél jobban elválasszuk a viselkedési réteget megvalósító JavaScriptet. (Hasonlóan, mint ahogy a megjelenítést meghatározó CSS-t is elkülönítjük.)

A függvények használatának egyik legegyszerűbb esete, amikor valamilyen esemény bekövetkeztekor egy eseménykezelő függvénynek adjuk át a vezérlést. Az előző fejezet példái is jól mutatják ezt a megközelítést.

Függvénydeklaráció

A deklaráció szintaxisa:

```
function függvénynév(par1,par2,...,parX) {
  függvénytörzs
}
```

A *function* kulcsszó szükségessége egy kicsi furcsa lehet a C/C++ nyelv után, de a kulcsszó nem elhagyható, így meg kell szokni a használatát. A kerek zárójelek között a függvények paramétereit adhatjuk meg. (Természetesen paraméter nélküli függvényeket is készíthetünk, de a zárójeleket ekkor is alkalmaznunk kell.)

Visszatérési érték

Egyes függvények visszatérési értéket is előállíthatnak. A visszatérési érték a *return* utasítással adható meg. A kód a return hatására a függvényből kilépve a vezérlést is visszaadja az őt hívó kódnak.

A következő egyszerű függvény a két paramétere szorzatát adja vissza:

```
function prod(a,b) {
  return a*b
}
```

A függvényt például a következő módon tudjuk meghívni:

```
product=prod(2,3)
```

A függvény 6-os visszatérési értéke bekerül a *product* változóba.

7.1.6 Ciklusok

A ciklusok lehetővé teszik, hogy egy kódot többször végrehajtsunk.

for ciklus

A *for* ciklust akkor szokás alkalmazni, ha a ciklus elkezdése előtt lehet tudni, hányszor kell majd a ciklusnak lefutnia. Jellemző szintaxis:

```
for (változó=kezdőérték; változó <=végérték; változó++) {
  ciklusmag
}
```

Fontos szerepet tölt be a *változó*, amelyet ciklusváltozónak is szokás hívni. A ciklusmag lefutásának száma a ciklusváltozó kezdő és végértékétől, valamint a változó növelési módjától függ.

A következő példa egy számkitaláló játék alapja lehet.

```
<script type="text/javascript">
  for (i = 1; i <= 10; i++) {
    document.write("<input type=button value=" + i +
                   " onclick=\"alert("+ i +")\"> ")
  }
</script>
```

while és *do-while* ciklusok

Ezeket a ciklusokat akkor szokás alkalmazni, ha nem tudjuk előre, hányszor kell a ciklusmagnak lefutnia. Általában ciklusváltozóra nincs is szükség, a ciklus futási feltétele valamilyen más módon áll össze.

A while ciklus magja addig fut, amíg a feltétel igaz. (Akár „végtelen” ciklus is lehet logikailag. Gyakorlatilag a böngésző egy idő után leállítja a túl sokáig futó szkriptet.)

```
while (feltétel) {
  cikusmag
}
```

A *do-while* ciklus logikája csak abban más, hogy mindenképpen lefut egyszer a ciklusmag, és csak ez után értékelődik ki a feltétel:

```
do {
  cikusmag
} while (feltétel);
```

Megjegyzés: Ellenőrző kérdésnek is beillik a következő trükkös kérdés: Ad-e hibát a következő kód, ha igen, hol, ha nem, mit csinál:

```
for (változó=kezdőérték; változó <=végérték; változó++) {
  ciklusmag
} while (feltétel);
```

Első ötletként mondhatjuk, hogy ilyen ciklust nem lehet írni, pedig a kód simán lefut. A feladat azért becsapós, mert nem a szokásos tagolást alkalmazza. Ha az előző kódot egyetlen sortöréssel módosítjuk a következő formára, látszani fog, hogy ez két független ciklus, a második törzs nélkül (;):

```
for (változó=kezdőérték; változó <=végérték; változó++) {
  ciklusmag
}
while (feltétel);
```

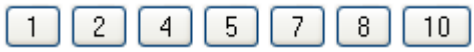
Vezérlésátadás

Nem kevés fejtörést okozott a programozás történetében a strukturálatlan nyelvek *goto* utasítása. Ezért a strukturált nyelvek szűk keretek között teszik csak lehetővé a vezérlésátadó utasítások használatát.

A C alapú nyelvekben a *break* és *continue* utasítások használhatók a ciklus teljes, vagy csak az aktuális ciklusmag futás megszakítására. (A *break* a *switch-case* szerkezetben már bemutatta kiugrási feladatát.)

A következő példa a korábbihoz hasonlóan gombokat hoz létre, de a 3-al osztható számokat kihagyja:

```
<script type="text/javascript">
  for (i = 1; i <= 10; i++) {
    if (i%3 == 0)
      continue;
    document.write("<input type=button value=" + i +
                   " onclick=\"alert("+ i +")\"> ")
  }
</script>
```



for-in ciklus

A *for* ciklust gyakran használjuk arra, hogy valamilyen tároló elem, például tömb minden elemével csináljunk valamit. Ilyen esetekben a szintaxist egyszerűsíteni tudja a *for-in* ciklus. Szintaxis:

```
for (változó in objektum) {
  ciklusmag
}
```

A ciklusmag annyiszor fog lefutni, amennyi az *objektum* elemeinek száma. Sőt a ciklusmagban a *változó* kifejezéssel közvetlenül hivatkozhatunk az éppen bejárt elemre.

A következő példa egy tömb elemeit írja ki egymás alá:

```
<script type="text/javascript">
  var mycars = new Array()
  var x
  mycars[0] = "Saab"
  mycars[1] = "Volvo"
  mycars[2] = "BMW"

  for (x in mycars) {
    document.write(mycars[x] + "<br />")
  }
</script>
```

7.1.7 Eseménykezelés

JavaScript segítségével készíthetünk dinamikus weblapokat. Alapvetően ez azt jelenti, hogy a honlap különböző felhasználói (és böngésző) eseményeket képes érzékelni, és azokra valamilyen módon reagálni.

Jellemző események a következők:

- egérkattintás
- a weboldal vagy egy objektum letöltődött
- az egeret érzékeny területen mozgatjuk
- listából egy elem kiválasztásra kerül
- űrlap elküldés
- billentyűleütés

A teljes lista referenciákban elérhető.

Az egyes események bekövetkezése esetén egy függvénnyel szokás a szükséges funkciót végrehajtani.

onload és *onUnload*

Ezek az események akkor következnek be, ha az oldal betöltése befejeződött, illetve mikor a felhasználó az oldal elhagyását kezdeményezi.

Nézzünk néhány hasznos példát (haszontalant, feleslegeset sajnos sokat találhatunk a weben):

- A sütikben tárolt felhasználói beállításokat (pl. a felhasználó által preferált betűméret) szeretnénk betölteni és elmenteni.
- Oldal elhagyása előtt ellenőrizhetjük, hogy az űrlapban történt-e változás. (Ha van begépelt adat, és véletlen kattintunk egy linkre, elveszhet a begépelt szövegünk.) Ha van változás, csak kérdés után engedjük el az oldalról a felhasználót.
- Oldal betöltődése után (*onload*) a spammer robotok miatt elkódolt e-mail címeket visszakódolhatjuk. (Ha saját, egyedi elkódolást alkalmazunk, a spammer robot programok nem fogják azt ismerni, tehát a kitett e-mail címet nem tudják ellopni.)

onFocus, *onBlur* és *onChange*

Ezek az események elsősorban űrlapok ellenőrzésére, vagy az űrlapok használatának kényelmesebbé tételére szolgálnak.

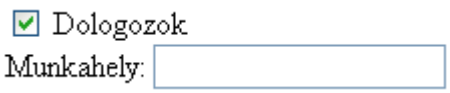
Az *onFocus* akkor következik be, ha az elem megkapja a fókuszt (például kattintás vagy a tab billentyű hatására).

Az *onBlur* az elem elhagyásakor, az *onChange* pedig bármilyen, a bevitt adatban történő változás esetén következik be. Nézzünk egy példát az e-mail cím helyes szintaxisának ellenőrzésére:

```
<input type="text" size="30" id="email"
  onchange="checkEmail()">
```

Összetettebb logikájú űrlapokon hasznos lehet, ha az összefüggéseket vizuálisan is jelezzük a felhasználónak. Például ha a felhasználó egy jelölőnégyzeten kattintva jelzi, hogy van munkája, adjunk lehetőséget a munkahely mező kitöltésére. Egy ehhez hasonló esetben megoldás lehet a következő példa:

```
<input type="checkbox" onclick=
  "document.getElementById('info').style.visibility =
   this.checked ? 'visible' : 'hidden'"> Dologozok
<br>
<div id="info" style="visibility:hidden">
  Munkahely: <input name="munkahely" type="text">
</div>
```



onSubmit

Űrlap elküldése előtt az összes ellenőrzést el lehet végezni ennek az eseménykezelőnek a segítségével. Az eseménykezelő logikai értékével jelezhetjük a böngészőnek, hogy az űrlap elküldhető-e:

```
<form method="post" action="check.php"
  onsubmit="return checkForm()">
```

A *checkForm* függvény visszatérési értékét az űrlap adatai érvényességének függvényében kell beállítanunk.

onClick, *onMouseDown* és *onMouseUp*

Az *onClick* esemény akkor következik be, ha a felhasználó kattint az elemen. Csak akkor érdemes ezt az eseményt kezelni, ha tényleg a kattintáshoz kell kötni egy feladatot, például egy Tili-toli játéknál a mozgatandó kép kijelölésére, vagy egy fényképalbum esetén a mozaikképre kattintva egyből megjelenjen a nagy méretű kép a nézőke dobozban. Az utóbbi példa vázlata:

```
<img src="sbergamot.jpg" alt=" "
  onmousedown="document.images['large'].src='bergamot.jpg'">
<img src="sfoxglove.jpg" alt=" "
  onmousedown="document.images['large'].src='foxglove.jpg'">
<img src="spansy.jpg" alt=" "
  onmousedown="document.images['large'].src='pansy.jpg'">
<br>
<img src="space.jpg" alt=" " id="large">
```

Még speciálisabban alkalmazható az *onMouseDown* és *onMouseUp* esemény kezelése, ami az egérgomb lenyomásakor és felengedésekor következik be. Példaként esetleg játékprogramokat lehetne említeni.

Sok helyen alkalmazott felesleges és rossz gyakorlat, hogy a HTML link helyett is valamelyik eseményt alkalmazzák.

Szintén elavult megoldás az oldalunk menüpontjait képekkel megvalósítani és a képeket az *onMouseDown* és *onMouseUp* események esetén cserélgetni. (CSS-el is megoldható a csere, másrészt az aktivitást jelző kép lassú kapcsolat esetén nem jelenik meg azonnal, ami nagyon zavaró.)

7.1.8 Kivételkezelés

A szkriptjeink futása közben előfordulnak különböző hibák, amelyeket kezelnünk kell. A látogatók se a felbukkanó (számukra) értelmetlen hibaüzeneteket, se a nem működő oldalakat nem szeretik.

A JavaScript nyelvben két lehetőség van a hibák kezelésére. A modern megoldás a kivételkezelés módszerén alapul, a hagyományos pedig az *onerror* esemény kezelésén. (Ebben a jegyzetben elsősorban a korszerűbb megoldással fogunk foglalkozni.)

try-catch szerkezet

A *try-catch* nyelvtani szerkezet a normál kód és a hibakezelő kód szétválasztására épül. Szintaxis:

```
try {
  // normál kód
}
catch(err) {
  // hibakezelő kód
}
```

A *try* blokkban szereplő kód fog futni mindaddig, amíg valamilyen hiba (kivétel) miatt a kód normális futása lehetetlenné nem válik. Ilyen hiba bekövetkezése esetén a *try* blokk futása megszakad, és a *catch* blokkban fog folytatódni. A *catch* kód lefutása a vezérlés soha nem tér vissza a *try* blokkba.

Nézzünk egy egyszerű példát. (Mint a legtöbb egyszerű példa, ez sem túl hasznos ebben a formában.) Tegyük fel, hogy egy függvény nevét elgépeltük: *alert* helyett *adlert*-et gépeltünk. Ekkor a szkript futása a hibás sornál megáll, és a JavaScript értelmező hibát jelez. (Lehet, hogy ezt a felhasználó észre sem veszi.) Ha azonban alkalmazzuk a *try-catch* szerkezetet, a hibáról értesülhetünk.

```
<head><script type="text/javascript">
  var txt=""
  function message() {
    try {
      adlert("Welcome guest!")
    }
    catch(err) {
      txt = "Hiba történt.\n"
      txt += "Hiba leírása: " + err.description + "\n"
      alert(txt)
    }
  }
</script></head>
<body>
  <input type="button" value="Üzenet" onclick="message()" />
</body>
```

Kivétel dobása

Kivételt nem csak elkapni, hanem dobni is tudunk. A *throw* utasítás után meg kell adnunk a kivétel információtartalmát, ami lehet szöveg, szám, logikai érték vagy objektum.

Nézzünk egy ellenőrzött adatbevitel példát:

```
var x=prompt("Írjon be egy számot 0 és 10 között:","")
try {
  if(x>10)
    throw "Err1"
  else if(x<0)
    throw "Err2"
  else if(isNaN(x))
    throw "Err3"
}
catch(err) {
  if(err=="Err1")
    alert("A szám túl nagy!")
  if(err == "Err2")
    alert("A szám túl kicsi!")
  if(err == "Err3")
    alert("Ez nem szám!")
}
```

Ez a példa első ránézésre valószínűleg senkit sem fog meggyőzni arról, hogy a kivételkezelés módszerével kezelje a felmerülő hibákat. Nagyobb bonyolultságú alkalmazások esetén és hosszabb távon a látszólagos többletmunka már inkább kevesebbet fog jelenteni.

7.2. Objektumorientált programozás

A JavaScript objektumorientált nyelv. Mielőtt azonban a részletekbe mennénk, érdemes megjegyezni, hogy a C++, Java, PHP stb. nyelvekkel szemben itt valódi objektumorientáltságról beszélhetünk, míg az előbb említett nyelveket pontosabb lenne osztály-orientált nyelveknek nevezni.

A böngésző JavaScript kód futtatásakor jó néhány objektumot bocsájt a rendelkezésünkre, amiken keresztül a weboldal egyes jellemzőit, részeit érhetjük el, vagy akár manipulálhatjuk is.

Kezdő JavaScript programozóként tehát nem sokban fog különbözni a dolgunk más nyelvi környezetekhez képet, komoly feladatok megoldásához már igen mély JavaScriptes OOP ismeretekre lesz szükségünk.

Definíció: Az objektum különböző adatok gyűjteménye. Tulajdonságokkal és metódusokkal rendelkezik.

Tulajdonságok

A következő példa bemutatja, hogy hogyan lehet egy objektum tulajdonságát lekérdezni:

```
<script type="text/javascript">
  var txt="Hello World!"
  document.write(txt.length)
</script>
```

A futás eredménye *12*.

Metódusok

A metódusok valamilyen tevékenységet hajtanak végre az objektumon. A következő példa egy *String* objektum *toUpperCase* metódusát használja a nagybetűs kiírás érdekében:

```
<script type="text/javascript">
  var str="Hello world!"
  document.write(str.toUpperCase())
</script>
```

Az eredmény: *Hello world!*

Objektumok létrehozása

A JavaScriptben nincsenek klasszikus értelemben vett osztályok. Az objektumokat függvényekkel hozhatjuk létre. Ennek speciális szintaxisa a *new* operátort alkalmazza. A következő példából az is látszik, hogy a függvényen belül a *this* kifejezéssel hozhatunk létre objektum-tulajdonságokat.

```
function osztaly() { //”osztály” létrehozása
  this.tulajdonsag = 5;
}
var objektum = new osztaly(); // objektum példányosítása
alert(objektum.tulajdonsag);
```

A kód eredménye 5.

További források

- Hodicska Gergely *Objektum orientált JavaScript programozás a felszín alatt*[49] című cikke
- Szabó Attila *Hardcore JavaScript*[50] technikák gyűjteménye

7.2.1 Fontosabb objektumok röviden

Ez a fejezet csupán néhány alapvető objektum felületes bemutatására vállalkozhat, elsősorban példákon keresztül. Részletes referencia található pl. a HTMLinfo[51] oldalon.

String

A *String* hosszának lekérdezésére már láttunk példát. A következő kód az *indexOf* metódus segítségével megkeresi egy szöveg első előfordulását:

```
var str="Hello world!"
document.write(str.indexOf("World") + "<br />")
```

Az eredmény 6.

Date

Az aktuális dátum és idő lekérdezését a paraméter nélküli hívással tudjuk megtenni.

```
document.write(Date())
```

Eredmény:

```
Fri Aug 18 2006 15:15:11 GMT+0200
```

[49] http://weblabor.hu/cikkek/oojsafelszinalatt

[50] http://javascript.w3net.eu/

[51] http://htmlinfo.polyhistor.hu/js13ref/contents.htm

A *Date* objektumtól további információkat kérhetünk. Például a hét napját a következő módon írhatjuk ki a *getDay* metódus segítségével:

```
var d=new Date()
var weekday=new Array(7)
weekday[0]="vasárnap"
weekday[1]="hétfő"
weekday[2]="kedd"
weekday[3]="szerda"
weekday[4]="csütörtök"
weekday[5]="péntek"
weekday[6]="szombat"
document.write("Ma " + weekday[d.getDay()] + " van.")
```

Az eredmény:

```
Ma péntek van.
```

Hasonló módon használhatjuk például a *getHours*, *getMinutes*, *getSeconds* metódusokat.

A dátum belső formátumáról érdemes tudni, hogy valós számként kerül ábrázolásra, vagyis számolhatunk is vele. A következő példa a mai napi dátumhoz képest 5 napot előre számol:

```
var myDate=new Date()
myDate.setDate(myDate.getDate()+5)
```

Array

A következő példában is látszik, hogyan tudunk tömböt létrehozni, és hogyan tudjuk azt egyszerű *for-in* ciklussal bejárni.

```
var x
var mycars = new Array()
mycars[0] = "Saab"
mycars[1] = "Volvo"
mycars[2] = "BMW"

for (x in mycars) {
document.write(mycars[x] + "<br />")
}
```

Érdemes megfigyelni, hogy a *for-in* ciklus az x kifejezéssel az éppen bejárt elem indexét kapjuk.

Lehet tömböket összefűzni a *concat* metódussal, *String*-gé összefűzni egy tömb elemeit a *join* metódussal, vagy éppen a *sort*-tal.

7.3. HTML DOM

A HTML *Document Object Model* egy olyan szabvány, amelynek segítségével a böngésző a HTML dokumentum struktúráját faszerkezetben ábrázolja és teszi elérhetővé, manipulálhatóvá a JavaScript számára.

Megjegyzés: A DOM három részre osztható: Core DOM, XML DOM és HTML DOM. Ezen kívül megkülönböztetünk Level 1/2/3 DOM-okat. :)

A fastruktúra minden pontja egy objektum, ami egy elemet reprezentál. Az elemek tartalmazhatnak szöveget és tulajdonságokat is, amit az objektumokból ki is lehet nyerni.

A következő példa az oldalon való kattintás hatására sárga hátteret állít be:

```
<head>
  <script type="text/javascript">
    function ChangeColor() {
      document.body.bgColor="yellow"
    }
  </script>
</head>
<body onclick="ChangeColor()">
  Klikkeljen az oldalon!
</body>
```

A *document* objektum tartalmazza az összes további objektumot, amelyek együttesen a HTML dokumentumot reprezentálják. A *document.body* objektum a *body* tagnak felel meg.

Az objektumok tulajdonságai egyszerűen elérhetők, illetve manipulálhatók. Az előző példában látott *document.body.bgColor* az oldal (*body*) háttérszínét képviseli.

A következő példa eredményében ugyanazt nyújtja, mint az előző, azonban a megközelítése más. A *style* tulajdonság a CSS-ben beállítható tulajdonságokat foglalja össze:

```
document.body.style.background="yellow"
```

A fejezet hátralévő részében különböző objektumok fontosabb szolgáltatásait (metódusait) fogjuk áttekinteni példákon keresztül.

7.3.1 *getElementById*

Az egyik leggyakrabban alkalmazott metódus, hiszen ennek segítségével lehet a HTML dokumentum egy adott elemét elérni, azon például valamilyen manipulációt elérni. A következő példából fontos megfigyelni, hogy csak az *id*-vel rendelkező elemeket érhetjük el a segítségével. A következő példa egy link elemet keres meg, majd több jellemzőjét megváltoztatja:

```
<body>
  <a id="micro" href="http://www.microsoft.com"> Microsoft</a>
  ...
```

A JavaScript kód:

```
var link = document.getElementById('micro')
link.innerHTML="Visit W3Schools"
link.href="http://www.w3schools.com"
link.target="_blank"
```

A *getElementById* metódussal megkapott objektum további érdekes szolgáltatásokat nyújt: például egy elem megkaphatja a fókuszt a *focus* metódus meghívásával, vagy éppen elvesztheti a *blur* metódus hatására. A *focus* praktikus lehet, ha a JavaScriptet űrlap adatok ellenőrzésére használjuk. Ekkor a hibaüzenet megjelenítése után vissza lehet küldeni a kurzort, a hibát tartalmazó beviteli mezőre, így egy kattintást megspórol a felhasználó.

Megjegyzés: Talán feleslegesnek tűnik az ilyen aprólékos kódolás, de sokszor az ilyen apró praktikus segítségek miatt érzi azt a felhasználó, hogy ez egy barátságos oldal.

7.3.2 A *document* objektum

A *document* objektum segítségével elsősorban az egész dokumentumra vonatkozó információkat tudunk kinyerni, vagy módosítani. A következő példa az oldal címét (*title*) szúrja be az oldal szövegébe is:

```
<head>
  <title>Cím</title>
</head>
<body>
  <p>Az oldal címe:
  <script type="text/javascript">
    document.write(document.title)
  </script>
  </p>
</body>
```

Érdemes megjegyezni, hogy a *write* metódus használata már elavult szemléletű, ilyen alkalmazása nem javasolt. Helyette általában az elemet szokás valamilyen módon „megkeresni” (pl. *getElementById*), majd az *innerHTML* segítségével a szövegét elérni.

Elavult szolgáltatások

A *document* nyújt még más olyan szolgáltatásokat is, amelyek elavultnak tekinthetők. Vannak ugyanis olyan tagjai, amelyek bizonyos gyakran hivatkozott elemeket egy-egy tömbben adják vissza. Például a *document.forms* tömb az oldalon található összes űrlapot indexelve teszi elérhetővé. Ezeknek a használatában az okozhat problémát, hogy ha később változik a HTML oldalunk, és újabb űrlapot szúrunk be az oldal elejére, akkor az összes indexszel hivatkozott űrlap mást fog jelenteni. Ha e helyett az űrlapokat az azonosítójuk segítségével keresnénk meg (*document. getElementById('azon')*), akkor ez soha nem fog nehézséget okozni. Másrészt a kódot később olvasni is egyszerűbb ezzel a megközelítéssel.

7.3.3 Az *Event* objektum

Egérrel vagy billentyűvel kiváltott felhasználói események kezelésekor az eseménykezelő függvény egy esemény objektumot kap paraméterként. Ebből az objektumból lehet kinyerni például, hogy melyik egérgomb volt lenyomva, vagy éppen milyen koordinátákon van az egérkurzor.

7.4. Diszkrét JavaScript

A JavaScript segítségével sok hasznos funkcionalitást adhatunk hozzá oldalainkhoz. Vannak azonban olyan szempontok is, amelyek az okos használatra ösztönöznek.

Ha a JavaScript fut a fejlesztő oldalán, akkor hajlamos azt feltételezni, hogy mindenki másnál is futni fog, pedig ez nincs így. Sok felhasználó fél a biztonsági kockázatoktól, vagy valami más ok miatt kapcsolja ki a JavaScriptet. Ezen kívül a robotok sem értelmezik a JavaScript kódokat, így például a JavaScripttel megoldott navigációt a Google nem fogja figyelembe venni, vagyis az oldal nem lesz kereshető a Google-lel.

A diszkrét JavaScript[52] kifejezés azt az alapelvet jelenti, hogy a csak JavaScripttel használható (tolakodó) szolgáltatások helyett olyan oldalakat készítsünk, amelyek JavaScript nélkül is működnek, esetleg kényelmetlenebbül, extrák nélkül, de mégis működnek. A JavaScript csak a plusz kényelmi szolgáltatásokat adja.

7.4.1 Előugró ablak példa

A téma rövid bevezetéseként nézzünk meg egy feladat többféle megoldását. A feladat egy előugró (popup) ablak nyitása lesz.

Az első megoldás kézenfekvőnek tűnek, de az előző megfontolások miatt nem célravezető:

```
<a href="javascript:window.open('popup.html','popup');">
  popup nyitás</a>
```

A következő megoldás szintén elérhetetlen JavaScript nélkül:

```
<a href="#"
  onclick="window.open('popup.html','popup');return(false);">
  popup nyitás</a>
```

A link a dokumentum elejére mutat (#). JavaScript használata esetén az előugró ablak megnyílik (*window.open*), majd a *return(false);* miatt a böngésző nem fogja a tényleges link célt (#) figyelembe venni. JavaScript nélkül azonban az oldal elejére ugrik a böngésző.

Egy jó megoldás lehet a következő:

```
<a href="popup.html"
  onclick="window.open('popup.html','popup');return(false);">
  popup nyitás</a>
```

Ekkor a JavaScript gondoskodik az előugró ablak megnyitásáról, és *false* visszatérési értékkel megakadályozza, hogy a főoldal a *popup.html*-re ugorjon. JavaScript nélkül pedig a főablak jeleníti meg a *popup.html*-t.

Még diszkrétebb

A diszkrét JavaScript elvével még ennél is tovább mehetünk. A JavaScript kód HTML oldalba keverése ugyanúgy nem praktikus, mint ahogy a HTML tartalom és a megjelenítés mikéntjét definiáló CSS összekeverése sem az. Célszerű tehát a JavaScriptet a lehető legjobban elkülöníteni a HTML dokumentumtól.

Megjegyzés: Ennek az elvnek megnyilvánulása az a cél, hogy a JavaScript kódunk jelentős része függvényekben, azok pedig külön *.js* állományokban legyenek.

A következő verzió ugyan feleslegesen bonyolultnak tűnhet, de nem szabad elfelejteni, hogy a megoldás gerincét alkotó elveket és függvényeket már mások kidolgozták, nekünk elég felhasználni az ő munkájukat, és hátradőlni a karosszékünkben. :)

A HTML kódon nem is látszik, hogy itt JavaScriptről lenne szó:

```
<a href="popup.html" class="popup">popup nyitás</a>
```

Az egész HTML kódban csak a *.js* állomány betöltése mutatja, hogy itt JavaScript kód fog futni.

A feladatot megoldó JavaScript állomány tartalmaz több függvényt, és a következő sort:

---

[52] Nagyon jó cikk a témához: http://weblabor.hu/cikkek/diszkretjavascript

```
addEvent(window, 'load', classPopupHandler);
```

Ennek a sornak az a feladata, hogy a *window* objektum *load* eseményéhez hozzákapcsoljuk a *classPopupHandler* metódust. (Ennek kevésbé szép megoldása lenne a *body onload="classPopupHandler"* a HTML kódban).

Az *addEvent* metódus és egy segédmetódusa a fejezet későbbi részében kerülnek bemutatásra, itt csak az eseménykezelő függvényt minden egyes *popup* osztályú linkhez kapcsoló *classPopupHandler* metódus és a tényleges eseménykezelő *doPopup* metódus következik.

```
function classPopupHandler() {
  var elems=getElementsByClass('popup');
  for(i=0;i<elems.length;i++) {
    if (elems[i].href && elems[i].href!='') {
      addEvent(elems[i],'click',doPopup);
    }
  }
}
```

Először is a (később bemutatásra kerülő) *getElementsByClass* metódus egy tömbbe kigyűjti a *popup* osztályú elemeket. A ciklusban a megadott *href* értékkel bíró tagokat szűrjük, és hozzárendeljük a *click* eseményhez a *doPopup* metódust.

Megjegyzés: a kevésbé diszkrét megoldás ugye pont azt jelentené, hogy a HTML kód tartalmazza minden helyen az *onclick="..."* szöveget.

Következzék a tényleges eseménykezelő, ami a kattintáskor fog lefutni:

```
function doPopup(ev) {
  ev || (ev = window.event);
```

Az *ev* esemény objektum elvileg a függvény paramétereként áll rendelkezésre, de Internet Explorer alatt ezt nem kapjuk meg, ezért a *window* objektumból kell kinyerni.

```
  var source;
  if (typeof ev.target != 'undefined') {
    source = ev.target;
  } else if (typeof ev.srcElement != 'undefined') {
    source = ev.srcElement;
  } else { return(true); }
```

Megpróbáljuk kideríteni, hogy melyik objektum váltotta ki az eseményt. A többféle próbálkozás itt is a különböző böngészők miatt szükséges.

```
  window.open(source.href,'popup');
```

A tényleges feladat, az ablak megnyitása.

```
  if (ev.preventDefault) {
    ev.preventDefault(); ev.stopPropagation();
   } else {
    ev.cancelBubble = true; ev.returnValue = false;
  }
  return false;
}
```

Végül arról is gondoskodnunk kell, hogy az eseményt még egyszer ne kezelje le a böngésző, vagyis a főablak maradjon eredeti állapotában. (Emlékeztetőül: az eredeti *a* tag *href* tulajdonsága volt a HTML oldal.)

7.4.2 E-mail cím elrejtése[53]

Nagyon felelőtlen dolog egy e-mail címet kitenni egy weboldalra. Úgynevezett spam robotok pont azt keresik, hogy a weben hol vannak kint levő e-mail címek, és a találatokat egy központi szerverre gyűjtik. Az ilyen e-mail gyűjteményeket aztán eladják, és jön a megszámlálhatatlanul sok kéretlen levél. Webfejlesztőként gondoskodnunk kell tehát az illetéktelen hozzáférés megakadályozásáról.

Korábban elterjedt megoldás volt a képek alkalmazása: a képszerkesztővel elkészített e-mail címet a látogató el tudja olvasni, de a spam robot nem (hacsak nem foglalkozik karakterfelismeréssel). Másik elterjedt megoldás a szöveges kódolás, ahol a pont és kukac írásjelek szövegesen jelennek meg. Itt a biztonság ugyan megvan, de kényelmetlen egy ilyen esetben begépelni az e-mail címet, fennáll a tévesztés lehetősége. Ebben a környezetben jön jól a következő megoldás, ami szintén diszkrét JavaScriptre épít.

A megoldás lénye, hogy az e-mail címet egy meghatározott algoritmus szerint kódoljuk, amit a böngészőben futó JavaScript visszaalakít eredeti formájára. Arra is gondolva, hogy a látogató nem biztos, hogy engedélyezi a JavaScript alkalmazását, a kódolt címnek az emberi felhasználó számára értelmezhetőnek kell lenni. A HTML kód tehát lehet például:

```
<a href="mailto:nagy.gusztav#KUKAC#gamf.kefo.hu"
  >nagy.gusztav#KUKAC#gamf.kefo.hu</a>
```

Ha a böngészőben fut a JavaScript, akkor a *#KUKAC#* karaktersorozatot visszacseréljük a @ karakterre.

```
function whatCorrector() {
  var replaces = 0;
  while (document.body.innerHTML.indexOf('#'+'KUKAC#')!=-1
         && replaces<10000) {
```

Az *innerHTML* hozzáfér a teljes *body* szöveghez. Az *indexOf* a cserélendő karaktersorozatot kutatja. Egy biztonsági intézkedés az esetleg valami problémai miatt fennálló végtelen ciklus elkerülése érdekében, hogy legfeljebb 10.000 futtatás engedélyezett.

```
    document.body.innerHTML =
      document.body.innerHTML.replace(
        '#'+'KUKAC#',String.fromCharCode(64));
```

A *replace* a tényleges cserét végzi, bár egy további trükk, hogy a @ karakter nem szerepel a kódban, hanem a *String.fromCharCode(64)* segítségével állítjuk elő.

```
    replaces++;
  }
}
```

Végül a csere elindítása:

```
addEvent(window, 'load', whatCorrector);
```

7.5. Gyakran használt függvények

Ebben a fejezetben[54] olyan függvények következnek, amelyek sokféle feladat esetén alkalmazhatók.

---

[53] A témáról részletesebben: http://weblabor.hu/cikkek/what

[54] Forrás: pl. http://www.dustindiaz.com/top-ten-javascript/

7.5.1 Sütik kezelése

A sütik (cookie-k) a HTTP alapú kérésekben és válaszokban megtalálható szöveges karakterláncok, amelyek az adott webhely különböző weblapjai közötti navigálás és a webhely későbbi meglátogatásának megkönnyítése érdekében állapotadatokat tárolnak.

Bár a sütiket szerver oldalról is kezelhetjük, időnként szükségünk lehet a JavaScript alapú megközelítésre is.

Egy egyszerű példa: egy adott oldal bejelentkezési procedúráját (név, jelszó begépelése) könnyíthetjük meg, ha ezeket elmentjük, és a legközelebbi oldalletöltés esetén automatikusan beléptetjük a látogatót. De maga a session azonosító is sütiben utazik a kliens és szerver között.

A következő HTTP példák segíthetik a megértést. A szerver az első oldal megtekintésekor beállít egy sütit, amit a böngésző minden egyes további kéréskor elküld a szervernek. Az első kérés:

```
GET /cikkek.html HTTP/1.1
HOST: felho.hu
```

Válasz:

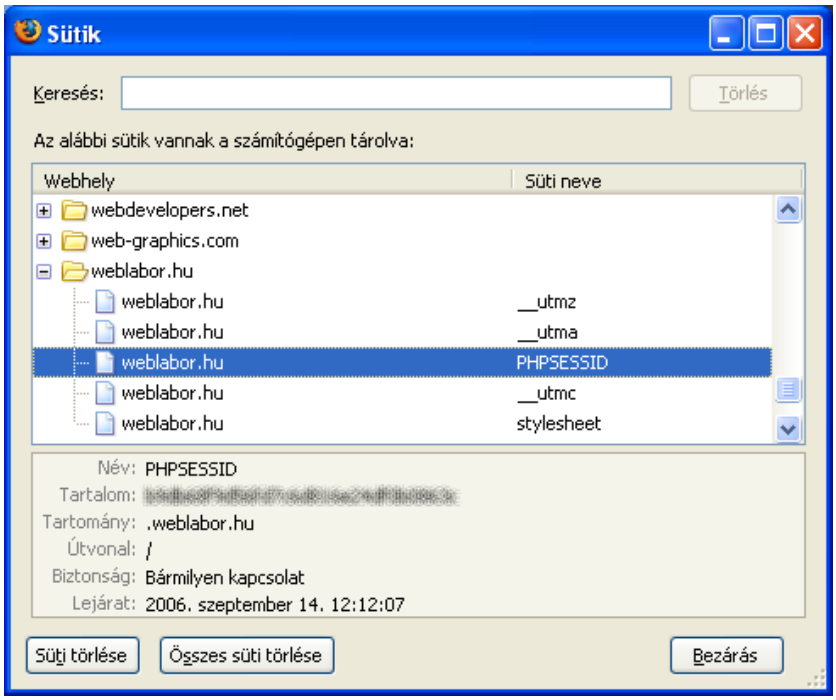
```
HTTP/1.x 200 OK
Server: Apache/2.4.12 (Unix) Debian GNU/Linux PHP/6.0.0
Last-Modified: Sun, 25 Apr 2006 13:57:03 GMT
Set-Cookie: sessionId=krixkrax
Content-Length: 177
Content-Type: text/html; charset=iso-8859-2


<html>
  <head>
    <title>Cikkek</title>
  </head>
  <body>
    <a href="cikk.php?id=1">1. cikk</a>
    <a href="cikk.php?id=2">2. cikk</a>
  </body>
</html>
```

Minden további kérés:

```
GET /cikkek.html HTTP/1.1
HOST: felho.hu
Cookie: sessionId=krixkrax
```

A következő ábra a weblabor.hu domainhez kapcsolt sütik közül a session azonosítót (PHPSESSID) mutatja. (Firefox, Eszközök/Beállítások menü, Sütik fül, Sütik megtekintése.



Megjegyzés: Az ábrán a session azonosító kisatírozva látszik. Ennek megszerzése ugyanis – hozzáértő kezekben – a szerveren levő „személyiségemmel" kapcsolatos visszaélésekre adnak lehetőséget[55].

Sütik kezelésével kapcsolatban két alapvető feladat szokott felmerülni: a süti elmentése és visszatöltése. Nézzük meg ezek egyszerűbb verzióit:

```
function setCookie (name,value) {
  document.cookie=name + "=" + escape(value) +
    ";expires=Thu Feb 10 2028 12:00:00 GMT+0100;path=/;domain="
    + document.domain;
}
```

A sütiket egy adott tartományhoz (domain-hez) és útvonalhoz (path-hoz) rendeljük (ez alapján a böngésző automatikusan tudja elküldeni az adott helyre küldött kérésekkel együtt a hozzá tartozó sütiket is). A példában ez az a domain, ahonnan az oldal érkezett (*document.domain* segítségével megszerezve), valamint a gyökér (/). A süti lejárata a példában 2028-ban lesz (bejelentkezési információkat például tipikusan 30 percig szokás érvényesnek tekinteni). Ahogy a metódus első sorában látható, a *document.cookie* menedzseli a sütiket.

Sütit kiolvasni hasonlóan egyszerű lesz:

```
function getCookie( name ) {
  var name = name + "=";
  var cs = document.cookie.split(';');
```

A sütik egyetlen sztringben tárolódnak, amiben a *;* választja el őket. A *split* metódus darabolja fel a sztringet sütikre.

[55] További információk: http://weblabor.hu/cikkek/munkamenetkezeles2

```
  for (var i=0; i<cs.length; i++) {
    var c = cs[i];
    while (c.charAt(0)==' ') c = c.substring(1,c.length);
    if (c.indexOf(name) == 0)
      return c.substring(name.length,c.length);
  }
  return null;
}
```

Megkeresi a *cs* tömbben szereplő sütik közül a keresettet, és visszaadja az értékét.

7.5.2 *getElementsByClass*

A *getElementById* DOM függvény mintájára hasznos lenne, ha egy adott osztályú elemeket könnyen ki tudnánk gyűjteni. A következő egyszerű verzió csak a keresett osztály nevét várja. Lehetne szűkíteni a keresést, például csak egy típusú elemekre, vagy adott azonosítójú elem leszármazottaiban, stb.

```
function getElementsByClass(name) {
  var found = 0;
  var elems = new Array();
```

Találatok megszámolásához és tárolásához két segédváltozó.

```
  var alltags = document.getElementsByTagName("*");
```

Minden tag lekérdezése. A * helyett lehetne csak bizonyos elemekben is keresni, például *h** a címekben. A visszaadott érték lehet *null* is, ezért ezt ki kell szűrni:

```
  if (alltags) {
    for (i=0; i < alltags.length; i++) {
        if (alltags[i].className==name) {
          elems[found++]=alltags[i];
        }
    }
  }
  return(elems);
}
```

7.5.3 *addEvent*

Ez a függvény egy adott böngésző objektum egy adott eseményéhez csatol hozzá egy adott függvényt, melyeket paraméteréül kap.

```
function addEvent(obj, evType, fn) {
  if (obj.addEventListener) {
    obj.addEventListener(evType, fn, true);
    return true;
  } else if (obj.attachEvent) {
    var r = obj.attachEvent("on"+evType, fn);
    return r;
  } else {
    return false;
  }
}
```

Bár a megoldás böngészőfüggő, azért könnyen átlátható. (A DOM az *addEventListener*-t tartalmazza, szokás szerint az Internet Explorer jár külön utakon.)

7.5.4 *addLoadEvent*

Ez a metódus az előző probléma speciális esetének is tekinthető. Az előző ugyanis a *body* tagra nem minden böngésző alatt működik, míg a következő megoldás igen.

```
function addLoadEvent(func) {
  var oldonload = window.onload;
  if (typeof window.onload != 'function') {
    window.onload = func;
  }
```

Először is megnézzük, hogy van-e eseménykezelő rendelve az *onload*-hoz. Ha nem, egyszerű a dolgunk.

```
  else {
    window.onload = function() {
      oldonload();
      func();
    }
  }
}
```

Ellenkező esetben egy új függvénybe kell összefoglalni az eddigi és az új kódot, hogy az új függvényt eseménykezelővé téve mindkettőt futtassa.

7.6. Felhasználói élmény

A felhasználói élmény elérése a JavaScript korszerű használatának egyik legfontosabb oka. E fejezetben néhány egyszerűbb példán keresztül megvizsgáljuk, milyen módon lehet a felhasználó munkáját könnyebbé, akár élményszerűvé tenni.

7.6.1 Kliens oldali űrlap ellenőrzés

Felhasználóként rendkívül kiábrándító tud lenni, ha egy komolyabb űrlap kitörlése értelmetlenül felesleges munkát okoz.

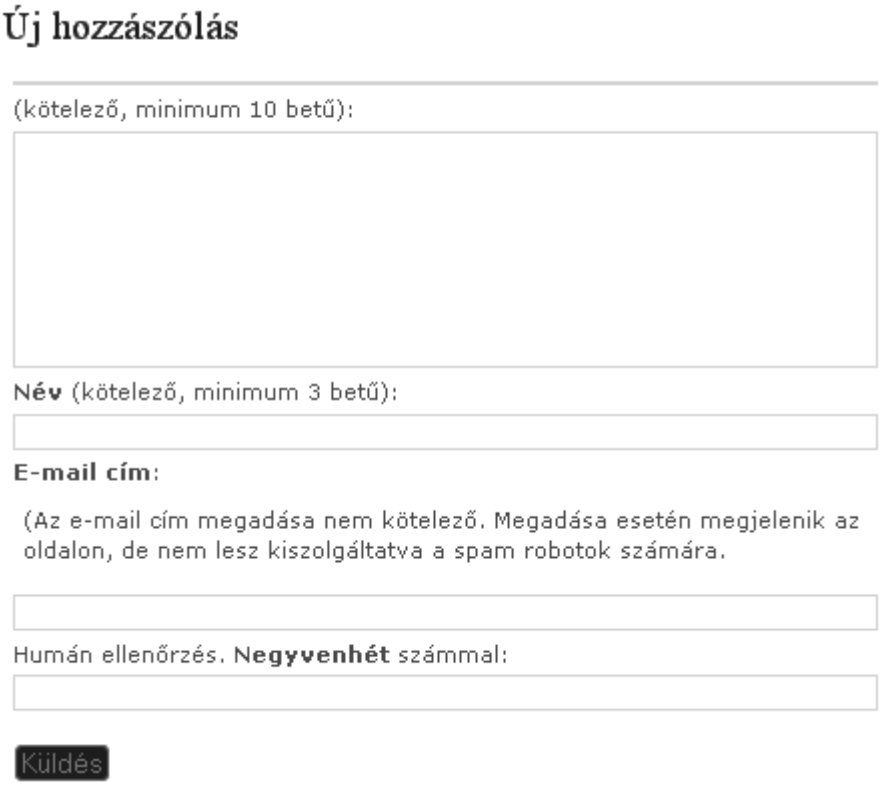
Egy űrlap áttekinthető megtervezése, a magyarázó szövegek pontos megfogalmazása gyakran nehéz feladat. De még jól megtervezett űrlapok esetén is előfordulhat, hogy a felhasználó hibát követ el, egy hosszabb űrlap esetén akár többet is.

Rendkívül rontja a felhasználói élményt, ha a hibákat csak hosszas procedúra után lehet kijavítani: pl. az elküldött adatokban levő első hibát a webalkalmazás hibaüzenettel jelzi, és így a felhasználó akár jó néhányszor kénytelen az adatokat javítani-újraküldeni, hogy egyáltalán a hibaüzenetekkel szembesülhessen. Akinek volt már „szerencséje” ilyen oldallal találkozni, az valószínűleg a szerzőhöz hasonlóan azt is megbánta, hogy rátalált a weboldalra.

Egy űrlap esetén az a minimum, hogy az összes hibaüzenetet egyszerre (és a beviteli elem környékén) mutatjuk meg, de még jobb a megoldásunk, ha az adatok elküldése előtt jelezzük a hibákat. A jelzésre egy praktikus lehetőség, ha a beviteli elem környékén pl. egy pipa megjelenésével, vagy éppen a háttérszín vagy a szegély zöldre állításával jelezzük az adott elem helyes kitöltését.

Az itt következő példa a szerző honlapjának egy 2006-os változatán a hozzászólás beküldésére szolgáló űrlap lesz. Először is érdemes megfigyelni, hogy a felhasználóval nem

egy kitalálós játékot játszunk, hanem a lehető legrészletesebb magyarázatot adjuk a beviteli mezőkhöz:



Az ábrán talán nem egyértelműen látszik, hogy a *Küldés* gomb induló állapotában nem használható, szürke színű. Csak akkor válik használhatóvá, ha a felhasználó minden szükséges mezőt korrekten kitöltött.

HTML kód

```
<h2>Új hozzászólás</h2>
<form id="hozzaszolas" action="index.php" method="post">
  <label for="szoveg">(kötelező, minimum 10 betű):</label>
  <textarea id="szoveg" name="szoveg"></textarea>

  <label for="nev"><strong>Név</strong>
    (kötelező, minimum 3 betű):</label>

  <input id="nev" name="nev" type="text" size="20" value="" />
  <label for="email"><strong>E-mail cím</strong>:</label>

  <p>(Az e-mail cím megadása nem kötelező. Megadása esetén
     megjelenik az oldalon, de nem lesz kiszolgáltatva a spam
     robotok számára.</p>
  <input id="email" name="email" type="text" size="20"
    value="" />

  <label for="human">Humán ellenőrzés.
    <strong>Negyvenhét</strong> számmal:</label>
  <input id="human" name="human" type="text" size="5"
    value="" />

  <button name="hozzaszolaskuld" id="hozzaszolaskuld"
    type="submit" value="kuld">Küldés</button>
</form>
```

Érdemes megfigyelni, hogy a JavaScript kód diszkrét, nem is látszik közvetlenül a HTML kódban. Ha a JavaScript nincs engedélyezve, akkor a kényelmi szolgáltatás nélkül ugyan, de az űrlap elküldhető.

A diszkrét működés érdekében a gomb igazából lenyomható, a következő JavaScript kód teszi használhatatlanná (*disabled*):

```
document.getElementById('hozzaszolaskuld').disabled = true;
```

Ezen kívül minden billentyűfelengedésre beregisztráljuk az *urlapEllenoriz* függvényt:

```
addEvent(document.getElementById('szoveg'),
         'keypress', urlapEllenoriz);
addEvent(document.getElementById('nev'),
         'keypress', urlapEllenoriz);
addEvent(document.getElementById('human'),
         'keypress', urlapEllenoriz);
```

A következő függvény csak a minimális szöveghosszakat ellenőrzi, ízlés szerint továbbfejleszthető pl. e-mail cím szintaxis-ellenőrzésére.

```
function urlapEllenoriz() {
  document.getElementById('hozzaszolaskuld').disabled =
    (document.getElementById('szoveg').value.length<10) ||
    (document.getElementById('nev').value.length<3) ||
    (document.getElementById('human').value.length<1);
}
```

7.6.2 Hosszú listák böngészése helyett

A GAMF-on működő BME Nyelvvizsgahely[56] honlapján minden nyelvvizsga előtt tájékozódniuk kell a vizsgázóknak, hogy pontosan milyen körülmények között (mikor, hol stb.) fog lezajlani a vizsga. Mivel elég sok vizsgázó van, nem triviális, hogy az információt hogyan is nyújtsuk a vizsgázóknak. (Az ábrán csak az információk töredéke látszik.)

	A	B	C	D	E	F	G	H	I	J
1	**Név**	**Nyelv**	**Szint**	**Tipus**	**Ib idő**	**Ib óra**	**Ib hely**	**Magnó idő**	**Magnó óra**	**Magnó hely**
2	[illegible]	Angol	Alap	A				2007.03.30	14::00	GAMF Főép.4
3	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
4	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
5	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
6	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
7	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
8	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
9	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
10	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
11	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
12	[illegible]	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
13	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
14	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
15	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
16	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
17	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
18	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
19	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4
20	[illegible]	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14::00	GAMF Főép.4

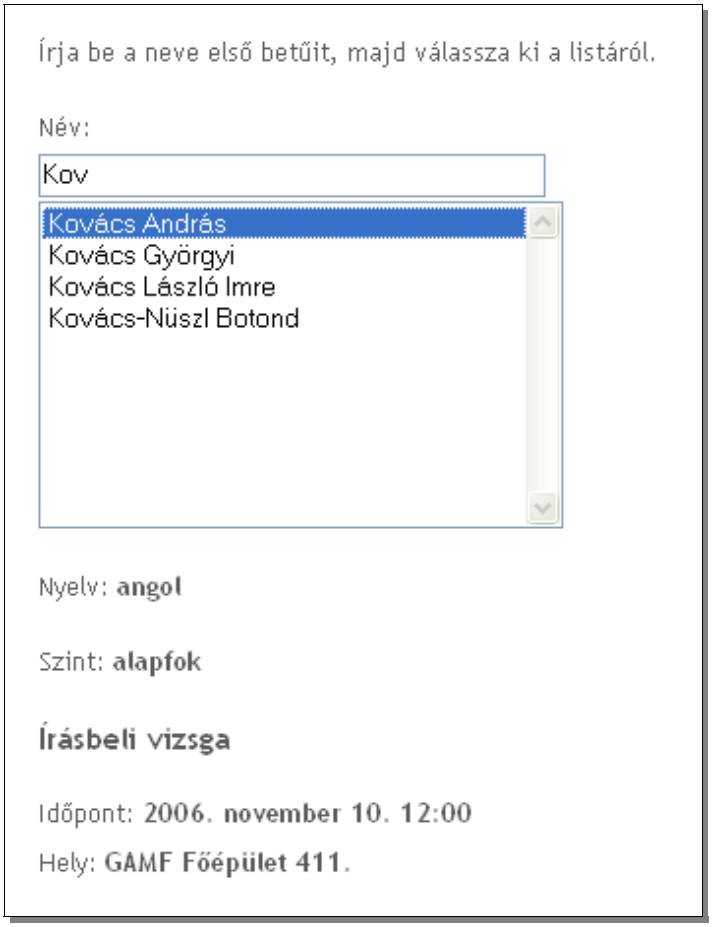
Néhány hagyományos megoldási lehetőség:

- valamilyen offline médium használata (pl. egy Excel táblázat), ebben történik a szervezési információk előállítása és publikálása is

[56] http://lekt.gamf.hu/BME/

- hosszú táblázatos oldal kialakítása, amiben már lehet szövegesen is keresni (nagyon hosszú tartalom esetén a lapozás szükségessé válhat, ami szintén nem túl szép megoldás)

Az itt következő megoldás az előző megközelítések nehézségeit próbálja kiküszöbölni. A Végeredmény működés közben:



A vizsgázónak elegendő a neve 1-3 betűs részét beírnia ahhoz, hogy a sok név közül csak azt a néhányat lássa, amelyikből már könnyedén ki tudja választani a saját nevét. (Példánkon a *Kov* begépelése látszik.)

Második lépésként egyetlen kattintás elegendő, hogy a számára fontos információk jelenjenek meg az oldalon (időpont, helyszín stb.).

Az űrlap felépítése

A megoldás logikája röviden az, hogy a név minden egyes karakterének leütése után automatikusan meghívódik a *gepel()* függvény, ami a találati lista megjelenítéséért felelős.

A névlistán való kattintás esetén a *valaszt()* függvény gondoskodik a keresett adatok megjelenítéséről.

Megjegyzés: a következő kódrészletek nem a forráskódban előforduló sorrendben, hanem a végrehajtás sorrendjében kerülnek bemutatásra.

```
<form method="get" action="#" onSubmit="return false;">
```

Az űrlap tényleges adatküldést nem fog megvalósítani. Erről az *onSubmit="return false;"* gondoskodik.

```
  <label accesskey="n">
    Név:<br />
    <input type="text" id="nev" name="nev" value=""
      onKeyUp="gepel();"></input>
  </label><br />
```

Minden karakter felengedésre (*onKeyUp*) meghívódik a *gepel()* függvény.

```
  <select size="10" id="lista" name="lista"
    onChange="valaszt();">
  </select>
</form>
```

A listára kattintáskor a *valaszt()* fut le.

Szükséges még bemutatni, hogy hol is fognak megjelenni a keresett adatok. Például az írásbeli vizsga adatai:

```
<div id="irasbeli" style="display: none">
  <h2>Írásbeli vizsga</h2>
  <p>Időpont: <span id="iido"></span><br />
    Hely: <span id="ihely"></span></p>
</div>
```

Alapértelmezetten a doboz nem jelenik meg (*display: none*), hiszen nem biztos, hogy a vizsgázóknak lesz egyáltalán írásbeli vizsgája. Másrészt érdemes megfigyelni a (most még) üres *span* elemeket: itt fognak megjelenni a keresett adatok.

Adatok tárolása

Mielőtt a *gepel()* függvényt megvizsgálnánk, át kell gondolnunk, hogy honnan fogja venni a kódunk a megjelenítendő adatokat. Több megközelítés is szóba jöhet, mi itt csak a JavaScript változó használatát vizsgáljuk meg.

(További megoldási lehetőségek az AJAX fejezet alapján kitalálhatók.)

Egyszerűbb esetekben és kisebb adatmennyiség (néhány kb) esetén tökéletesen megfelel az adatok tömbben való tárolása. Jelen megoldás ezt az egyszerű módszert alkalmazza:

```
var vizsgazok = new Array(
  new Array("Antal Alma","a","a","C","7","1","8:45","8"),
  new Array("Bera Berta","n","k","B","","","","  "),
  ...
)
```

Megjegyzés: Jelen fejezetben azzal nem foglalkozunk, hogy ezt a tömböt hogyan is állítsuk elő. Mint lehetőség, meg kell azonban említeni, hogy szerver oldalon nem csak HTML, hanem akár JavaScript kódot is generálhatunk, így nincs annak sem akadálya, hogy PHP-vel az adatbázisból származó adatokat JavaScript tömbbé írjuk ki, hogy a fenti eredményt kapjuk. Még egyszerűbb esetekben a tömb „kézi” módszerekkel is előállítható.

A *gepel* függvény

A *gepel()* függvény néhány fontosabb részlete:

```
function gepel(){
  var nev = document.getElementById('nev')
  var lista = document.getElementById('lista')
```

Kinyerjük a név begépelésére és a névlista megjelenítésére szolgáló objektumot.

```
  var str = ''
```

Az *str* változóban fogjuk felépíteni a névsort. Ha nem üres a név mező, listát generálunk:

```
if (nev.value.length>0) {
  for (i=0; i<vizsgazok.length; i++) {
    var vnev = vizsgazok[i][0];
```

Ha az aktuális név tartalmazza a minta szöveget (*nev.value*), akkor jó nevet találtunk:

```
    if (vnev.indexOf(nev.value) != -1) {
      str += '<option value="o' + i + '">' + vnev + '<'
             + '/option>\n'
    }
  }
```

Ha üres a név mező, törölni kell mindent:

```
} else {
  var szobeli = document.getElementById('szobeli')
  szobeli.style.display = 'none'
  var irasbeli = document.getElementById('irasbeli')
  irasbeli.style.display = 'none'
}
```

Végül egy „technikai” érdekesség: Az Internet Explorer hibásan működik, ha az *innerHTML* adattagot *select* elemre használjuk. Ezért – a nem szabványos – *outerHTML*-t kell alkalmaznunk helyette:

```
if (lista.outerHTML) {
  lista.outerHTML =
    '<select size="10" id="lista" name="lista"'
    .' onChange="valaszt();">' + str + '</select>'
} else {
  lista.innerHTML = str
}
```

A *valaszt* függvény

Az előző függvény alapján sok részfeladat már könnyen elkészíthető.

A kiválasztott név kinyerése:

```
var nev = lista.options[lista.selectedIndex].text
```

Megkeressük, hogy a tömb melyik indexén találhatóak a keresett adatok:

```
for (i=0; i<vizsgazok.length; i++) {
  if (vizsgazok[i][0] == nev) {
    break
  }
}
```

Megjegyzés: A példán jól látszik, hogy a kód csak egyedi nevek esetén működik hibátlanul.

A függvény további része a talált adatok megfelelő beillesztésére szolgál.

További források

- Web Essentials 2005 – Internetes világkonferencia Sydney-ben
  http://www.internet-marketing.hu/we05-1

7.7. Elavult technikák

A JavaScript számos hasznos nyelvi lehetőséggel támogatja munkánkat, de sok elemét elavultnak tekinthetjük. Néhány lehetőség következik, amit érdemes elkerülni, és helyette korszerűbbet alkalmazni.

document.write

A HTML oldalt csupán szövegként tekintve logikus e metódus alkalmazása, de a DOM filozófiája sokkal jobb alternatívákat ad, például *getElementById* vagy *getElementsByTagName* a tartalom keresésére valamint *createElement* és *createTextNode* a tartalom beszúrására.

Megjegyzés: Ezeket a technikákat nagyon jól demonstrálja a Dinamikus betűméret választó[57] cikk is.

noscript

Ezt a HTML tagot korábban döntően arra használták, hogy a látogató tudomást szerezzen arról, hogy az oldal nem fog menni JavaScript nélkül. Jobb esetben valami szerényebb helyettesítő tartalom definiálására alkalmazták.

A korszerű megközelítés a diszkrét JavaScript kapcsán már kifejtésre került.

href="javascript: ", onclick="javascript:"

A JavaScript kódot a lehető legkevésbé keverjük a HTML tartalommal. Pozitív megoldásokra itt is láttunk példákat.

onmouseover="..."

A CSS *:hover* kiválasztója segítségével szebb megoldást kaphatunk.

7.8. További források

Nem egyszerű magyar nyelven jó forrást ajánlani. A ma elérhető források jelentős része a ’90-es évek végén készült, mára nagyon elavult szemléletet közvetít. Egyedül a leírások részletessége miatt érdemes megemlíteni a HTMLInfo anyagait (ahogy az URL-en is látszik, az 1.3-as verzió idején készült):

- Kliensoldali JavaScript Útmutató
  http://htmlinfo.polyhistor.hu/js13guide/contents.htm
- Kliensoldali JavaScript Referencia
  http://htmlinfo.polyhistor.hu/js13ref/contents.htm
- Szabó Attila: Hard Core JavaScript
  http://javascript.w3net.eu

[57] http://weblabor.hu/cikkek/betumeretvalaszto

8. PHP

A PHP (**P**HP: **H**ypertext **P**reprocessor) erőteljes szerver-oldali szkriptnyelv, jól alkalmazható dinamikus és interaktív weboldalak elkészítéséhez. A versenytársakkal szemben (mint például az ASP) széles körben alkalmazott és alkalmazható, szabad és hatékony alternatíva.

A PHP használata esetén a PHP kódot közvetlenül a HTML kódba ágyazhatjuk be. A nyelv szintaxisa nagyon hasonlít a C/C++, Java, Javascript vagy Perl nyelvekhez. Leggyakrabban az Apache webszerverrel használjuk együtt különböző operációs rendszerek alatt.

8.1. Alapok

Egy PHP állomány szöveget, HTML tagokat és PHP kódokat (szkripteket) tartalmazhat.

A PHP kód még a szerveren lefut, mielőtt a webszerver a látogató kérését kiszolgálná. A látogató kérésére egyszerű HTML állománnyal válaszol.

Futása során kommunikálhat adatbázis-szerverrel (pl. MySQL, Informix, Oracle, Sybase, PostgreSQL, ODBC) is.

A PHP állomány szokás szerint *.php* kiterjesztésű, bár a webszerver beállításai akár ettől eltérő megoldásokat is lehetővé tesznek. (Előfordul pl., hogy az éppen aktuális beállítások szerint a *.html* állományokat is értelmezi a PHP motor.)

A PHP nyílt forrású, szabadon letölthető és használható.

8.1.1 Szintaxis

Figyelem! Egy weboldal PHP forrását nem tudjuk a böngészőnk segítségével megnézni. Ott mindig csak a PHP által előállított HTML kimenet látható.

Megjegyzés: A fejezet példaprogramjai – helytakarékossági okokból - a legtöbb esetben nem teljes vagy nem szabványos HTML oldalakat készítenek.

A HTML oldalba ágyazott PHP kód kezdetét és végét egyértelműen jelölni kell a PHP értelmező számára. Ennek többféle módja közül leginkább a következő ajánlható:

```
<?php ... ?>
```

Szokás még használni egy rövidebb jelölést is, de ez nem biztos, hogy minden konfiguráció esetén működik:

```
<? ... ?>
```

A következő „Hello World” példa jól mutatja, hogyan ágyazzuk be a PHP kódot a HTML oldalba:

```
<html>
  <body>
<?php
  echo "Hello World";
?>
  </body>
</html>
```

Majdnem minden PHP utasításnak *;*-vel kell végződnie. (Kivétel: a PHP záró tag előtt elhagyható.) Két egyszerű lehetőségünk van arra, hogy a HTML állományba kimenetet szúrjunk be: *echo* és *print*. A két függvény közül szabadon választhatunk, de az olvashatóság kedvéért nem illik keverni a kettőt egy kódon belül.

Sablonszerű megközelítés

A későbbi, sablonrendszerekről szóló fejezet példái elsősorban a következő példához hasonlóan, *print* és *echo* nélkül kezelik a kimenetet:

```
<acronym class="datum" title="<?= $datum['hosszu'] ?>">
  <span class="honap"><?= $datum['honap'] ?></span>
  <span class="nap"><?= $datum['nap'] ?></span>
</acronym>
```

A példa lényege, hogy nem a PHP-be illesztjük a HTML-t készítő függvénysorokat, hanem fordítva: a fix HTML szövegbe szúrjuk be a PHP kifejezéseket (elsősorban változók beszúrásához szükséges utasításokat). Ehhez az érték kiírására szolgáló szintaxis használható:

```
<?= kifejezés ?>
```

A kifejezés kiértékelődik, és értéke a tag helyére kerül beszúrásra.

Megjegyzések

A PHP kódban egysoros (sorvégi) és többsoros megjegyzéseket is használhatunk:

```
<html>
<body>
<?php
  //This is a comment
  /*
    This is
    a comment
    block
  */
?>
</body>
</html>
```

Nem szabad összekeverni a HTML és a PHP megjegyzések szerepét. A HTML megjegyzés a kliens gépén is megjelenik, „fogyasztja” a sávszélességet, általában nem javasolt. Ezzel szemben a PHP megjegyzések nem kerülnek el a klienshez, az ilyen megjegyzéseket a PHP nem továbbítja a kimenetre.

Egy példa a szerző korábbi honlapjáról:

```
<? /* Az előzetes megjelenítésére, a hozzászólások számával */?>
<div class="hir">
<? if ($lehethozza || $tartalom) { ?>
  <h2><a href="<?= $index ?>?<?= $blognev ?>/<?= $url ?>">
    <?= $hirCim ?></a></h2>
<? } else { ?>
  <h2><?= $hirCim ?></h2>
<? }?>
```

8.1.2 Változók

A PHP nyelvben minden változó neve $ karakterrel kezdődik. A változók alapvetően szövegeket, számokat és tömböket tartalmazhatnak.

A következő példa a *$txt* változót használja a kiírandó szöveg tárolására:

```
<html>
<body>
<?php
  $txt="Hello World";
  echo $txt;
?>
</body>
</html>
```

Ha két vagy több szöveges kifejezést össze akarunk fűzni, akkor a . (pont) operátor használható:

```
<html>
<body>
<?php
  $txt1="Hello World";
  $txt2="1234";
  echo $txt1 . " " . $txt2 ;
?>
</body>
</html>
```

A programunk a következő kimenetet állítja elő:

```
Hello World 1234
```

Változónevek

A változónevek az ABC betűiből, számokból és az aláhúzás (_) karakterből állhatnak, de nem kezdődhetnek számmal.

A nevek szóközt nem tartalmazhatnak, így az összetett változónevek esetén az aláhúzás karaktert (*$my_string*) , vagy a közbülső szó nagy kezdőbetűjét (*$myString*) alkalmazzuk.

Változók típusai

A PHP nyelv a következő típusok használatát teszi lehetővé:

- Logikai
- Egész számok
- Lebegőpontos számok
- Sztringek
- Tömbök
- Objektumok
- Erőforrások
- NULL

8.1.3 Sztringek használata

Sztring létrehozása aposztróffal

A legkönnyebben úgy adhatunk meg egy egyszerű sztringet, hogy aposztrófok (' karakterek) közé tesszük.

```
echo 'Arnold egyszer azt mondta: "I\'ll be back"';
// Kimenet: Arnold egyszer azt mondta: "I'll be back"
echo 'You deleted C:\\*.*?';
// Kimenet: You deleted C:\*.*?
```

Sztring létrehozása idézőjellel

A legfontosabb jellemzője az idézőjeles sztringeknek az, hogy a változók behelyettesítésre kerülnek.

Ha dollár ($) jelet talál a PHP egy sztringben, megkeresi az összes ezt követő azonosítóban is használható karaktert, hogy egy érvényes változónevet alkosson. Ezért kapcsos zárójeleket kell használnunk, ha pontosan meg szeretnénk határozni, meddig tart egy változó.

```
$ingatlan = 'ház';

echo "kertes $ingatlan kerítéssel";
// működik, szóköz nem lehet változónévben

echo "páros $ingatlanszám";
// nem működik, az 's' és további karakterek lehetnek
változónévben

echo "páros ${ingatlan}szám";
// működik, mert pontosan meg van adva a név

echo "páros {$ingatlan}szám";
// működik, mert pontosan meg van adva a név
```

Sztring létrehozása 'heredoc' szintaxissal

Egy másfajta módszer a sztringek megadására a heredoc szintaxis ("<<<"). A <<< jelzés után egy azonosítót kell megadni, majd a sztringet, és végül az azonosítót még egyszer, ezzel zárva le a sztringet.

A lezáró azonosítónak mindenképpen a sor legelső karakterén kell kezdődnie. Ugyancsak figyelni kell arra, hogy ez az azonosító is az általános PHP elemek elnevezési korlátai alá tartozik.

```
$str = <<<VAS
Példa egy stringre, amely
több sorban van, és heredoc
szintaxisú
VAS;
```

Sztring karaktereinek elérése és módosítása

A string karaktereire nullától számozott indexekkel lehet hivatkozni, a string neve után megadott kapcsos zárójelek között.

```
$str = 'Ez egy teszt.';
$first = $str{0};
$harmadik = $str{2};
```

8.1.4 Operátorok

Az operátorok műveleteket végeznek az operandusaikkal.

A fejezetben egy gyors összefoglaló következik az operátorokról.

Aritmetikai operátorok

Megjegyzés: A táblázat példái nem tartalmaznak ;-t, mivel itt nem komplett kódrészletekről, csupán az operátorokat tartalmazó kifejezések eredményéről van szó.

Operátor	Művelet	Példa	Eredmény
+	Összeadás	$x=2 $x+2	4
-	Kivonás	$x=2 5-$x	3
*	Szorzás	$x=4 $x*5	20
/	Osztás	15/5 5/2	3 2.5
%	Maradékos osztás maradéka	5%2 10%8 10%2	1 2 0
++	Növelés	$x=5 $x++	$x=6
--	Csökkentés	$x=5 $x--	$x=4

Figyelem! Az osztás eredménye mindig valós, tehát soha nem maradékos osztás történik.

Értékadó operátorok

Operátor	Példa	Jelentés
=	$x=$y	$x=$y
+=	$x+=$y	$x=$x+$y
-=	$x-=$y	$x=$x-$y
*=	$x*=$y	$x=$x*$y
/=	$x/=$y	$x=$x/$y
%=	$x%=$y	$x=$x%$y

Összehasonlító operátorok

Operátor	Művelet	Példa	Eredmény
==	egyenlő	5==8	false
!=	nem egyenlő	5!=8	true
>	nagyobb	5>8	false
<	kisebb	5<8	true
>=	nagyobb-egyenlő	5>=8	false
<=	kisebb-egyenlő	5<=8	true
===	típus és érték egyenlő	”5”===5	false

Logikai operátorok

Operátor	Művelet	Példa	Eredmény
&&	és	$x=6 $y=3 ($x < 10 && $y > 1)	true
||	vagy	$x=6 $y=3 ($x==5 || $y==5)	false
!	nem	$x=6 $y=3 !($x==$y)	true

8.1.5 Elágazások

Az *if*, *elseif* és *else* kulcsszavak használatosak a PHP-ben feltételhez kötött utasítások végrehajtására.

Feltételes kifejezések

Sok esetben merül fel egy-egy programozási feladat kapcsán annak szükségessége, hogy különféle lehetőségekhez más-más viselkedést rendeljünk. Ennek megoldására használhatjuk a feltételes kifejezéseket.

if-else - Akkor használjuk, ha a feltétel igaz volta esetén egy adott kódot, hamis volta esetén egy másik kódot szeretnénk futtatni.

elseif - Az *if...else* kifejezéssel együtt használjuk, ha azt szeretnénk, hogy egy adott kódrészlet akkor fusson, ha több feltétel közül csak egy teljesül.

Az *if...else* szerkezet

Egy feltétel teljesülése esetén futtat egy kódrészletet, amennyiben a feltétel nem teljesül, akkor egy másikat. Szintaxis:

```
if (feltétel)
    végrehajtandó kód, ha a feltétel teljesül;
else
    végrehajtandó kód, ha a feltétel nem teljesül;
```

A következő példa kimenete a *Have a nice weekend!* szöveg lesz, ha épp péntek van, egyébként pedig a *Have a nice day!*:

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
    echo "Have a nice weekend!";
else
    echo "Have a nice day!";
?>
</body>
</html>
```

Ha egy sornyinál több kódot szeretnénk futtatni a feltétel teljesülése/nem teljesülése mellet, a sorokat kapcsos zárójelek közé kell írnunk:

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
{
    echo "Hello!<br />";
    echo "Have a nice weekend!";
    echo "See you on Monday!";
}
?>
</body>
</html>
```

Az *elseif* szerkezet

Ha több feltételből egy teljesülés esetén szeretnénk adott kódot futtatni, az *elseif* szerkezetet használjuk. Szintaxis:

```
if (feltétel)
    végrehajtandó kód, ha a feltétel teljesül;
elseif (feltétel)
    végrehajtandó kód, ha a feltétel teljesül;
else
    végrehajtandó kód, ha a feltétel nem teljesül;
```

Az alábbi példa kimenete *Have a nice weekend!* ha ma péntek van, *Have a nice Sunday!* ha ma vasárnap van, egyébként pedig *Have a nice day!*:

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
    echo "Have a nice weekend!";
elseif ($d=="Sun")
    echo "Have a nice Sunday!";
else
    echo "Have a nice day!";
?>
</body>
</html>
```

8.1.6 A *switch* szerkezet

A *switch* utasítást a PHP-ben akkor használjuk, hogyha más-más kódot szeretnénk végrehajtani különböző feltételek alapján. A *switch* használatával bizonyos esetekben elkerülhetőek a hosszú *if..elseif..else* blokkok. A *switch* utasítás általános formája:

```
switch (kifejezés) {
case cimke1:
  a végrehajtandó kód, ha a kifejezés = cimke1;
  break;
case cimke2:
  a végrehajtandó kód, ha a kifejezés = cimke2;
  break;
default:
  a végrehajtandó kód
  ha a kifejezés különbözik
  cimke1 és cimke2-től;
}
```

Működése:

- Egy egyszerű kifejezés (leggyakrabban egy változó) egyszer kiértékelődik
- A kifejezés értéke összehasonlításra kerül mindegyik *case* kifejezéssel a blokkon belül
- Ha egyezést talál, akkor az adott *case*-ben található kód végrehajtódik
- Miután ez megtörtént, a *break* utasítás leállítja az utasítások végrehajtását
- A *default* utasítás arra használható, hogy le tudjuk kezelni azokat a kifejezéseket, értékeket is, amelyek egy *case* utasításban sem voltak kezelve

Egy példa:

```
<html>
<body>
<?php
  switch ($x) {
    case 1:
      echo "Number 1";
      break;
    case 2:
      echo "Number 2";
      break;
    case 3:
      echo "Number 3";
      break;
    default:
      echo "No number between 1 and 3";
  }
?>
</body>
</html>
```

Megjegyzés: Más nyelvektől eltérően a *case* értékei nem kell, hogy konstansok legyenek.

8.1.7 Tömbök

A tömb egy vagy több értéket tárolhat egyetlen változónévvel.

Ha PHP-vel dolgozunk, akkor előbb vagy utóbb valószínűleg szeretnénk több hasonló változót létrehozni. Több változó használata helyett az adatot tömb elemeiként is tárolhatjuk. A tömb minden elemének egyedi azonosítója van, így könnyen hozzáférhető.

A tömböknek három[58] különböző típusa van:

- **Numerikus tömb –** Olyan tömb, amelynek egyedi numerikus azonosítókulcsa van
- **Asszociatív tömb –** Olyan tömb, melyben minden azonosítókulcshoz egy érték társul
- **Többdimenziós tömb** – Olyan tömb, mely egy vagy több tömböt tartalmaz

Numerikus tömbök

A numerikus tömb minden eleméhez egy numerikus azonosítókulcsot tárol. A numerikus tömb létrehozásának több módja van.

Ebben a példában az azonosítókulcsok (indexek) automatikusan kerülnek hozzárendelésre az elemekhez:

```
$names = array("Peter","Quagmire","Joe");
```

Ebben a példában azonosítókulcsot manuálisan rendelünk hozzá:

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

Az azonosítókulcsokat felhasználhatjuk szkriptekben:

```
<?php
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
echo $names[1] . " and " . $names[2] .
" are ". $names[0] . "'s neighbors";
?>
```

Az előző kódrészlet kimenete:

```
Quagmire and Joe are Peter's neighbors
```

Tömbök feltöltésekor az index megadása sem kötelező. Ha elhagyjuk, akkor a legnagyobb használt index után folytatja a számozást. Az előző példát folytatva a 3-as index alá kerül *Jack*:

```
$names[] = "Jack";
```

Megjegyzés: ahogy az eddigiek alapján sejthető, az index-számok nem csak folytonosak lehetnek.

[58] Hamarosan pontosítani fogjuk: igazából egyféle tömb van, de háromféle tipikus felhasználás szokott előfordulni.

Asszociatív tömbök

Az asszociatív tömb minden azonosítókulcsához egy értéket rendelünk hozzá. Ha egyedi értékeket szeretnénk tárolni, nem feltétlenül a numerikus tömb a legjobb megoldás erre. Az asszociatív tömb értékeit kulcsként használhatjuk, s értékeket rendelhetünk a kulcsokhoz.

Ebben a példában tömböt használunk, hogy különböző személyekhez életkorokat rendeljünk:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Ez a példa az előzővel azonos értékű, de a tömb létrehozásának más módját mutatja:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

Az azonosítókulcsot így használhatjuk szkriptben:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
echo "Peter is " . $ages['Peter'] . " years old.";
```

Az előző kód kimenete:

```
Peter is 32 years old.
```

Megjegyzés: Bizonyos értelemben pontosabb lenne úgy fogalmazni, hogy a PHP csak az asszociatív tömböket ismeri, és mellesleg a kulcs lehet szám is.

Többdimenziós tömbök

Egy többdimenziós tömbben a fő tömb minden eleme lehet maga is egy tömb. Az altömb minden eleme lehet szintén tömb, és így tovább.

Ebben a példában olyan többdimenziós tömböt hozunk létre, melynek azonosítókulcsai automatikus hozzárendeléssel rendelkeznek:

```
$families = array (
  "Griffin"=>array (
    "Peter",
    "Lois",
    "Megan"
  ),
  "Quagmire"=>array (
    "Glenn"
    ),
  "Brown"=>array (
    "Cleveland",
    "Loretta",
    "Junior"
  )
);
```

A fenti tömb így nézne ki a kimenetre írva (*print_r($families)*):

```
Array (
  [Griffin] => Array (
    [0] => Peter
    [1] => Lois
  ),
  [Quagmire] => Array (
    [0] => Glenn
  ),
  [Brown] => Array (
    [0] => Cleveland
    [1] => Loretta
    [2] => Junior
  )
)
```

A szerző oldalának 2006-os verziójában a honlap szerkezete a következő tömb alapján generálódik:

```
$oldalak = array (
  'Személyes' => array(
    'elerhet' => array(
      'cim' => 'Elérhetőségek',
      'tipus' => 'statikus'
    ),
    'magamrol' => array(
      'cim' => 'Magamról',
      'tipus' => 'statikus'
    )
    // ...
  ),
  'Oktatás' => array(
    'java' => array(
      'cim' => 'Java programozás',
      'tipus' => 'blog',
      'felirat' => 'Tantárgyi információk',
    ),
    // ...
  ),
  // ...
);
```

8.1.8 Ciklusok

A ciklusokat a PHP-ben ugyanazon programrész adott számú futtatására használjuk. Mikor programozunk, nagyon gyakran fordul elő, hogy egy adott programrészt többször szeretnénk lefuttatni. Ezt elérhetjük ciklusok használatával.

PHP-ben a következő ciklusok használhatók:

- *while* – lefuttat egy programrészt, ha a feltétel igaz, és addig amíg az adott feltétel igaz
- *do...while* – egyszer lefuttatja az adott programrészt és megismétli a ciklust ha a feltétel igaz, és addig, amíg egy adott feltétel igaz
- *for* – az adott programrészt meghatározott számban futtatja le

- *foreach* – az adott programrészt a tömb minden elemére lefuttatja

A *while* utasítás

A *while* utasítás lefuttat egy programrészt **ha és amíg** egy adott feltétel igaz. Szintaxis:

```
while (feltétel)
  ciklusmag
```

Az alábbi példa egy olyan ciklust mutat be, mely addig fut, amíg az *i* változó kisebb vagy egyenlő, mint 5. Az *i* értéke a ciklus minden lefutása után 1-el növekszik:

```
<html>
<body>
<?php
  $i=1;
  while($i<=5) {
    echo "The number is " . $i . "<br />";
    $i++;
  }
?>
</body>
</html>
```

A *do...while* ciklus

A *do...while* utasítás **legalább egyszer** lefuttat egy adott programrészt, majd annyiszor ismétli azt, **amíg** egy adott feltétel igaz. Szintaxis:

```
do {
  code to be executed;
} while (feltétel);
```

Az alábbi példaprogram legalább egyszer növeli az *i* változó értékét, majd addig növeli *i*-t, amíg annak értéke kisebb, mint 5.

```
<html>
<body>
<?php_
  $i=0;
  do {
    $i++;
    echo "The number is " . $i . "<br />";
  } while ($i<5);
?>
</body>
</html>
```

A *for* utasítás

A *for* utasítást akkor használjuk, amikor előre tudjuk, hányszor kell futtatni egy adott utasítást vagy utasítás listát. Szintaxis:

```
for (initialization; condition; increment) {
  code to be executed;
}
```

Megjegyzés: A *for* utasításnak három paramétere van. Az első ad értéket a változóknak, a második tartalmazza a feltételt és a harmadikban van az inkrementáló utasítás, ami a ciklust vezér-

li. Ha több mint egy változó is szerepel az értékadásban vagy az inkrementáló részben, vesszővel válasszuk el őket. A feltétel igaznak vagy hamisnak bizonyul a futás során.

Az alábbi példa ötször írja a képernyőre a *Hello World!* szöveget:

```
<html>
<body>
<?php
  for ($i=1; $i<=5; $i++) {
    echo "Hello World!<br />";
  }
?>
</body>
</html>
```

A *foreach* utasítás

A *foreach* utasítást tömbelemekre alkalmazzuk.

A *$value* minden ciklusban az adott tömbelem értékét kapja meg, és a tömb mutató eggyel továbblép – így a következő ciklusban már a következő tömbelemre mutat. Szintaxis:

```
foreach ($array as $value) {
    ciklusmag
}
```

Megjegyzés: A *$value* értéke alapértelmezetten egy másolata a tömb elemének.

A következő példa azt a ciklust mutatja be, amely a képernyőre írja az adott tömb értékeit.

```
<html>
<body>
<?php
  $arr=array("one", "two", "three");
  foreach ($arr as $value) {
    echo "Value: " . $value . "<br />"; }
?></body>
</html>
```

Bizonyos esetekben szükségünk lehet nemcsak a tömb elemeire, hanem a kulcsokra is. Ekkor a következő szintaxis alkalmazható:

```
foreach (tömb as kulcs =>tömbelem) {
    ciklusmag
}
```

A szerző oldalán a korábban bemutatott kétdimenziós tömb alapján a következő kód generálja a főmenüt:

```
<? foreach ($fomenu as $cim => $menublokk) { ?>
        <h2><?= $cim ?></h2>
        <ul>
<?   foreach ($menublokk as $azon => $menupont) { ?>
<?     if ($aktualisFomenu == $azon) { ?>
          <li><?= $menupont['cim'] ?></li>
<?     } else { ?>
          <li><a <?
         if ($menupont['felirat']) { ?>title="<?=
           $menupont['felirat'] ?>" <?
         } ?> href="<?= $index ?>?<?= $azon ?>"><?=
           $menupont['cim'] ?></a></li>
<?     } ?>
<?   } ?>
        </ul>
<? } ?>
      </div>
```

8.1.9 Függvények

A PHP igazi ereje a függvényeiből ered. A PHP-ben közel ezer beépített függvény érhető el.

Ebben a fejezetben megmutatjuk, hogyan készítsünk saját függvényeket.

PHP függvény készítése

Egy függvény tulajdonképpen egy kód blokk, amit akkor futtathatunk, amikor csak akarjuk.

Minden függvény így kezdődik: *function()*

A függvény nevét lehetőleg úgy válasszuk meg, hogy utaljon rá, milyen feladatot végez el a függvény. Ezen kívül betűvel kezdődjön vagy alsó vonással, de ne számmal.

Aztán írjunk egy *{* jelet. A függvény kód a nyitó kapcsos zárójel után kezdődik. Aztán megírjuk a függvény kódját, végül írjunk egy *}* jelet. A függvény kódját mindig egy kapcsos zárójel zárja. Példa:

Egy egyszerű függvény, amely a nevemet írja ki, ha meghívjuk:

```
<html>
<body>
<?php
function writeMyName() {
  echo "Kai Jim Refsnes";
}
writeMyName();
?>
</body>
</html>
```

Egy függvény használata

Most az iménti függvényt használjuk egy PHP szkriptben:

```
<html>
<body>
<?php
function writeMyName() {
  echo "Kai Jim Refsnes";
}

echo "Hello world!<br />";
echo "My name is ";

writeMyName();
echo ".<br />That's right, ";
writeMyName();
echo " is my name.";
?>
</body>
</html>
```

A felső kódnak ez lesz a kimenete:

```
Hello world!
My name is Kai Jim Refsnes.
That's right, Kai Jim Refsnes is my name.
```

Függvények paraméterezése

Első függvényünk (*writeMyName*) egy igen egyszerű függvény. Pusztán egy egyszerű szöveget ír ki.

Ha szeretnénk több funkcióval ellátni egy függvényt, paramétereket adhatunk hozzá. A paraméter olyan mint egy változó. A paramétereket a zárójelen belül adhatjuk meg.

A következő példa különböző keresztneveket ír ki azonos vezetéknevek után. (angol nyelvű példa lévén a keresztnév van előbb.)

```
html>
<body>
<?php
function writeMyName($fname) {
  echo $fname . " Refsnes.<br />";
}

echo "My name is ";
writeMyName("Kai Jim");
echo "My name is ";
writeMyName("Hege");
echo "My name is ";
writeMyName("Stale");
?>
</body>
</html>
```

A kód kimenete a következő:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes.
My name is Stale Refsnes.
```

A következő függvénynek két paramétere van:

```
<html>
<body>
<?php
function writeMyName($fname,$punctuation) {
  echo $fname . " Refsnes" . $punctuation . "<br />";
}
```

```
echo "My name is ";
writeMyName("Kai Jim",".");
echo "My name is ";
writeMyName("Hege","!");
echo "My name is ";
writeMyName("Ståle","...");
?>
</body>
</html>
```

A kód kimenetet a következő lesz:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes!
My name is Ståle Refsnes...
```

Függvények visszatérési értéke

A függvények használhatnak visszatérési értékeket is.

```
<html>
<body>
<?php
function add($x,$y) {
  $total = $x + $y;
  return $total;
}

echo "1 + 16 = " . add(1,16)
?>
</body>
</html>
```

A fenti kód kimenete ez lesz:

```
1 + 16 = 17
```

A szerző 2006-os oldalán a következő függvény segítségével kereshetők meg az egy hírhez (blogbejegyzéshez) tartozó hozzászólások:

```
  function hozzaszolasok($kod) {
    $eredmeny = mysql_query (
      "SELECT * FROM hozzaszolas WHERE kod='{$kod}' ");

    if ($eredmeny) {
      $sorok = array();
      while ($sor = mysql_fetch_array($eredmeny)) {
        $sorok[] = $sor;
      }
      return $sorok;
    } else {
      return 0;
    }
    mysql_free_result($eredmeny);
  }
```

8.1.10 Űrlapok és felhasználói adatbevitel

A PHP *$_GET* és *$_POST* változói használatosak az információ űrlapokból való továbbítására.

Minden HTML oldalon található űrlapelem automatikusan elérhető a PHP szkriptek számára. Példa:

```
<html>
<body>
  <form action="welcome.php" method="post">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
  </form>
</body>
</html>
```

A fenti példaoldal két beviteli mezőt és egy továbbító gombot tartalmaz. Ha a felhasználó kitölti a szövegmezőket, és a gombra kattint, az űrlap adatai a *welcome.php* fájlhoz továbbítódnak.

A *welcome.php* fájl tartalma a következő:

```
<html>
<body>
  Welcome <?php echo $_POST["name"]; ?>.<br />
  You are <?php echo $_POST["age"]; ?> years old.
</body>
</html>
```

A fenti szkript egy lehetséges kimenete:

```
Welcome John.
You are 28 years old.
```

A *$_GET* és *$_POST* változók használatáról hamarosan több szó esik.

Adatok érvényessége

A felhasználó által bevitt adatok érvényességét minden esetben vizsgálni kell. A kliensoldali vizsgálat gyorsabb és csökkenti a szerver terheltségét, de önmagában sosem elegendő. Ezért az adatok szerver oldali vizsgálata is szükséges, különösen, ha az űrlapnak egy adatbázishoz kell hozzáférnie.

Az adatok ellenőrzésére bevett módszer, hogy az űrlap adatait az űrlapot is tartalmazó szkript dolgozza fel, így nincs szükség arra, hogy az adatokat egy újabb oldalra küldjük. Ebben az esetben a felhasználó a hibaüzeneteket ugyanazon az oldalon láthatja, ahol az űrlap található, így megkönnyítve a hiba felfedezését és kijavítását.

Megjegyzés: Egy komplex megoldás is bemutatásra kerül a későbbiekben.

8.1.11 A $_*GET* tömb

A $_GET változót arra használjuk, hogy értékeket gyűjtsünk az űrlapról, ha az űrlap *method* tulajdonsága *get*.

A *$_GET* egy tömb, amelyben a HTTP GET metódussal elküldött változók nevei és értékei szerepelnek.

A *$_GET* változót arra használjuk, hogy értékeket gyűjtsünk az űrlapról a GET metódussal. A GET-tel mindenki számára látható módon tudunk adatokat küldeni az űrlapból (megjelenik a böngésző címsorában, egész pontosan az URL-ben).

Az elküldhető információ mennyisége korlátozva van (maximum 100 karakter). Példa:

```
<form action="welcome.php" method="get">
  Name: <input type="text" name="name" />
  Age: <input type="text" name="age" />
  <input type="submit" />
</form>
```

Amikor a felhasználó a *submit* gombra kattint, az URL-ben megjelennek az adatok:

```
http://www.w3schools.com/welcome.php?name=Peter&age=37
```

A *welcome.php* fájl arra használja a *$_GET* változót, hogy elkapja az űrlap adatait (megjegyezzük, hogy az űrlap mezők nevei automatikusan a *$_GET* tömb azonosítói lesznek):

```
Welcome <?php echo $_GET["name"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

Miért használjuk a *$_GET*-et?

Megjegyzés: Amikor használjuk a *$_GET* változót, akkor az összes változó neve és értéke megjelenik az URL-ben. Tehát ennek a metódusnak a használata nem ajánlott jelszó és egyéb bizalmas információk küldésekor. Viszont pont emiatt lehetséges könyvjelző elhelyezése. Ez sok esetben hasznos, máskor kifejezetten hátrányos lehet.

Megjegyzés: A HTTP GET metódus nem alkalmas nagy méretű változók értékeinek küldésére; a változó hossza nem haladhatja meg a 100 karaktert.

$_GET űrlap nélkül

Elsőre talán meglepőnek tűnik, de a *$_GET* tömb elemei nem csak űrlap kitöltésével jöhetnek létre. Semmi akadálya annak, hogy az URL eleve tartalmazzon kulcs-érték párokat, például a link eleve ilyen volt:

```
<a href="index.php?id=23">Masik oldal</a>
```

Ebben az esetben űrlap nélkül is lesz tartalma a *$_GET* tömbnek.

Megjegyzés: Hasonló okokból szokás űrlapba rejtett (*hidden*) mezőt elhelyezni. A kitöltött adatokkal együtt ezek az adatok is el fognak jutni a *$_GET* tömbbe.

A *$_REQUEST* változó

A PHP *$_REQUEST* tartalmazza a *$_GET*, *$_POST* és *$_COOKIE* változók elemeit.

Arra használjuk, hogy megkapja a GET és POST metódussal történt adatküldés eredményét.

Példa:

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br />
You are <?php echo $_REQUEST["age"]; ?> years old!
```

Megjegyzés: Biztonsági okokból a *$_REQUEST* használata általában kerülendő. Meg kell ugyanis győződnünk arról, hogy a fontos adatok tényleg onnan érkeztek-e, ahonnan mi várjuk, és nem valahonnan máshonnan.

8.1.12 A *$_POST* tömb

A *$_POST* változó értékeket gyűjt össze azokból az űrlapokból, melyek POST metódust használnak.

A *$_POST* változó neveket és értékeket tartalmazó tömb, melyek a HTTP POST metódussal lettek továbbítva.

Az az információ, melyet egy űrlapról küldenek POST metódussal, láthatatlan a többi felhasználó részére, és nincs korlátozva az információ mennyiségét illetően. Példa:

```
<form action="welcome.php" method="post">
  Enter your name: <input type="text" name="name" />
  Enter your age: <input type="text" name="age" />
  <input type="submit" />
</form>
```

Amikor a felhasználó a *submit* gombra kattint, az URL nem fog semmilyen űrlap adatot tartalmazni, és a valahogy így fog kinézni:

```
http://www.w3schools.com/welcome.php
```

Így a *welcome.php* fájl most már használhatja a *$_POST* változót, hogy az űrlap adatokat elérhesse (észre kell venni, hogy az űrlap mezők neve automatikusan az azonosító kulcsok a *$_POST* tömbből):

```
Welcome
<?php
  echo $_POST["name"];
?>.<br />
  You are
<?php
 echo $_POST["age"];
?> years old!
```

Miért használjunk *$_POST*–ot?

- A HTTP POST-al küldött változók nem láthatók az URL-ben
- A változóknak nincs hosszúsági korlátjuk

Egyébként, mivel az URL-ben nem láthatók a változók, nem lehetséges az oldalakat könyvjelzővel ellátni. (Ez általában nem is lenne praktikus.)

A *$_REQUEST* változó

A *$_REQUEST* változó tartalmazza a *$_GET*, *$_POST*, és *$_COOKIE* tartalmát. Ez a változó használható a GET és a POST metódusokkal elküldött űrlap adatok elérésére. Példa:

```
Welcome
<?php
echo $_REQUEST["name"];
?>.<br />
You are
<?php
echo $_REQUEST["age"];
?> years old!
```

Megjegyzés: Biztonsági okokból a *$_REQUEST* használata általában kerülendő. Meg kell ugyanis győződnünk arról, hogy a fontos adatok tényleg onnan érkeztek-e, ahonnan mi várjuk, és nem valahonnan máshonnan.

8.2. Haladó témák

A nyelvi és nyelvtani alapok áttekintése után lépjünk egy kicsit tovább.

8.2.1 Dátumok kezelése

A PHP a dátumok kezelésére az ún. Unix időbélyeg (timestamp) megközelítést alkalmazza. (A Unix időbélyeg az 1970. január 1-óta eltelt másodpercet jelenti.)

A *date* függvény

A PHP *date* függvény tetszőlegesen formáz egy adott dátumot/időt. Szintaxis:

```
date(format,timestamp)
```

A *format* paraméter kötelező, a kinézetet fogja meghatározni. A *timestamp* elhagyása esetén az aktuális (szerver) időt fogja formázni.

A dátum formázására a legalapvetőbb lehetőségeket nézzük meg:

d: a hónap napja 2 jeggyel

m: a hónap száma 2 jeggyel

Y: az év 4 jeggyel

Nézzünk egy egyszerű példát:

```
<?php
  echo date("Y/m/d");
  echo "<br />";
  echo date("Y.m.d");
  echo "<br />";
  echo date("Y-m-d");
?>
```

Az eredmény:

```
2006/07/11
2006.07.11
2006-07-11
```

A *date* (elhagyható) második paramétere egy időbélyeg lehet.

Az *mktime* függvény

Az *mktime* függvény egy időbélyeget képes előállítani. Szintaxis:

```
mktime(hour,minute,second,month,day,year,is_dst)
```

A következő példa a honlapi dátumot írja ki:

```
<?php
  $tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
  echo "Tomorrow is ".date("Y/m/d/", $tomorrow);
?>
```

A *checkdate* függvény

A *checkdate()* függvény visszatérési értéke igaz, ha az argumentumban megadott dátum érvényes, egyébként hamis.

Egy dátum érvényes ha:

- a hónap 1 és 12 közötti
- A *day* paraméter a *month* paramétereknek megfelelő értéket vesz fel.
- az év 1 és 32767 között van

Szintaxis:

```
checkdate(month,day,year)
```

Példa:

```
<?php
  var_dump(checkdate(12,31,2000));
  var_dump(checkdate(2,29,2003));
  var_dump(checkdate(2,29,2004));
?>
```

Az eredmény:

```
true
false
true
```

8.2.2 Az *include* és társai

Az *include* függvény használata esetén a PHP lényegében bemásolja a sor helyére a paraméterként megadott fájlt, és – ha tartalmaz PHP kódot – lefuttatja azt.

Az *include* függvény

A következő példa a *header.php* tartalmát szúrja be az *include* sorába.

```
<html>
<body>
<?php include("header.php"); ?>
<h1>Welcome to my home page</h1>
<p>Some text</p>
</body>
</html>
```

Tegyük fel, hogy az oldalainkra ugyanazt a menüt akarjuk beszúrni:

```
<html>
<body>
<a href="default.php">Home</a> |
<a href="about.php">About Us</a> |
<a href="contact.php">Contact Us</a>

...
```

Ha mindhárom oldalra (*default.php*, *about.php*, *contact.php*) ugyanezt a menüt akarjuk alkalmazni, akkor hagyományos megoldással mindhárom oldalra be kell illeszteni. Ez azonban több okból sem szerencsés. Ehelyett tegyük a fenti kódot egy külön *menu.php* fájlba, majd minden további oldalról csak illesszük be. Például a *default.php*:

```
<?php include("menu.php"); ?>
<h1>Welcome to my home page</h1>
<p>Some text</p>
</body>
</html>
```

Természetesen ha megnézzük a böngészőben az állományt, akkor ott a beillesztett tartalmat fogjuk látni.

A *require* függvény

Ez a függvény nagyon hasonlít az előzőre: mindössze a hibakezelésben van köztük eltérés. Az *include warning*-ot ad (és a kód tovább fut) hiba esetén, míg a *require error*-t ad, és a kód futását megszakítja.

Tegyük fel, hogy a *wrongFile.php* hibás kódot tartalmaz.

```
<html>
<body>

<?php
  include("wrongFile.php");
  echo "Hello World!";
?>

</body>
</html>
```

Az *include* miatt a futás nem szakad meg a hibánál, a hibaüzenetek után az *echo* is lefut:

```
Warning: include(wrongFile.php) [function.include]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5

Warning: include() [function.include]:
Failed opening 'wrongFile.php' for inclusion
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5

Hello World!
```

Ha az *include*-ot kicseréljük *require*-re, akkor a futás megszakad, és a következő eredményt kapjuk:

```
Warning: require(wrongFile.php) [function.require]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5

Fatal error: require() [function.require]:
Failed opening required 'wrongFile.php'
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
```

Egyedi beillesztés

Van, amikor szándékosan illesztünk be egy állományt többször (pl. egy ciklus belsejében), de van, amikor a többszörös beillesztés káros lenne (pl. függvény vagy osztálydefiníció esetén), vagyis el kell kerülni.

A többszörös beillesztés elkerülése végett alkalmazhatjuk az *include_once* és *require_once* függvényeket, amelyek hatására az esetleges többszörös beillesztés nem fog megtörténni.

8.2.3 Fájlkezelés

Az *fopen()* függvényt fájlok megnyitására használjuk a PHP-ben. A függvény első paramétere tartalmazza a megnyitandó fájl nevét, a második azt határozza meg, hogy milyen módon nyissuk meg a fájlt:

```
<html>
<body>
<?php
  $file=fopen("welcome.txt","r");
?>
</body>
</html>
```

A fájl az alábbi módokban nyitható meg:

Mód	Leírás
r	Csak olvasás. A fájl elején kezdődik.
r+	Írás/Olvasás. A fájl elején kezdődik.
w	Csak írás. Megnyitja és törli a fájl tartalmát; vagy egy új fájlt készít ha a fájl nem létezik.
w+	Olvasás/Irás. Megnyitja és törli a fájl tartalmát; vagy egy új fájlt készít ha a fájl nem létezik.
a	Hozzáfűzés. Megnyitja a fájlt és a végére kezd írni; vagy egy új fájlt készít ha a fájl nem létezik.
a+	Olvasás/Hozzáfűzés. Megőrzi a fájl tatalmát és csak a végére ír.
x	Csak írás. Készít egy új fájlt. Hibát jelez (*false*) ha már létezik.
x+	Olvasás/Írás. Készít egy új fájlt. Hibát jelez (*false*) ha már létezik.

Megjegyzés: Ha az *fopen()* nem tudja megnyitni a fájlt, *false*-t ad vissza.

Az alábbi példa egy üzenetet ad vissza, ha az *fopen()* nem tudja megnyitni a megadott fájlt:

```
<html>
<body>
<?php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
?>
</html>
</body>
```

Megjegyzés: A kód az *or* rövidzár kiértékelése miatt csak az *fopen false* visszaadott értéke esetén futtatja az *exit* függvényt.

A fájl bezárása

Az *fclose()* függvényt fájlok bezárására használjuk. Illik – más erőforrásokhoz hasonlóan – a fájlokat a lehető legkorábban „elengedni”.

```
<?php
  $file = fopen("test.txt","r");
  //...
  fclose($file);
?>
```

Fájlvég ellenőrzés

Az *feof()* függvény ellenőrzi a „fájlvég” (EOF) elérését.

Az *feof()* függvény hasznos ismeretlen hosszúságú adatokon való iterációknál.

```
if (feof($file)) echo "End of file";
```

Fájl soronkénti beolvasása

Az *fgets()* függvényt egy fájl egy sorának beolvasására használjuk.

Megjegyzés: A függvény hívása után a „fájl mutató” a következő sorra mutat.

Az alábbi példa sorról sorra beolvas egy fájlt, míg el nem éri annak végét:

```
<?php
  $file = fopen("welcome.txt", "r") or
      exit("Unable to open file!");
  while(!feof($file)) {
    echo fgets($file). "<br />";
  }
  fclose($file);
?>
```

Fájl karakterszintű beolvasása

Az *fgetc()* függvény egy fájl egy karakterének beolvasására szolgál.

Megjegyzés: A függvény hívása után a fájlmutató a következő karakterre mutat.

Az alábbi példa karakterenként olvas be egy fájlt, míg el nem éri annak végét.

```
<?php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
while (!feof($file)) {
  echo fgetc($file);
}
fclose($file);
?>
```

Megjegyzés: PHP-ben nem jellemző a bináris fájlkezelés, legtöbbször szöveges állományokkal dolgozunk.

8.2.4 Fájl feltöltése

PHP-vel lehetséges fájlok szerverre történő feltöltése is.

Fájlfeltöltő űrlap készítése

Egy fájl űrlapból való feltöltése igen hasznos lehet. Nézzük meg az alábbi fájl feltöltésre használt HTML űrlapot:

```
<html>
<body>
  <form action="upload_file.php" method="post"
    enctype="multipart/form-data">
    <label for="file">Filename:</label>
    <input type="file" name="file" id="file" />
    <br />
    <input type="submit" name="submit" value="Submit" />
  </form>
</html>
</body>
```

Jegyezzük meg a következőket a HTML űrlappal kapcsolatban:

- A *form enctype* attribútuma határozza meg, hogy melyik tartalom típust használjuk, amikor az űrlapot elfogadjuk. A *multipart/form-data* akkor használatos, ha az űrlap bináris adatot vár, mint pl. egy fájl tartalma feltöltéskor.
- Az *input* elem *file* típusa határozza meg, hogy a függvény bemenete fájlként legyen feldolgozva. Például böngészőben való megjelenítéskor lesz egy *Tallózás* gomb a bemenet mező mellett.

Megjegyzés: Feltöltés engedélyezése felhasználók számára nagy kockázattal jár. Kizárólag megbízható felhasználók számára tegyük lehetővé fájlok feltöltést, és feltöltés után is ellenőrizzük a fájlt.

Feltöltő szkript készítése

Az *upload_file.php* fájl tartalmazza a fájl feltöltésére szolgáló programot:

```
<?php
  if ($_FILES["file"]["error"] > 0) {
    echo "Error: " . $_FILES["file"]["error"] . "<br />";
  } else {
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024)
         . " Kb<br />";
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
  }
?>
```

A *$_FILES* globális PHP tömb használatával fájlokat tölthetünk fel egy kliens gépről a távoli szerverre.

Az első paraméter az űrlap bemenetének a neve, a második lehet *name*, *type*, *size*, *tmp_name* vagy *error*, a következőképpen:

$_FILES["file"]["name"]	a feltöltött fájl neve
$_FILES["file"]["type"]	a feltöltött fájl típusa
$_FILES["file"]["size"]	a feltöltött fájl mérete byte-okban
$_FILES["file"]["tmp_name"]	a szerveren tárolt fájl másolatának átmeneti neve
$_FILES["file"]["error"]	a fájl feltöltése során kapott hibakód

Ez a fájlfeltöltés egy nagyon egyszerű módja. Biztonsági okokból korlátozásokat kell bevezetni a felhasználóknál a feltöltésekre vonatkozóan.

A feltöltés korlátozása

Ebben a szkriptben a fájlfeltöltést korlátokhoz kötjük. A felhasználó kizárólag .gif vagy .jpeg fájlokat tölthet fel, és a fájlméretnek 20 kb alatt kell maradnia:

```
<?php
  if ((($_FILES["file"]["type"] == "image/gif")
      || ($_FILES["file"]["type"] == "image/pjpeg"))
        && ($_FILES["file"]["size"] < 20000)) {
    if ($_FILES["file"]["error"] > 0) {
      echo "Error: " . $_FILES["file"]["error"] . "<br />";
    } else {
      echo "Upload: " . $_FILES["file"]["name"] . "<br />";
      echo "Type: " . $_FILES["file"]["type"] . "<br />";
      echo "Size: " . ($_FILES["file"]["size"] / 1024)
           . " Kb<br />";
      echo "Stored in: " . $_FILES["file"]["tmp_name"];
    }
  } else {
    echo "Invalid file";
  }
?>
```

A feltöltött fájl elmentése

A fenti példák elkészítik a szerveren a feltöltött fájlok egy-egy átmeneti másolatát a PHP *temp* mappában.

Megjegyzés: a *temp* mappa egy logikai kifejezés, ténylegesen a php.ini beállításától függ.

Az átmeneti fájlok törlésre kerülnek, amint a szkript futása véget ér. Ahhoz, hogy a fájlt eltároljuk, át kell másolnunk máshová:

```
<?php
  if ((($_FILES["file"]["type"] == "image/gif")
        || ($_FILES["file"]["type"] == "image/pjpeg")
      && ($_FILES["file"]["size"] < 20000)) {
    if ($_FILES["file"]["error"] > 0) {
      echo "Return Code: " . $_FILES["file"]["error"]
           . "<br />";
    } else {
      echo "Upload: " . $_FILES["file"]["name"] . "<br />";
      echo "Type: " . $_FILES["file"]["type"] . "<br />";
      echo "Size: " . ($_FILES["file"]["size"] / 1024)
           . " Kb<br />";
      echo "Temp file: " . $_FILES["file"]["tmp_name"]
           . "<br />";

      if (file_exists("upload/" . $_FILES["file"]["name"])) {
        echo $_FILES["file"]["name"] . " already exists. ";
      } else {
        move_uploaded_file($_FILES["file"]["tmp_name"],
        "upload/" . $_FILES["file"]["name"]);
        echo "Stored in: " . "upload/"
             . $_FILES["file"]["name"];
```

```
      }
    }
  } else {
    echo "Invalid file";
  }
?>
```

A fenti szkript leellenőrzi, létezik-e már a fájl, ha pedig nem, bemásolja azt a megadott mappába.

Megjegyzés: Ez a példa lementi a fájlt az *upload* nevű mappába.

8.2.5 Sütik kezelése

A sütiket (cookie) általában a felhasználó azonosítására használjuk.

Mi is az a süti?

A süti a szerver által a kliens gépen tárolt adat, ami a felhasználó azonosítására szolgál. A böngésző a sütit minden kérés alkalmával elküldi a kiszolgálónak. PHP segítségével készíthetünk sütit, valamint a meglévőket kiolvashatjuk.

Hogyan készítsünk sütit?

Süti készítésére a *setcookie()* függvényt használhatjuk.

Fontos: a *setcookie()* függvénynek még a<html> tag előtt meg kell jelennie.

Megjegyzés: A HTTP protokoll szerint a szerver először ún. fejlécet küld, ennek a fejlécnek lesz része a sütink is. A HTML oldal a fejléc után kerül küldése, egy elválasztó üres sor után. Ha tehát akár egyetlen betűt is elküldünk az oldalból, akkor már nem küldhetünk fejlécet.

Szintaxis:

```
setcookie(name, value, expire, path, domain);
```

Az alábbi példában létrehozunk egy *user* nevű sütit és az *Alex Porter* értéket rendeljük hozzá, továbbá az érvényességi időt egy órában határozzuk meg.

```
<?php
  setcookie("user", "Alex Porter", time()+3600);
?>
<html>
<body>
</body>
</html>
```

Megjegyzés: a süti értéke automatikusan kódolásra kerül küldéskor, és automatikusan dekódolódik kiértékeléskor. Kódolás nélküli küldéshez a *setrawcookie()* függvény használható.

A süti kiolvasása

A süti értéke a *$_COOKIE* tömbben kerül tárolásra. Az alábbi példában a *user* nevű süti értékét, majd az egész tömb tartalmát írjuk ki a képernyőre.

```
<?php
  echo $_COOKIE["user"];
  print_r($_COOKIE);
?>
```

A következő kódrészletben az *isset()* függvény segítségével ellenőrizzük, hogy érkezett e süti.

```
<html>
<body>
<?php
  if (isset($_COOKIE["user"]))
    echo "Welcome " . $_COOKIE["user"] . "!<br />";
  else
    echo "Welcome guest!<br />";
?>
</body>
</html>
```

Süti törlése

A süti törlése gyakorlatilag azt jelenti, hogy a sütit lejártnak állítjuk be.

Példa törlésre (az érvényesség lejártának beállítása egy órával ezelőttre):

```
<?php
  setcookie("user", "", time()-3600);
?>
```

Ha a böngésző nem támogatja a sütiket

Amennyiben a böngésző nem támogatja a sütiket, űrlapokkal kell továbbítani az adatokat.

Az alábbi példában a felhasználói bemenet a *welcome.php* oldalnak kerül elküldésre a *submit* gomb megnyomására.

```
<html>
<body>
  <form action="welcome.php" method="post">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
  </form>
</body>
</html>
```

Az elküldött adatok kiolvasása az alábbi módon történik (*welcome.php*):

```
<html>
<body>
  Welcome <?php echo $_POST["name"]; ?>.<br />
  You are <?php echo $_POST["age"]; ?> years old.
</body>
</html>
```

8.2.6 Munkamenet-kezelés

A munkamenet-kezelést (session) arra használjuk, hogy információt tároljon a felhasználó beállításairól, vagy annak megváltozásáról.

Amikor egy asztali (desktop) alkalmazással dolgozunk, megnyitjuk, változtatunk valamit, majd bezárjuk. A számítógép tudja, hogy ki végzi a műveletet. Tudja, amikor elindítjuk az alkalmazást, és amikor bezárjuk. De az interneten ez nem így van. A webszerver nem tudja, hogy kik vagyunk és mit csinálunk, mert a HTTP protokoll nem tartja meg a kérések közt az állapotát.

Megjegyzés: Úgy is mondjuk, hogy a HTTP protokoll állapotmentes.

A PHP session (munkamenet) kezelés megoldja ezt a problémát. Engedélyezi a felhasználókra vonatkozó információk tárolását a szerveren (felhasználó név, vásárlási tételek, stb.). A session információk ideiglenesek, és törlődnek, miután a felhasználó elhagyta az oldalt. Ha állandó tárolásra van szükség, akkor az adatokat adatbázisba kell elmenteni.

A session-ök úgy működnek, hogy létrehozunk egy egyedi azonosítót (UID) minden látogatónak, és változókban tároljuk. Az UID vagy cookie-ban, vagy az URL-ben kerül továbbításra.

Munkamenet indítása

Mielőtt információkat tárolnánk, el kell indítani a munkamenetet.

Megjegyzés: A session_start() függvénynek a <html> tag előtt kell szerepelnie:

```
<?php session_start(); ?>
<html>
<body>

</body>
</html>
```

Az előbbi kód regisztrálja a session-t a szerveren, engedélyezi a felhasználói információk mentését, és az UID-t továbbítja a felhasználó számára.

A session változó tárolása

A session változók tárolása és visszaolvasása a PHP $_SESSION változóval lehetséges:

```
<?php
session_start();
$_SESSION['views']=1;
?>
<html>
<body>
<?php
  echo "Pageviews=". $_SESSION['views'];
?>
</body>
</html>
```

Kimenet:

```
Pageviews=1
```

Az előző példában készítettünk egy egyszerű oldal látogatottság számlálót. Az *isset()* függvény ellenőrzi, hogy a *views* munkamenet-változó kapott-e már értéket. Ha igen, akkor növeljük eggyel. Ha a *views* nem létezik, akkor létrehozzuk, és beállítjuk 1-re:

```
<?php
session_start();

if(isset($_SESSION['views']))
  $_SESSION['views']=$_SESSION['views']+1;
else
  $_SESSION['views']=1;

echo "Views=". $_SESSION['views'];
?>
```

A session törlése

A session adatok törlése az *unset()* vagy a *session_destroy()* függvényekkel történik.

Az *unset()* függvényt a session változó törlésére használjuk:

```
<?php
  unset($_SESSION['views']);
?>
```

A session teljes törlése a *session_destroy()* függvénnyel lehetséges:

```
<?php
  session_destroy();
?>
```

Megjegyzés: a *session_destroy()* függvény újraindítja a session-t, és az összes tárolt session adatot elveszítjük.

Megjegyzés: korábban a *session_register* és a *session_unregister* függvények voltak használatosak a munkamenet-változók kezelésére. Ez azonban elavultnak tekinthető, használjuk mindig közvetlenül a *$_SESSION* tömböt.

8.2.7 Levélküldés

A PHP lehetővé teszi, hogy közvetlenül szkriptből küldjünk e-maileket.

A PHP *mail()* függvényt arra használják, hogy segítségével szkripten belül e-maileket küldjenek. Szintaxis:

```
mail(to,subject,message,headers,parameters)
```

paraméter	**leírás**
to	Kötelezően megadandó. Az e-mail címzettjét/címzetteit határozza meg.
subject	Kötelezően megadandó. Az email témája. Megjegyzés: Ez a paraméter nem tartalmazhat újsor (newline) karaktereket.
message	Kötelezően megadandó. Az elküldendő üzenetet határozza meg. Minden új sort LF (\n) karakterrel kell elválasztani. Egy sor terjedelme nem haladhatja meg a 70 karaktert.
headers	Opcionális. További fejléceket adhatunk meg, mint például From, Cc és Bcc. A további fejléceket CRLF (\r\n) karakterrel kell elválasztani.
parameters	Opcionális. További paramétereket adhatunk meg a sendmail programnak

Megjegyzés: A *mail* függvény használatához a PHP-nek szüksége van egy feltelepített és működő levelezőrendszerre. A használandó programot a *php.ini* fájl konfigurációs beállításai határozzák meg. További információt a PHP levél referencia részben találhatunk.

PHP levélküldés egyszerűen

A legegyszerűbb levélküldési mód PHP-ben a szöveges levél küldése.

Az alábbi példában először deklaráljuk a változókat (*$to*, *$subject*, *$message*, *$from*, *$headers*), majd a *mail* függvényben felhasználjuk ezeket a változókat e-mailküldésre:

```
<?php
  $to = "someone@example.com";
  $subject = "Test mail";
  $message = "Hello! This is a simple email message.";
  $from = "someonelse@example.com";
  $headers = "From: $from";
  mail($to,$subject,$message,$headers);
  echo "Mail Sent.";
?>
```

PHP levél űrlapból

A PHP segítségével létre tudunk hozni egy visszajelző űrlapot weboldalunkon. A következő példában egy szöveges üzenetet küldünk el a megadott email címre.

```
<html>
<body>
<?php
if (isset($_REQUEST['email'])) {
  $email = $_REQUEST['email'] ;
  $subject = $_REQUEST['subject'] ;
  $message = $_REQUEST['message'] ;
  mail( "someone@example.com", "Subject: $subject",
  $message, "From: $email" );
  echo "Thank you for using our mail form";
} else {
  echo "<form method='post' action='mailform.php'>
  Email: <input name='email' type='text' /><br />
  Subject: <input name='subject' type='text' /><br />
  Message:<br />
  <textarea name='message' rows='15' cols='40'>
  </textarea><br />
  <input type='submit' />
</form>";
}
?>
</body>
</html>
```

A példa működése a következő:

Először megvizsgáljuk, hogy az *email* beviteli mező ki van-e töltve. Ha nincs kitöltve (például ha ez az első látogatás az oldalon), akkor megjelenítjük a HTML űrlapot.

Ha ki van töltve (miután az űrlap kitöltésre került), elküldjük az űrlapból az emailt. Ha az űrlap kitöltése után nyomják meg az elküld (submit) gombot, a lap újratöltődik, megnézi, hogy az *email* beviteli mező ki van-e töltve, majd elküldi az e-mailt.

Biztonságos levélküldés

Az előző fejezetben bemutatott *mail* függvénynek van egy gyenge pontja: lehetőséget ad befecskendezéses támadás (injection) megvalósítására.

A probléma a fent bemutatott kóddal, hogy illetéktelen felhasználók be tudnak szúrni adatokat a levél fejlécébe a beviteli űrlapon keresztül.

Mi történik, ha a felhasználó a következő szöveget írja be az email beviteli mezőbe az űrlapon?

```
someone@example.com%0ACc:person2@example.com
%0ABcc:person3@example.com,person3@example.com,
anotherperson4@example.com,person5@example.com
%0ABTo:person6@example.com
```

A *mail* függvény a szokott módon a fenti szöveget helyezi el a levél fejlécében, és így a fejléc többlet Cc:, Bcc:, és To: mezőket tartalmaz. Amikor a felhasználó az elküld (Submit) gombra kattint, a fenti összes címre elküldi a levelet.

Befecskendezéses támadás meggátolása

A legjobb módja az e-mail-befecskendezéses támadás megállításának a bevitt adatok ellenőrzése.

Az alábbi kód majdnem ugyanaz, mint a korábbi, de most ellenőrizzük az *email* mezőt:

```
<html>
<body>
<?php
  function spamcheck($field) {
    return eregi("to:",$field) || eregi("cc:",$field);
  }

  if (isset($_REQUEST['email'])) {
    $mailcheck = spamcheck($_REQUEST['email']);
    if ($mailcheck==TRUE) {
      echo "Invalid input";

    } else {
      $email = $_REQUEST['email'] ;
      $subject = $_REQUEST['subject'] ;
      $message = $_REQUEST['message'] ;
      mail("someone@example.com", "Subject: $subject",
      $message, "From: $email" );
      echo "Thank you for using our mail form";
    }

  } else {
    echo "<form method='post' action='mailform.php'>
    Email: <input name='email' type='text' /><br />
    Subject: <input name='subject' type='text' /><br />
    Message:<br />
    <textarea name='message' rows='15' cols='40'>
    </textarea><br />
    <input type='submit' />
    </form>";
  }
?>
</body>
</html>
```

8.2.8 PHP hibakezelés

Amikor webalkalmazásokat készítünk, a hibakezeléssel nagyon fontos törődni. Ha az alkalmazásunk híján van az ellenőrzésnek, az oldal nagyon szakszerűtlennek néz ki és biztonsági réseket tartalmazhat.

Ez a fejezet a PHP-ben leggyakrabban alkalmazott hibakezelő módszereket mutatja be:

- egyszerű *die* kifejezés

- szokásos hibák és hiba triggerek
- hibajelentések

Alapvető hibakezelés: a *die* függvény használata

Az első példa egy egyszerű szkriptet mutat be, mely egy szövegfájlt nyit meg:

```
<?php
  $file=fopen("welcome.txt","r");
?>
```

Ha a fájl nem létezik, akkor egy ehhez hasonló hibaüzenetnek kell megjelennie:

```
Warning: fopen(welcome.txt) [function.fopen]: failed to open
stream:
No such file or directory in C:\webfolder\test.php on line 2
```

Ennek elkerülésére tesztelnünk kell, hogy a fájl valóban létezik-e, mielőtt megpróbálnánk hozzáférni:

```
<?php
if(!file_exists("welcome.txt")) {
  die("File not found");
} else {
  $file=fopen("welcome.txt","r");
}
?>
```

Ha a fájl nem létezik, akkor egy ehhez hasonló hibaüzenetet fogunk kapni:

```
File not found
```

A fenti kódrészlet sokkal hatékonyabb az előzőnél, mert a szkript további futásának megakadályozására egy egyszerű hibakezelő eljárást alkalmaz. Másrészt megakadályozható vele, hogy olyan információkat szivárogtassunk ki, amelyek rosszindulatú támadásokhoz adhatnak támpontot

A szkript leállítása nem mindig a megfelelő út. Vessünk egy pillantást pár alternatív hibakezelő PHP funkcióra.

Alapértelmezett hibakezelő függvény készítése

Egy alapértelmezett hibakezelő függvény készítése meglehetősen egyszerű. Egyszerűen egy speciális függvényt kell létrehozni, ami meghívható, ha hiba merül fel futás közben.

Ennek a függvénynek tudnia kell kezelni legalább két paramétert (hiba szintje és hibaüzenet), de elfogadhat maximum öt paramétert. (Az opcionálisak: fájl, a sor száma és a hibakörnyezete.)

Szintaxis:

```
error_function(error_level,error_message,
               error_file,error_line,error_context)
```

Paraméter	**Leírás**
error_level	Kötelező. Megadja, hogy milyen súlyos hibákat tekintsünk errornak Egy számértéknek kell lennie.
error_message	Kötelező. Megadja a hibaüzenetet.

error_file	Opcionális. A fájlnevet írja le, ahol a hiba felmerült.
error_line	Opcionális. A sor számát határozza meg, ahol a hiba felmerült.
error_context	Opcionális. Egy tömböt tartalmaz, amiben benne van minden változó és annak értéke, melyek használatban voltak, amikor a hiba felmerült.

Hiba értesítési szintek

Ezek a hiba értesítési szintek azok a hibafajták, melyek orvoslására a különböző felhasználó által létrehozott hibakezelőket használni lehet:

Érték	Konstans	Leírás
2	*E_WARNING*	Nem végzetes futásidejű hibák. A szkript végrehajtása nem lesz megszakítva.
8	*E_NOTICE*	Futásidejű figyelmeztetés. A szkript talált valamit, ami lehetséges hibaforrás, de az értesítés akkor is előfordulhat, ha a szkript hibamentesen fut.
256	*E_USER_ERROR*	Végzetes felhasználó által generált hiba. Ez olyan, mint egy E_ERROR, melyet a programozó állít be a *trigger_error()* függvényt használva.
512	*E_USER_WARNING*	Nem végzetes felhasználó által generált figyelmeztetés. Ez olyan, mint egy E_WARNING, melyet a programozó állít be a *trigger_error()* függvényt használva.
1024	*E_USER_NOTICE*	Felhasználó által generált értesítés. Ez olyan, mint egy E_NOTICE, melyet a programozó állít be a *trigger_error()* függvényt használva.
4096	*E_RECOVERABLE_ERROR*	Elkapható végzetes hiba. Ez olyan, mint egy E_ERROR, de ez elkapható egy felhasználó által definiált hibakezelővel.
8191	*E_ALL*	Minden hiba és figyelmeztetés, kivéve a E_STRICT szintet.

Hozzunk létre egy függvényt a hibák kezelésére:

```
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br />";
  echo "Ending Script";
  die();
}
```

A fenti kódrészlet egy egyszerű hibakezelő függvény. Amikor megkapja a vezérlést, megkapja a hiba szintjét és egy hiba üzenetet. Ezután a kimenetre küldi a hibaszintet és üzenetet, és a szkript futása befejeződik.

Most tehát, miután írtunk egy hibakezelő függvényt, el kell döntenünk, milyen esetekben legyen meghívva.

A hibakezelő beállítása

Az alapértelmezett hibakezelő a PHP-ban egy beépített függvény. Most azonban a beépített megoldás helyett a fenti függvényt fogjuk használni.

Lehetséges úgy megváltoztatni a hibakezelőt, hogy csak bizonyos hibákra vonatkozzon, és a szkript különböző hibákat különbözőképpen tudjon kezelni. Mi most ebben a példában a szokásos hibakezelőt fogjuk használni az összes hiba esetén:

```
set_error_handler("customError");
```

Mivel azt akarjuk, hogy a hibakezelő függvényünk minden hibát kezeljen, a *set_error_handler()* csak egy paramétert kívánt, a második csak opcionális, és a hiba szintjét határozza meg.

Teszteljük a hibakezelőt, olyan változót megpróbálva kiíratni, ami nem létezik:

```
<?php
  function customError($errno, $errstr) {
    echo "<b>Error:</b> [$errno] $errstr";
  }

  set_error_handler("customError");

  echo($test);
?>
```

A kód kimenete valami ehhez hasonló lesz:

```
Error: [8] Undefined variable: test
```

Hiba előidézése

Egy szkriptben, melybe a felhasználók adatokat vihetnek be, hasznos lehet a hibák előidézése, amikor egy nem megfelelő bevitel adódik. A PHP-ban ez a trigger_error() által van kezelve.

Ebben a példában hiba történik, ha a *test* változó nagyobb, mint 1:

```
<?php
  $test=2;
  if ($test>1) {
    trigger_error("Value must be 1 or below");
  }
?>
```

A kód kimenete valami ehhez hasonló lesz:

```
Notice: Value must be 1 or below
in C:\webfolder\test.php on line 6
```

Egy hibát meg lehet hívni bárhol a szkripten belül, és egy második paraméter hozzáadásával meg lehet határozni, hogy milyen hiba szintet akarunk előidézni.

Lehetséges hiba típusok:

- E_USER_ERROR – Fatális felhasználó által generált futásidejű hiba. Hibák, melyeket nem lehet kijavítani. A szkript végrehajtása leáll.
- E_USER_WARNING - Nem fatális felhasználó által generált futásidejű figyelmeztetés. A szkript végrehajtása nem áll le.
- E_USER_NOTICE – Alapértelmezett. Felhasználó által generált futásidejű értesítés. A szkript talált valamit, ami lehetséges hibaforrás, de az értesítés akkor is előfordulhat, ha a szkript hibamentesen fut.

Ebben a példában egy E_USER_WARNING üzenet jelenik meg, ha a *test* változó nagyobb, mint 1. Egy ilyen esetben a szokásos hibakezelőnket használjuk, és befejezzük a szkript futtatását:

```
<?php
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br />";
  echo "Ending Script";
  die();
}

set_error_handler("customError",E_USER_WARNING);

$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

A kód kimenete valami ehhez hasonló lesz:

```
Error: [512] Value must be 1 or below
Ending Script
```

Most, hogy megtanultunk saját hibakezelőt készíteni, és hogy hogyan lehet meghívni őket, vessünk egy pillantást a hiba naplózásra.

Hiba naplózása

A php.ini *error_log* konfigurációjában beállított módon lehetőség van a hiba naplózásának finomhangolására.

Hibaüzenetek a saját email címre küldése egy jó módszer arra, hogy értesítve legyünk a különböző hibákról.

Egy hiba üzenet elküldése e-mailben

Az alábbi példában egy e-mailt fogunk elküldeni egy hibaüzenettel, és leállítjuk a szkriptet, ha egy bizonyos hiba megjelenik:

```
<?php
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br />";
  echo "Webmaster has been notified";
  error_log("Error: [$errno] $errstr",1,
  "someone@example.com","From: webmaster@example.com");
}

set_error_handler("customError",E_USER_WARNING);
```

```
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below",E_USER_WARNING);
}
?>
```

A kód kimenete valami ehhez hasonló lesz:

```
Error: [512] Value must be 1 or below
Webmaster has been notified
```

A levél, amit kapunk, valami ehhez hasonló lesz:

```
Error: [512] Value must be 1 or below
```

Ezt nem tanácsos használni minden hiba esetén. A rendszeres hibákat inkább a szerveren naplózzuk, a PHP alapértelmezett naplózási rendszerét használva.

8.2.9 Kivételkezelés

Akkor kell a kivételt használni, ha meg akarjuk változtatni az utasítások végrehajtásának normál menetét, hogyha egy meghatározott hiba bekövetkezik.

Mi is a kivétel

A PHP 5-tel egy új objektum orientált lehetőség is adott a hibák kezelésére.

A kivételkezelés használatakor az megváltoztatja az utasítások végrehajtásának normál menetét, ha egy meghatározott hiba (kivétel) feltétel bekövetkezik. Ezt a feltételt kivételnek nevezzük.

Általában a következő történik kivétel kiváltódásánál:

- A jelenlegi állapot lementődik.
- A kód végrehajtása átkapcsol egy előre meghatározott kivétel kezelési metodikára.
- A helyzettől függően a kezelő folytathatja a munkát a lementett állapottól, vagy pedig meghatározhatjuk a parancsvégrehajtást, illetve folytathatjuk a kód egy másik helyétől.

Megjegyzés: A kivételeket csak a hibák felmerülésekor kell lekezelni, és nem lehet arra használni, hogy a kód egy másik helyére ugorjunk velük.

A kivételek használata

Mikor kivétel keletkezik, az azt követő kód nem hajtódik végre, és a PHP megpróbálja megtalálni a hozzá tartozó *catch* blokkot.

Ha egy kivétel nincs elkapva, egy hiba fog keletkezni, egy "Uncaught Exception" („Lekezeletlen kivétel!”) üzenettel.

Dobjunk egy kivételt anélkül, hogy elkapnánk:

```
<?php
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}
checkNum(2);
?>
```

A fenti kód a következő hibát fogja generálni:

```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in
C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

try, *throw* és *catch*

Ha el szeretnénk kerülni a fenti hibát, létre kell hoznunk a megfelelő kódot, hogy kezelni tudja a kivételt.

A kivételkezelő kódnak tartalmaznia kell a következőket:

- *try* – Egy függvénynek, ami kivételt alkalmaz, benne kell lennie egy *try* blokkban. – (ha nem jelenik meg kivételként) ha nincs kivétel dobva, a kód normális módon fog folytatódni. Azonban, kivétel lekezelése esetén kivétel fog keletkezni
- *throw* – Így kapjuk el a kivételt. Minden egyes *throw*-hoz kell lennie legalább egy *catch*-nek
- *catch* - A *catch* blokk elkapja a hibát és egy objektumot készít, mely a kivételről tartalmaz információkat.

Próbáljunk megjeleníteni egy kivételt egy érvényes kóddal:

```
<?php
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}

try {
  checkNum(2);
  echo 'If you see this, the number is 1 or below';
}

catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

A fenti kód a következő hibát fogja okozni:

```
Message: Value must be 1 or below
```

A fenti kód dob egy kivételt, és el is kapja azt:

- A *checkNum()* függvény azt nézi, hogy a szám nagyobb-e 1-nél. Ha nagyobb, akkor a kivétel dobódik.

- A *checkNum()* függvény meg van hívva a "*try"* blokk-ban.
- A kivétel a *checkNum()* függvényen belül dobódik
- A *catch* blokk lekezeli a kivételt, és egy objektumot kreál, ami tartalmazza a kivétel információit.
- A hiba üzenet a kivételtől megjelenik a *$e->getMessage()* meghívása által a kivétel objektumból.

Azonban, az a szabály, hogy „minden *throw*-hoz kell lennie egy *catch*-nek" csak egyfajta megközelítése a kivétel lekezelésének. Az ezen keresztülcsúszott hibák lekezelésére egy felsőbb szintű kivételkezelést tudunk beállítani.

Kivétel osztály készítése

Kivétel osztályt készíteni elég egyszerű. Készíteni kell egy olyan osztályt, amely olyan függvényeket tartalmaz, amiket meg lehet hívni, amikor kivételek keletkeznek a PHP-ben. Természetesen ezt az osztályt az *Exception* osztály leszármazottjaként hozzuk létre.

Az általános kivétel osztály örökli a PHP *Exception* osztály tagjait, és további lehetőségeket tudunk hozzárendelni.

Készítsünk egy példa osztályt:

```
<?php
class CustomException extends Exception {
  public function errorMessage() {
    $errorMsg = 'Error on line '.$this->getLine().' in '
        .$this->getFile().': <b>'.$this->getMessage()
        .'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}

$email = "someone@example...com";
try {
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    throw new CustomException($email);
  }
}
catch (CustomException $e) {
  echo $e->errorMessage();
 }
?>
```

Az új leszármazott osztály az errorMessage() függvénnyel kibővíti az ősét. Mivel ez egy leszármazottja az ősosztálynak, olyan metódusokat is tudunk használni, mint: *getLine()* és *getFile()* vagy *getMessage()*.

A fenti kód egy kivételt dob, és a *CustomException* osztállyal kapja el:

- A C*ustomException()* osztály valójában az ős osztály kiterjesztése. Ez magába foglalja a metódusokat és annak tulajdonságait az ős osztálynak.
- Az *errorMessage()* függvény visszatér egy hibaüzenettel, ha az e-mail cím érvénytelen.
- Az *$email* változó - ami egy sztring - nem tekinthető érvényes e-mail címnek.
- A *try* blokk végrehajtódik, és kivétel dobódik, mivel az e-mail cím érvénytelen.

- A *catch* blokk elkapja a kivételt, és megjeleníti a hibaüzenetet.

Többféle kivétel használata

Lehetséges, hogy egy szkript többféle kivételt használ, hogy kezelni tudjon többféle hibalehetőséget.

Programozástechnikailag lehetne alkalmazni *if.else* blokkokat, *switch* szerkezetet is, de célszerűbb lehet többféle kivétel használata. Ez utóbbi különböző kivétel osztályok használatát fogja jelenteni, amelyek különböző hibaüzenetekkel fognak visszatérni:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    $errorMsg = 'Error on line '.$this->getLine().' in '
        .$this->getFile().': <b>'.$this->getMessage()
        .'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}

$email = "someone@example.com";

try {
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    throw new customException($email);
  }
  if(strpos($email, "example") !== FALSE) {
    throw new Exception("$email is an example e-mail");
  }
}
catch (customException $e) {
  echo $e->errorMessage();
}
catch(Exception $e) {
  echo $e->getMessage();
}
?>
```

A fenti kód a két lehetséges hibát teszteli és egy kivételt dob, hogyha nem felel meg valamelyik feltételnek:

- A C*ustomException* osztály az *Exception* osztály kiterjesztése.
- Az *errorMessage()* függvény egy hibaüzenetet ad vissza, ha az e-mail cím érvénytelen.
- A *$email* változó - ami egy sztring - egy érvényes e-mail cím, de tartalmazza az *example* sztring
- A *try* blokk végrehajtódik, és a kivétel az első feltételnél nem dobódik ki.
- A második feltétel dob egy kivételt, mivel az e-mail tartalmazza az *example* sztringet.
- A *catch* blokk elkapja a kivételt, és megjeleníti a helyes hibaüzenetet.

Ha a kódunk nem kapta volna el a C*ustomException* kivételeket, csak az általános kivételeket, akkor a hibaüzenetet az *Exception* osztálytól kaptuk volna meg.

Kivételek dobása

Néha mikor egy kivétel dobódik, talán az általánostól eltérően, másképp kezelnéd le. A *catch* blokkon belül lehetséges kivételt dobni másodszorra is.

A szkriptnek el kell rejtenie a rendszer hibáit a felhasználótól. Rendszerhibák fontosak a kódoló számára, de a felhasználónak nem. Hogy a felhasználó dolgát megkönnyítsük, újra kivétel tudunk dobni egy felhasználóbarát üzenetként:

```
<?php
class customException extends Exception {
  public function errorMessage() {
    $errorMsg = $this->getMessage()
        .' is not a valid E-Mail address.';
    return $errorMsg;
  }
}

$email = "someone@example.com";
try {
  try {
    if(strpos($email, "example") !== FALSE) {
      throw new Exception($email);
    }
  }
  catch(Exception $e) {
    throw new customException($email);
  }
}
catch (customException $e) {
  echo $e->errorMessage();
}
?>
```

A fenti kód azt teszteli, hogy az e-mail cím tartalmazza-e az *example* sztringet. Amennyiben igen, a kivétel újradobódik:

- A *customException()* osztály az ős osztályból kerül kiterjesztésre, így megkapja annak tulajdonságait és metódusait.
- Az *errorMessage()* függvény visszatér egy hibaüzenettel ha az e-mail cím érvényes
- A *$email* változó - ami egy sztring - , egy érvényes e-mail cím, de tartalmazza az *example* sztringet.
- A *try* blokk egy másik *try* blokkot tartalmaz, ami lehetségessé teszi, hogy a kivétel újra kidobódjon.
- A kivétel keletkezik, mivel az e-mail tartalmazza az *example* sztringet.
- A *catch* blokk elkapja a kivételt, és újra kidobja, mint *customException*.
- A *customException* el van kapva, és megjelenik egy hibaüzenetet.

Ha a kivétel nincs lekezelve a jelenlegi *try* blokkban, magasabb szinten fog egy *catch* blokkot keresni.

Felsőbb szintű kivételkezelés

A *set_error_handler()* függvény meghatároz egy felhasználó által definiált függvényt, ami lekezel minden eddig nem elkapott kivételt.

```
<?php
function myException($exception) {
  echo "<b>Exception:</b> " , $exception->getMessage();
}
set_exception_handler('myException');
  throw new Exception('Uncaught Exception occurred');
?>
```

A kód kimenetele a következő lesz:

```
Exception: Uncaught Exception occurred
```

A fenti kódban nem volt *catch* blokk használva. Helyette egy felsőbb szintű kivétel kezelés volt beállítva. Ezzel a függvénnyel el tudjuk kapni az eddig le nem kezelt kivételeket.

A kivételkezelés szabályai

- A kód egy *try* blokkban kell hogy legyen, a lehetséges kivételek lekezelését elősegítve.
- Minden egyes *try* blokknak vagy *throw* –nak rendelkeznie kell legalább egy *catch* blokkal.
- Összetett (többszörös) *catch* blokkokat arra használjuk, hogy le tudjuk kezelni a különböző osztályú kivételeket.
- Kivételek dobódhatnak (vagy újradobódhatnak) egy *catch* blokkban.

A legalapvetőbb szabály: **Ha kivételt dobunk, azt el is kell kapnunk.**

8.3. Adatbázis-kapcsolat kezelése PHP-ben

A PHP fejlesztők leggyakrabban MySQL adatbázisszervert alkalmaznak. Ezért ebben a fejezetben a MySQL használatának alapjait fogjuk áttekinteni.

8.3.1 MySQL alapok

A MySQL a legnépszerűbb[59] nyílt forrású adatbázis szerver.

Adatbázis táblák

Egy adatbázis többnyire egy vagy több táblát tartalmaz. Minden egyes táblának van neve (pl.: "Vásárlók" vagy "Rendelések"). Minden tábla tartalmaz rekordokat (sorokat) melyek adat(oka)t tartalmaznak.

Egy példa a táblákra, *Person* néven:

LastName	FirstName	Address	City

[59] A MySQL-t 2008 elején felvásárolta a SUN, amivel a MySQL jövőjét – a szerző véleménye szerint – még inkább erősítette.

Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

A fenti tábla három rekordot tartalmaz (egy minden személyre) és négy oszlopot, vagy más néven mezőt (*LastName*, *FirstName*, *Address*, és *City*).

Lekérdezések

Egy lekérdezés lehet információkérés (lekérdezés) vagy az adatokat megváltoztató parancs. Először az előbbivel foglalkozunk.

A MySQL-től le tudunk kérdezni az adatbázisból információkat. A lekérdezés eredménye egy rekordhalmaz lesz.

Nézzük meg a következő lekérdezést:

```
SELECT LastName FROM Person
```

A fenti lekérdezés kiválaszt minden adatot a *Person* tábla *LastName* oszlopában és visszaadja azt:

LastName
Hansen
Svendson
Pettersen

8.3.2 Kapcsolódás egy MySQL adatbázishoz

Mielőtt hozzáférnél és dolgoznál adatokkal az adatbázisban, a kapcsolatot kell létrehozni az adatbázishoz. PHP-ben ezt a mysql_connect() függvénnyel lehet megvalósítani. Szintaxis:

```
mysql_connect(servername,username,password);
```

Paraméter	**Leírás**
servername	Opcionális. Meghatározza azt a szervert, amihez kapcsolódni akarunk. Az alapbeállítás a következő: *localhost:3306*
username	Opcionális. Meghatározza a MySQL felhasználói nevet, amivel bejelentkezünk.
password	Opcionális. Meghatározza a MySQL felhasználó jelszavát . Az alapbe-

	állítás: "" (üres sztring).

Megjegyzés: Több paraméter is megadható, de a fentiek a legfontosabbak.

A következő példában elmentjük a kapcsolatot egy változóba *($con*) későbbi használatra. A *die* rész akkor fog végrehajtódni, ha a kapcsolódás nem sikerül:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}
// ...
?>
```

A kapcsolat bontása

A kapcsolat bontásra kerül, ahogy a szkript véget ér. Ha ezelőtt szeretnénk bontani a kapcsolatot, akkor a *mysql_close* függvényt használva tehetjük meg.

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// ...
mysql_close($con);
?>
```

8.3.3 Adatbázisok és táblák létrehozása

Egy adatbázis egy vagy több táblát tartalmazhat.

A CREATE DATABASE parancsot használhatjuk adatbázis létrehozására a MySQL-ben. Szintaxis:

```
CREATE DATABASE database_name
```

Hogy a PHP végrehajtsa a fenti parancsot, használnunk kell a *mysql_query()* függvényt. Ez elküld egy lekérdezést vagy parancsot a MySQL szervernek.

A következő példában létrehozunk egy adatbázist *my_db* néven:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

if (mysql_query("CREATE DATABASE my_db",$con)) {
  echo "Database created";
} else {
  echo "Error creating database: " . mysql_error();
}
mysql_close($con);
?>
```

Tábla létrehozása

A CREATE TABLE parancs használatával létre tudunk hozni egy táblát a MySQL adatbázisban. Szintaxis:

```
CREATE TABLE table_name (
  column_name1 data_type,
  column_name2 data_type,
  column_name3 data_type,
...
)
```

Hozzá kell adni a CREATE TABLE parancssort a *mysql_query()* függvényhez, hogy a parancs végrehajtódjon.

A következő példa megmutatja, hogyan lehet egy *person* nevű táblát létrehozni három mezővel. A mezők nevei: *FirstName*, *LastName* és *Age*:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

if (mysql_query("CREATE DATABASE my_db",$con)) {
  echo "Database created";
} else {
  echo "Error creating database: " . mysql_error();
}

mysql_select_db("my_db", $con);
$sql = "CREATE TABLE person (
  FirstName varchar(15),
  LastName varchar(15),
  Age int
)";
mysql_query($sql, $con);
mysql_close($con);
?>
```

Figyelem! Az adatbázist ki kell választani, mielőtt a táblát létrehozzuk. Az adatbázist a *mysql_select_db* függvénnyel tudjuk kiválasztani.

MySQL adattípusok

A különböző MySQL adattípusok a következő táblázatban láthatóak:

Numerikus adattípus

int(size) smallint(size) tinyint(size) mediumint(size) bigint(size)	Csak egész számokat tartalmaz. A maximális számjegyek számát meg lehet határozni a *size* paraméterben
decimal(size,d) double(size,d) float(size,d)	Tizedesvesszőt tartalmazó számok. A maximális számjegyek számát meg lehet határozni a *size* paraméterben A tizedesvesszőtől jobbra lévő számjegyek maximális számát a

	d paraméterben lehet megadni.

Szöveges adattípusok

char(size)	Fix hosszúságú sztringet tartalmaz (betűk, számok és speciális karakterek). A fix hosszúságot zárójelben lehet megadni.
varchar(size)	Változó hosszúságú sztringet tartalmaz (betűk, számok és speciális karakterek). A maximális hosszúságot zárójelben lehet megadni
tinytext	Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 255 karakter.
text blob	Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 65535 karakter
mediumtext mediumblob	Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 16777215 karakter.
longtext longblob	Változó sztringet tartalmaz egy megadott maximális hosszúsággal, ami 4294967295 karakter.

Dátum típusú adatok

date(yyyy-mm-dd) datetime(yyyy-mm-dd hh:mm:ss) timestamp(yyyymmddhhmmss) time(hh:mm:ss)	Dátumot és/vagy időt tartalmaz

Összetett típusok

enum(value1,value2,ect)	ENUM az ENUMERATED lista rövid formája. 1-65535 értéket tud raktározni listázva a ( ) zárójelben. Ha egy olyan értéket akarunk beszúrni, ami nincs a listában, úgy egy üres érték fog beszúródni.
set	A SET (halmaz) hasonló az ENUM-hoz. Azonban a SET 64 listázott tételet tartalmazhat több mint egy választási lehetőséggel

Elsődleges kulcsok és *autoincrement* mezők

Minden egyes táblának tartalmaznia kell egy elsődleges kulcs mezőt.

Az elsődleges kulcs arra szolgál, hogy egyedileg azonosítani lehessen a sorokat a táblában. Minden elsődleges kulcs értékének egyedinek kell lennie a táblán belül. Továbbá az

elsődleges kulcsú mező nem lehet *null*, mert az adatbázis (működése?) megkíván egy értéket, amely azonosítja a rekordot.

Az elsődleges kulcsú mező mindig indexelve van. Ez alól a szabály alól nincs kivétel!

A következő példa beállítja a *personID* mezőt elsődleges kulcsként. Az elsődleges kulcsú mező gyakran egy ID (azonosító) szám és gyakran van használva az *AUTO_INCREMENT* beállítással. Ez automatikusan növeli a mező értékét, ha egy új rekord adódik hozzá az eddigiekhez. Hogy biztosítva legyen, hogy az elsődleges kulcs mező nem nulla, kötelezően hozzá kell adni a *NOT NULL* beállítást a mezőhöz.

Példa:

```
$sql = "CREATE TABLE person (
  personID int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(personID),
  FirstName varchar(15),
  LastName varchar(15),
  Age int
)";
mysql_query($sql,$con);
```

8.3.4 Adatok bevitele adatbázisba

Az INSERT INTO paranccsal adatokat illeszthetünk be egy adatbázis táblába. Szintaxis:

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

Azt is meghatározhatjuk, hogy melyik oszlopba akarjuk az adatot beilleszteni:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Ezt a formát akkor érdemes használni, ha nem minden mezőnek akarunk értéket adni, vagy nem ugyanabban a sorrendben akarjuk az értékeket felsorolni.

Megjegyzés: az SQL parancsok nem érzékenyek a kis és nagybetűkre: az. INSERT INTO ugyanaz, mint insert into.

Ahhoz, hogy a PHP végrehajthassa a fenti parancsokat, a *mysql_query()* függvényt kell meghívni.

Az előző fejezetben egy *Person* nevű táblát hoztunk létre, három oszloppal: *Firstname*, *Lastname* és *Age*. Ugyanezt a táblát használjuk ebben a fejezetben is. A következő példa két új rekordot visz be a *Person* táblába:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);
mysql_query("INSERT INTO person (FirstName, LastName, Age)
  VALUES ('Peter', 'Griffin', '35')");
mysql_query("INSERT INTO person (FirstName, LastName, Age)
  VALUES ('Glenn', 'Quagmire', '33')");
mysql_close($con);
?>
```

Adatok beillesztése űrlapról egy adatbázisba

Most egy HTML űrlapot fogunk készíteni, melyet új adat bevitelére fogunk használni a *Person* táblába.

Íme a HTML űrlap:

```
<html>
<body>
  <form action="insert.php" method="post">
    Firstname: <input type="text" name="firstname" />
    Lastname: <input type="text" name="lastname" />
    Age: <input type="text" name="age" />
    <input type="submit" />
  </form>
</body>
</html>
```

Amikor egy felhasználó az Elküld gombra kattint a HTML űrlapon, az adatok elküldésre kerülnek az *insert.php* fájlnak. Az *insert.php* fájl egy adatbázishoz kapcsolódik, a *mysql_query()* függvény végrehajtja az INSERT INTO parancsot, és az új adat beszúrásra kerül az adatbázis táblába.

Az *insert.php* oldal kódja a következő:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);
$sql="INSERT INTO person (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";
if (!mysql_query($sql,$con)) {
  die('Error: ' . mysql_error());
}
echo "1 record added";
mysql_close($con)
?>
```

Megjegyzés: A példa nem foglalkozik a biztonsági kérdésekkel. Ellenőrzés nélkül soha nem szabad az adatokat felhasználnunk!

8.3.5 Lekérdezés

A SELECT parancs adatok kiválasztására szolgál egy adatbázisból. Alapvető (nem teljes) szintaxis:

```
SELECT column_name(s)
FROM table_name
WHERE conditions
```

Ahhoz, hogy a PHP végrehajthassa a fenti parancsokat, a *mysql_query()* függvényt kell meghívni.

Az alábbi példa kiválasztja az összes adatot, mely a *Person* táblában van tárolva (A * karakter kiválasztja az összes adatot a táblában):

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person");
while($row = mysql_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
}
mysql_close($con);
?>
```

A *mysql_query* függvény által visszaadott erőforrást a *$result* változóban tároljuk. Ezzel az erőforrással érhetőek el a lekérdezés eredményeként kapott rekordok.

A következő példában a *mysql_fetch_array* függvényt használjuk, hogy az első sort megkapjuk tömbként az adathalmazból. Minden későbbi meghívás a *mysql_fetch_array*-ra a következő sorral tér vissza az adathalmazból. Ahhoz, hogy kiírjuk az összes sort, a *$row* PHP változót használjuk (*$row['FirstName']* és *$row['LastName']*).

Megjegyzés: a *mysql_fetch_array* függvény *false* értéket ad, ha már nincs több rekord. Ezért is alkalmazható ilyen egyszerűen egy *while* ciklusban.

A *mysql_fetch_array* függvény a rekord mezőit asszociatív indexekkel és sorszámokkal is elérhetővé teszi. A példánkban a *$row['FirstName']* helyett *$row[1]*-et is írhattunk volna.

A fenti kód kimenete a következő lesz:

```
Peter Griffin

Glenn Quagmire
```

Eredmények ábrázolása HTML Táblában

A következő példa kiválasztja ugyanazokat az adatokat, mint a fenti példa, de az eredményeket HTML táblában fogja ábrázolni:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());}

mysql_select_db("my_db", $con);
$result = mysql_query("SELECT * FROM person");
echo "<table border='1'>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
  </tr>";
while($row = mysql_fetch_array($result)) {
  echo "<tr>";
  echo "  <td>" . $row['FirstName'] . "</td>";
  echo "  <td>" . $row['LastName'] . "</td>";
  echo "</tr>";
}
echo "</table>";
mysql_close($con);
?>
```

A fenti kód kimenete a következő lesz:

Firstname	**Lastname**
Glenn	Quagmire
Peter	Griffin

8.3.6 A *WHERE* záradék

Ha olyan adatokat szeretnénk kiválasztani, ami valamilyen feltételeknek megfelel, akkor a *SELECT*-hez hozzá kell adnunk egy *WHERE* záradékot. Szintaxis:

```
SELECT column FROM table
WHERE condition(s)
```

A következő operátorok használhatók a *WHERE*-rel:

Operátor	**Leírás**
=	Egyenlő
!=	Nem egyenlő
>	Nagyobb
<	Kisebb
>=	Nagyobb, vagy egyenlő
<=	Kisebb, vagy egyenlő
BETWEEN	Tartományba esés
LIKE	Mintával való egyezés

Megjegyzés: Az SQL kifejezések nem tesznek különbséget kis- és nagybetűk között. Tehát a *WHERE* ugyanaz, mint a *where*.

Az SQL parancs PHP-ben való futtatásához a mysql_query függvényt kell használnunk.

A következő példa kiválasztja az összes sort a *Person* táblából, ahol *FirstName='Peter'*:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person
  WHERE FirstName='Peter'");

while($row = mysql_fetch_array($result)) {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
}
?>
```

Eredmény:

```
Peter Griffin
```

8.3.7 Az *ORDER BY* kulcsszó

Az *ORDER BY* kulcsszó a lekérdezett adatok rendezésére szolgál. Szintaxis:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
```

Megjegyzés: Az SQL kifejezések nem tesznek különbséget a kis- és nagybetűk között. Tehát az *ORDER BY* ugyanaz, mint az *order by*.

A következő példa az eredményt rendezi az *Age* mező szerint:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person ORDER BY age");

while($row = mysql_fetch_array($result)) {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br />";
}

mysql_close($con);
?>
```

Eredmény:

```
Glenn Quagmire 33
Peter Griffin 35
```

Rendezés növekvő és csökkenő sorrendben

Ha az *ORDER BY* kulcsszót használjuk, a rendezés alapértelmezetten növekvő (pl. 1 után 9 és „a” után „p”).

A csökkenő sorrendbe való rendezéshez használjuk a *DESC* szót (pl. 9 után 1 és „p” után „a”):

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name DESC
```

Rendezés két vagy több oszlop alapján

A rendezés lehetséges egynél több oszlop alapján is. Ilyenkor a második (és az esetleges további) oszlopot csak akkor vesszük figyelembe, ha az első azonos:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name1, column_name2
```

8.3.8 Adatok módosítása

Az UPDATE utasítás az adatok módosítására szolgál az adatbázis táblában. Szintaxis:

```
UPDATE table_name
SET oszlop_nev = uj_ertek
WHERE oszlop_nev = valamilyen_ertek
```

Megjegyzés: Az SQL utasításoknál nem számít a kis és nagy betű. UPDATE ugyanaz, mint az update.

Az előbbi utasítássorozat lefordítására PHP-ben a *mysql_query* függvényt kell használnunk.

Példa

A következő példa feltölt pár adatot a *Person* táblába:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con) {
  die('Could not connect: ' . mysql_error());
}

mysql_select_db("my_db", $con);

mysql_query("UPDATE Person SET Age = '36'
WHERE FirstName = 'Peter' AND LastName = 'Griffin'");

mysql_close($con);
?>
```

A feltöltés után a *Person* tábla kinézete ilyen lesz:

FirstName	**LastName**	**Age**
Peter	Griffin	36
Glenn	Quagmire	33

Megjegyzés: a tábla többi rekordját nem akartuk megváltoztatni. Végzetes következménye lenne, ha a *where* záradékot lefelejtenénk.

8.3.9 Adatok törlése az adatbázisból

A DELETE FROM kifejezés használatos sorok törlésére az adatbázis egy adott táblájából. Szintaxis:

```
DELETE FROM table_name
WHERE column_name = some_value
```

Megjegyzés: az SQL nyelv nem különbözteti meg a kis- és nagybetűket, ebből következően a *DELETE FROM* és *delete from* kifejezések egyenértékűek.

A fenti példa PHP szkriptből való végrehajtására a *mysql_query* függvényt használhatjuk.

Az alábbi kódrészlet töröl minden olyan sort a táblából, ahol a *LastName* mező értéke *Griffin*.

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con){
  die('Could not connect: ' . mysql_error());
}
mysql_select_db("my_db", $con);
mysql_query("DELETE FROM Person WHERE LastName='Griffin'");
mysql_close($con);
?>
```

A végrehajtás után a tábla tartalma:

FirstName	LastName	Age
Glenn	Quagmire	33

8.3.10 ODBC kapcsolat létesítése

Megjegyzés: ODBC kapcsolat csak Windows operációs rendszer alatt használható.

Az ODBC egy olyan API, melynek segítségével sokféle adatforráshoz (pl MS Access adatbázis) csatlakozhatunk.

MS Access adatbázishoz az alábbi módon hozhatunk létre kapcsolatot.

1. A vezérlőpultban nyissuk meg a Felügyeleti eszközökön belül található ODBC adatforrásokat.
2. Válasszuk a Rendszer DSN fület.
3. Kattintsunk a Hozzáadásra.
4. Válasszuk ki a Microsoft Access driver-t.
5. A következő ablakban válasszuk a Kiválaszt gombot az adatbázis megadásához.
6. Adjuk meg az adatforrás nevét.
7. Az ok gombbal zárjuk le a műveletet.

A fenti beállításokat azon a számítógépen kell elvégezni, ahol a weboldal található. Amennyiben a saját számítógépünkön futtatjuk a webszervert, magunk is végrehajthatjuk azokat, azonban ha egy távoli gép a kiszolgáló, akkor a gép adminisztrátorát kell megkérni, hogy készítse el a DSN-t.

Kapcsolódás ODBC adatforráshoz

ODBC adatforráshoz az *odbc_connect()* függvény segítségével csatlakozhatunk. A függvény négy paramétert vár: adatforrás neve, felhasználó név, jelszó, és a kurzor típusa. Ez utóbbi paraméter elhagyható.

SQL utasítást az *odbc_exec()* függvénnyel hajthatunk végre.

Az alábbi kódrészlet egy *northwind* nevű DSN-hez kapcsolódik, felhasználónév és jelszó nélkül, valamint végrehajt egy lekérdezést.

```
$conn=odbc_connect('northwind','','');
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

Rekordok kiolvasása

Az *odbc_fetch_row()* függvény használatos rekordok kiolvasására egy eredménytáblából. Visszatérési értéke *true*, ha van megjeleníthető sor, egyébként *false*.

Két paramétert vár, az egyik az eredménytábla azonosítója, a második kért sor száma, de ez elhagyható.

```
odbc_fetch_row($rs)
```

Mezők kiolvasása

Egy rekord adott mezőjét az *odbc_result()* függvény segítségével olvashatjuk ki. Paraméterei az eredménytábla azonosítója valamint a keresett mező neve vagy sorszáma.

Az alábbi sor a rekord első mezőjének tartalmát adja meg:

```
$compname=odbc_result($rs,1);
```

A következő sor pedig a *CompanyName* nevű mező értékét:

```
$compname=odbc_result($rs,"CompanyName");
```

ODBC kapcsolat lezárása

ODBC kapcsolat lezárására az *odbc_close()* függvény használatos.

```
odbc_close($conn);
```

Egy ODBC példa

Az alábbi példában kapcsolódunk egy ODBC adatforráshoz, majd kiíratjuk egy tábláját egy HTML táblázatban.

```
<html>
<body>
<?php
$conn=odbc_connect('northwind','','');
if (!$conn) {
  exit("Connection Failed: " . $conn);
}

$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs) {
  exit("Error in SQL");
}
```

```
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs)) {
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
}
odbc_close($conn);
echo "</table>";
?>
</body>
</html>
```

8.4. XML kezelés

A PHP beépítve tartalmaz több XML kezelési könyvtárat is. Ebben a fejezetben ezek használatával fogunk megismerkedni.

8.4.1 Expat XML elemező

Az XML állományok kezelésének több fontos része van. Tudnunk kell létrehozni, olvasni és módosítani az XML állományainknak.

Az XML elemzőknek két fő fajtájuk van:

- A fa megközelítésű értelmezők az XML állományt fává transzformálják, és így a szabványos DOM eszközeivel könnyedén hozzá tudunk férni az elemeihez.
- Az eseménybázisú értelmezők az XML dokumentumot események sorozataként tekintik.

8.4.2 XML DOM

A W3C DOM egy alap objektumhalmazt biztosít a HTML és XML dokumentumoknak és standard interfészt a hozzáféréshez és módisításhoz.

A W3C DOM több különböző részből áll (XML, HTML, Core(mag)) és különböző szintek-ből (DOM Szintek 1/2/3):

- Core DOM – standard objektumhalmazt definiál bármilyen strukturális dokumentumhoz
- XML DOM - standard objektumhalmazt definiál XML dokumentumhoz
- HTML DOM - standard objektumhalmazt definiál HTML dokumentumhoz

XML elemző

XML dokumentumot olvasni, fejleszteni, készíteni vagy módosítani akkor neked szükséged lesz egy XML elemzőre (*parser*).

Két alap típusuk van az XML elemzőknek:

- A fa bázisú elemzők: Ez az elemző átalakítja az XML dokumentumotfa struktúrájuvá. Elemzi a teljes dokumentumot és bejárást biztosít a fa elemeihez.

- Esemény alapú elemzők: Megnézi az XML dokumentum eseménysorozatát. Mikor különös esemény előfordulást talál, hív egy függvényt és lekezeli azt.

A DOM elemző fa bázisú.

Nézzük a következő XML dokumentum részletet:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<from>Jani</from>
```

Az XML DOM megnézi XML alatti rész fa struktúráját:

1. szint: XML dokumentum
2. szint: gyökér tag: *from*
3. szint: szöveg összetevők: *Jani*

Installálás

A DOM XML elemző függvényei a PHP mag részei. Nem kell installálni ha használni szeretnénk ezeket a függvényeket.

Példa XML fájl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

XML betöltése és kiírása

Ha inicializálni akarjuk az XML elemzőt, töltsük be az XML-t és a kimenetet. Példa:

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");
print $xmlDoc->saveXML();
?>
```

A fenti példa kimenete:

```
Tove Jani Reminder Don't forget me this weekend!
```

A példaprogramunk készít egy DOM dokumentum-objektumot, és betölti az XML-t a *note.xml*-ből.

A *saveXML()* függvény átalakítja a belső XML-dokumentumot sztringgé.

Dokumentum bejárása

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");
$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " = " . $item->nodeValue . "<br />";
}
?>
```

A példa kimenete:

```
#text =
to = Tove
#text =
from = Jani
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
```

A fölül lévő példában az összes tagnál üres a *text* elem.

Mikor az XML generálódik, gyakran tartalmaz üres helyeket a csomópontok között. Az XML DOM elemző általános elemeknek kezeli, és ha nem vagyunk tudatában akkor hibát fog okozni.

8.4.3 SimpleXML

A SimpleXML a PHP 5-ös verziója óta használható. Egyszerű megoldás arra, hogy információt tudjunk kinyerni XML dokumentumból.

A SimpleXML az XML dokumentumot egy objektummá alakítja át.

- Az elemeket *SimpleXMLElement* objektumokká alakítja. Ha több, mint egy elemet tartalmaz, akkor tömbben fognak tárolódni.
- A tulajdonságokat asszociatív tömbben érhetjük el.
- Az elemek adatát szövegként kaphatjuk meg.

A SimpleXML gyors és könnyen használható például a következő egyszerű esetekben:

- XML fájlok olvasása
- Adatok kiolvasása az XML fájlokból
- Elemek, tulajdonságok szerkesztése

Bonyolultabb esetekben (pl. névterek alkalmazása) jobb az Expat Parsert vagy az XML DOM-ot használni.

Installálás

Nem szükséges installáció ahhoz, hogy használni lehessen ezeket a funkciókat.

A SimpleXML használata

Tegyük fel, hogy ki akarjuk nyerni az alábbi adatokat a fenti XML fájlból.

1. Töltjük be az XML fájlt
2. Nézzük meg a gyökérelem nevét
3. Listázzuk ki a gyermek elemek nevét és az adatát

Példa:

```
$xml = simplexml_load_file("test.xml");
echo $xml->getName() . "<br />";
foreach($xml->children() as $child) {
  echo $child->getName() . ": " . $child . "<br />";
}
```

A fenti kód eredménye lesz:

```
note
to: Tove
from: Jani
heading: Reminder
body: Don't forget me this weekend!
```

8.5. Esettanulmányok

Ebben a fejezetben néhány összetettebb alkalmazási példát fogunk megvizsgálni.

8.5.1 Beléptető-rendszer

A felhasználó azonosítása a legtöbb esetben szükséges szolgáltatás. A regisztrált látogatók számára több lehetőséget szoktunk nyújtani, mint a névtelen látogatók számára. Szinte minden esetben szükség van arra, hogy az adminisztrációs feladatokat ellátó személy (ú.n. adminisztrátor) számára a megfelelő lehetőségeket biztosítsuk. Néha a felhasználók jogosultságai feladatokhoz, csoportokba tartozáshoz is köthető.

A felhasználói jogosultságkezelés egyik része a felhasználó azonosítása (legegyszerűbben egy név-jelszó párossal). Itt most ezzel a részfeladattal fogunk foglalkozni. A feladat másik fele a felhasználók jogosultságainak kezelése.

Az itt következő nagyon egyszerű megoldás a *segedletek.hu* oldalon bemutatott *Php + mysql alapú beléptető rendszer* című cikk[60] alapján készült. A megoldás nem teljes, csupán egy kiinduló állapotnak tekintendő. Egy sokkal komolyabb, felhasználói jogosultságokat is kezelő megoldás található *PHP Login System with Admin Features*[61] címmel *jpmaster77* tollából.

Adatbázis

A felhasználók adatainak tárolásához (legegyszerűbb esetben) egyetlen táblára van szükségünk. Ennek szerkezete a következő SQL paranccsal kialakítható:

[60] http://www.segedletek.hu/segedletek/php/php_p_mysql_alapu_belepteto_rendszer.html

[61] http://www.evolt.org/PHP-Login-System-with-Admin-Features

```
CREATE TABLE users(
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
nick VARCHAR(30),
jelszo VARCHAR(32),
email VARCHAR(60)
);
```

Az *id* mező igazából csak a továbbfejlesztéshez szükséges, ebben az önálló változatban nem lesz szerepe.

Az adatbázis-kapcsolat kiépítéséhez szükséges információk a *config.php* állományba kerülnek, amit több esetben is *include*-olni fogunk:

```
<?
  $dbhost = "localhost";
  $dbuser = "root";
  $dbpass = "***";
  $dbname = "test";
?>
```

Regisztráció

A felhasználók a regisztrációs oldalon tudják magukat regisztráltatni. A regisztrációs oldalt fel kell arra készíteni, hogy a lehetséges hibák miatt az adatokat a regisztráló megfelelő hibaüzenet kíséretében visszakapja, és tudja azokat korrigálni. (A jelszavakat nem szokás visszaadni, azokat ismét ki kell tölteni.) Az ilyen típusú feladatok megoldásához egy olyan forrásállományt szokás írni, amelyik először az elküldött adatokat ellenőrzi – ha voltak egyáltalán.

A *register.php* tartalma:

```
<?
  if (isset($_POST['regisztracio'])) {
    include "config.php";
    include "reg_check.php";
    $reg_check_result = reg_check($_POST);
```

Az ellenőrzést a *reg_check* függvény végzi. (Később következik.) Ha jók az adatok, akkor azok be is kerülnek az adatbázisba:

```
    if ($reg_check_result === true)
    {
      $sql =
        "INSERT INTO belepteto_users (id,nick,jelszo,email) "
          . "VALUES('','".$_POST['nick']."','"
          . md5($_POST['pass1'])."','".$_POST['email']."')";
      $query = mysql_query($sql)
          or die ("Valami baj van az adatbázissal.");
    }
  }
?>
```

Ez után következik az oldal HTML kódja, közbe ékelve a megfelelő PHP betétekkel. Ha sikeres volt a regisztráció, egyszerűen jelezzük a regisztrált felhasználónak:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="hu">
  <head>
    <title>Regisztráció</title>
  </head>
  <body>
    <h1>Regisztráció</h1>
<? if ($reg_check_result === true) { ?>
    <p>Sikeres regisztráció!</p>
    <p><a href="index.php">Kezdő oldal</a></p>
<? } else { ?>
```

Ha nem volt sikeres, vagy még ez az oldal első letöltése, akkor következik az űrlap kódja:

```
    <form method="post" action="<?= $_SERVER['PHP_SELF']?>">
      <table>
        <tr>
          <td>Nick:<br />
            <small>legalább 4 betű</small></td>
          <td><input type="text" name="nick"
<? if ($_POST['nick']) { ?> value="<?= $_POST['nick'] ?>"<? } ?>
/></td>
          <td><?= $reg_check_result['nick'] ?></td>
        </tr>
        <tr>
          <td>E-mail cím:<br />
            <small>kötelező</small></td>
          <td><input type="text" name="email"
<? if ($_POST['email']) { ?> value="<?= $_POST['email'] ?
>"<? } ?> /></td>
          <td><?= $reg_check_result['email'] ?></td>
        </tr>
        <tr>
          <td>Jelszó:<br />
            <small>legalább 5 betű</small></td>
          <td><input type="password" name="pass1" /></td>
          <td><?= $reg_check_result['pass1'] ?></td>
        </tr>
        <tr>
          <td>Jelszó mégegyszer:</td>
          <td><input type="password" name="pass2" /></td>
          <td></td>
        </tr>
        <tr>
          <td colspan="2"><input type="submit"
name="regisztracio" value="regisztrálok" />
          </td>
        </tr>
      </table>
    </form>
<? } ?>
  </body>
</html>
```

Érdemes megfigyelni, hogy a POST adatok közül visszaküldjük a korábban kitöltött nevet és e-mail jelszót. A hibaüzeneteket a *reg_check* függvény állította elő. Érdekessége, hogy több mező hibáját is tudja egyszerre jelezni.

Megjegyzés: A felhasználó kényelme érdekében semmiképpen nem érdemes olyan megoldást alkalmazni, amelyik a hibaüzeneteket egyesével adja, rákényszerítve ezzel a felhasználót, hogy akár 5-6-szor letöltse ugyanazt az oldalt. A megoldás teljesebb lenne kliens oldali ellenőrzéssel, pl. JavaScript ellenőrizhetné a feltételek teljesülését, a felhasználónév foglaltságát akár AJAX-xal is.

Nézzük a *reg_chech.php* kódját. Az ellenőrzés során jó néhány hibaesetre fel kell készülnünk. A hibákat egy tömbben gyűjtjük. A név ellenőrzésekor a foglaltságra is figyelni kell.

```
<?
function reg_check($adatok){
  $hibak = array();
  if(strlen($adatok['nick']) < 4) {
    $hibak['nick'] = 'A nick legalább 4 betűs legyen';
  } else if (!ereg('^[a-zA-Z0-9\-\_áéíóöüóűÁÉÍÓÖÜŐŰ]+$',
      $adatok['nick' ])) {
    $hibak['nick'] = 'A nickben csak a magyar ábécé kis - és
nagybetűi, a számok, illetve a  _ és - jelek engedélyezettek';
  } else {
    include "config.php";
    mysql_connect($dbhost,$dbuser,$dbpass);
    mysql_select_db($dbname);
    $sql = 'SELECT * FROM belepteto_users'
        . ' WHERE nick="' . $adatok['nick']. '"';
    $query = mysql_query($sql)
        or die ("Valami baj van az adatbázissal.");
    if (mysql_num_rows($query) !== 0) {
      $hibak['nick'] = 'A nick már foglalt';
    }
  }
```

Ehhez képest a többi adat ellenőrzése nem túl bonyolult:

```
  if($adatok['email'] == '') {
    $hibak['email'] = 'Az e-mail cím kitöltése kötelező';
  } else if ($adatok['email'] == (!eregi('^[_\.0-9a-z-]+@([0-9a-
z][0-9a-z-]+\.)+[a-z]{2,6}$',$adatok['email']))) {
    $hibak['email'] = 'Hibás e - mail cím';
  }
  if (strlen($adatok['pass1']) < 5) {
    $hibak['pass1'] = 'A jelszó legalább 5 betűs legyen';
  }
  if ($adatok['pass1'] !== $adatok['pass2']) {
    $hibak['pass1'] = 'A két jelszó nem egyezik';
  } else if ($adatok['pass1'] == '' || $adatok['pass2']==''){
    $hibak['pass1'] = 'A jelszavak kitöltése kötelező';
  }
  if ($hibak)
    return $hibak;
  else
    return true;
}
?>
```

A kezdőoldal (*index.php*) csupán a bejelentkezést és a védett tartalom elérését teszi lehetővé:

```
<?
  session_start();
  ob_start();
?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="hu">
  <head>
    <title>Beléptető rendszer</title>
  </head>
  <body>
    <h1>Beléptető rendszer</h1>
<?php include "login.php"; ?>
    <p>Ez itt a mindenki számára elérhető rész </p>
    <p><a href="vedett.php">Védett rész</a></p>
  </body>
</html>
<?
ob_end_flush();
?>
```

A *login.php* csak akkor kér azonosítási adatokat, ha a belépést jelző munkamenet-változónak nincs értéke. Ha már újbóli betöltés, és POST adatok feldolgozása szükséges, akkor azt is megtesszük:

```
<?
if ($_SESSION['belepett']!== true) {
  if (isset($_POST['login']))   {
    $nick = addslashes($_POST['nev']);
    $pass = md5($_POST['jelszo']);
    $sql = "SELECT * FROM belepteto_users "
        . "WHERE (nick='".$nick."'"
        . " AND jelszo='".$pass."')";
    include "config.php";
    mysql_connect($dbhost,$dbuser,$dbpass);
    mysql_select_db($dbname);
    $query = mysql_query($sql);
    if (mysql_num_rows($query) !== 0) {
      $_SESSION['nick'] = addslashes($_POST['nev']);
      $_SESSION['belepett'] = true;
      header("Location: ".$_SERVER['PHP_SELF']);
    } else { ?>
      <p>Hibás nick/jelszó</p>
<?  }
  } ?>
```

```
      <form action="<?= $_SERVER['PHP_SELF'];?>" method="post">
        <table>
          <tr>
            <td>nick:</td>
            <td><input type="text" name="nev" /></td>
          </tr><tr>
            <td>jelszó:</td>
            <td><input type="password" name="jelszo" /></td>
          </tr><tr>
            <td colspan="2" align="center"><input type="submit"
                name="login" value="Belépés" /></td>
          </tr>
        </table>
      </form>
      <p><a href="register.php">regisztráció</a></p>
<? } else { ?>
      <p>Bejelentkezve: <?= $_SESSION['nick'] ?></p>
      <p><a href="logout.php">kijelentkezés</a></p>
<? } ?>
```

A *vedett.php* oldal lehet a sémája minden olyan oldanak, amit csak a jogosult felhasználók használhatnak. A $_SESSION['belepett'] változó vizsgálatával állapítható meg, hogy belépett felhasználó szeretné-e letölteni az oldalt.

```
<?
  session_start();
  ob_start();
?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="hu">
  <head>
    <title>Védett tartalom</title>
  </head>
  <body>
<?
if($_SESSION['belepett'] == true) { ?>
    <p>Védett tartalom, ha ezt olvasod, sikerült belépned,
gratulálok</p>
    <p><a href="index.php">kezdő oldal</a></p>
<? } else { ?>
    <p>Csak szeretnéd... :)</p>
<? } ?>
  </body>
</html>
<?
ob_end_flush();
?>
```

Végül a kilépés lehetőségét nézzük meg a *logout.php* alapján:

```
<?
session_start();
unset($_SESSION['belepett']);
unset($_SESSION['nick']);
header("location: index.php");
?>
```

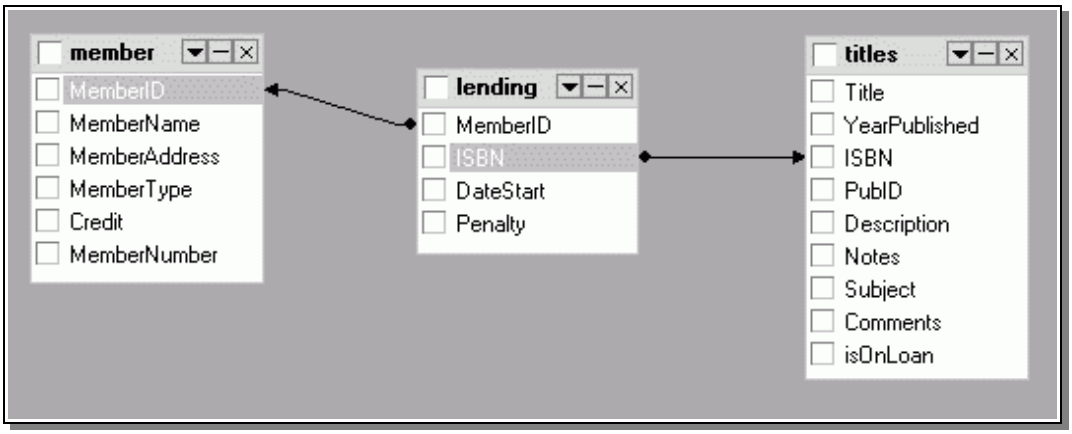
8.5.2 Könyvtári kölcsönző rendszer

A Pannon Egyetem Információs Rendszerek Tanszékén található[62] egy nagyon tömör PHP-MySQL bevezetés, nagyrészt egy könyvtári kölcsönző rendszer fejlesztésének példáján bemutatva. A példa alkalmazásból itt csak néhány tervezési lépést említünk meg.

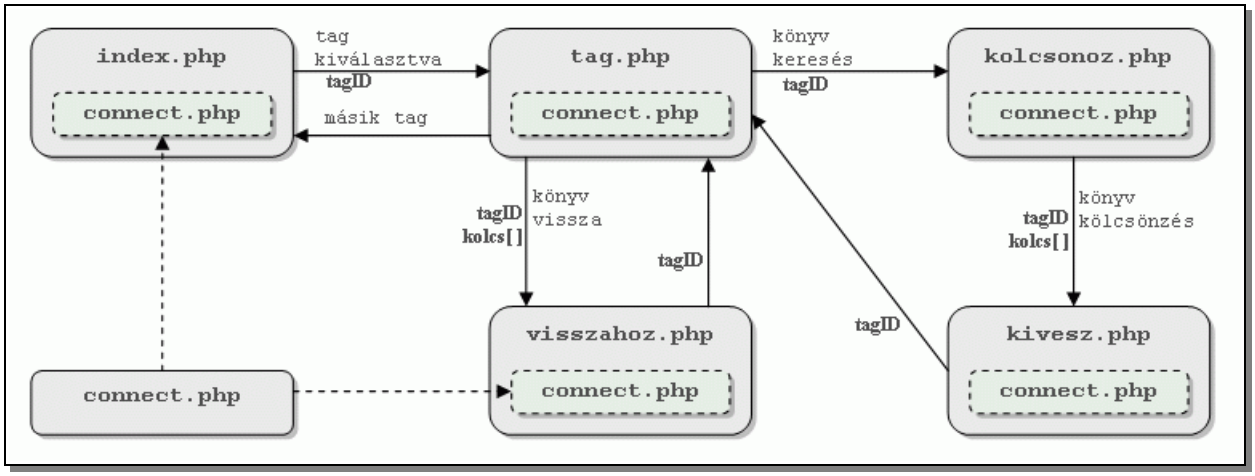
Feladat

Egy könyvtári nyilvántartó rendszer létrehozása, ami egy tag-szám alapján azonosítja a könyvtári tagot, majd megmutatja a kölcsönzéseit. A tag kereshet a könyvek között (legalább a címtöredék alapján), majd kölcsönözheti azokat. Visszavitelkor leveszi a kölcsönzött könyvek listájáról az adott tételt.

Először is nézzük meg az alkalmazás adatbázis-tervét:



A fenti honlapról letölthető kész alkalmazás navigációs és szerkezeti vázlata:



A nyilak az egyes PHP oldalak közötti átmeneteket jelképezik: vékonnyal a felhasználói művelet, vastaggal a két oldal között átadott információ látható.

8.6. További források

- PHP kézikönyv
  http://www.php.net/manual/hu/

[62] http://www.irt.vein.hu/~bvegso/php_doc/

- George S. Schlossnagle: PHP fejlesztés felsőfokon
  Kiskapu Kiadó

8.7. Feladat

A következő feladat egy elég komplex összefoglalását adja a webfejlesztés eddigi témáinak. A feladat egy dinamikus űrlap generáló osztály elkészítése.

Írjunk olyan űrlap generáló osztályt, ami megadott bemenő paraméterek alapján legenerál egy komplett, működő HTML űrlapot. Az űrlapot és a mezőket beágyazott CSS segítségével formázzuk.

Az űrlap legeneráláshoz szükséges adatokat adatbázisban tároljuk. (Az adatbázis feltöltéséhez nem kell programot írni.)

A legenerált űrlapot HTML fájlként mentsük.

Az űrlap paraméterei:

- az űrlap címe,
- az űrlap bevezető szövege,
- az űrlap *action* paramétere,
- az űrlap metódusa (*get* vagy *post*),
- az űrlapon megjelenő mezők adatai,
- a *submit* gomb felirata.

Az űrlapon megjelenő mezők adatai:

- sorszám (hányadikként jelenjen meg)
- mező leírása (az űrlap elem előtt megjelenő szöveg)
- mezőnév (az elem *name* paramétere)
- mező hossza (szöveges mező esetén) – hosszú szöveg esetén sorok és oszlopok száma
- mező típusa

A mező típusa szöveges (*text*), hosszú szöveg (*textarea*) vagy *checkbox* lehet.

A megvalósításnál a fő szempont a modularitás, objektum orientál felépítés és paraméterezhetőség.

A feladat elkészítése után érdemes megnézni két megoldást, amelyek a fenti célok mellett a validálást is megvalósítják:

- Building a PHP 5 Form Processor: Coding the Form Generator Module[63]
- Clonefish[64]

[63] http://www.devshed.com/c/a/PHP/Building-a-PHP-5-Form-Processor-Coding-the-Form-Generator-Module/

[64] http://www.phpformclass.com/page/index

9. Tervezési minták

A webfejlesztés során használt tervezési minták (*design patterns*) nagyban segíthetik a munkánkat, hiszen mások bevált ötletei alapján valószínűleg gyorsabb és hatékonyabb alkalmazást készíthetünk, mint csupán a saját ötleteinkre és tapasztalatainkra támaszkodva.

A weben igen sok (angol nyelvű) forrás található a témakörben. A szerző véleménye szerint az egyik legjobb oldal a phpPatterns[65] oldala. Ezen kívül a Zend *PHPPatterns: Instructions*[66] cikke is nagyon hasznos.

Mi a tervezési minta?

Ha egy feladat újra előkerül a fejlesztés folyamán, akkor valószínűleg a megoldás hasonló lesz a korábbihoz. A tervezési minták olyan objektumközpontú megoldásokat jelentenek, amelyek már bizonyítottak a gyakorlatban. Ezek felhasználása rugalmasabban módosítható és könnyebben, jobban újrahasznosítható alkalmazásokat készíthetünk.

E fejezetben néhány alapvetőbb tervezési mintát fogunk megvizsgálni.

9.1. Front Controller

A *Front Controller* minta a weboldalhoz érkezett kérés feldolgozásához és a kérések kiszolgálásának vezérléséhez használt módszer. (A Front Controller nagyon közel áll az MVC minta Controller eleméhez.)

Statikus példa

A statikus megoldás lehet pl. a következő:

```
<?php // index.php
switch ( @$_GET['action'] ) {
  case 'edit':
    include ('actions/edit.php'); break;
  case 'post':
    include ('actions/post.php'); break;
  case 'delete':
    include ('actions/delete.php'); break;
  case 'default':
    include ('actions/view.php'); break;
} ?>
```

Jól látható, hogy itt a lehetőségeket előre beleégetjük a forráskódba.

Dinamikus példa

A dinamikus megoldás nem igényli a lehetőségek (az előző példában akciók) felsorolását, hanem valamely más forrásból (pl. fájlrendszer, adatbázis, konfigurációs adatok) deríti ki az érvényes kérések körét.

---

[65] http://www.phppatterns.com/

[66] http://devzone.zend.com/node/view/id/3

Nézzünk meg egy egyszerűbb változatot, ami konfigurációs fájlban, tömbben tárolja az egyes oldalak adatait. (A példa könnyen továbbfejleszthető abba az irányba, hogy tömb helyett adatbázisból olvassuk ki az oldalaink adatait.) A példa 3 egyedi és több hasonló forrásfájlból épül fel.

Az index.php

Az *index.php* feladata a honlap feldolgozási folyamatának indítása, vezérlése. Először is a konfigurációs adatokat töltjük be:

```
include('config.inc.php');
```

A *$keres* változóba állítjuk elő, hogy melyik oldal lekérése is történt. Először az *oldal* GET paraméter hiányára készülünk fel, majd az *$oldal* tömb alapján érvényesítjük a kérést:

```
$keres = current($oldalak);
if (isset($_GET['oldal'])) {
  if (isset($oldalak[$_GET['oldal']])) {
    $keres = $oldalak[$_GET['oldal']];
  } else {
    $keres = $hiba_oldal;
    header("HTTP/1.0 404 Not Found");
  }
}
```

Látszik, hogy hibás kérésre is fel kell készülnünk. Végül az *index.tpl.php* sablon fájl fejezi be a feladatot:

```
include('index.tpl.php');
```

A config.inc.php

Ebben a fájlban olyan konfigurációs adatokat helyezünk el, amelyek minden oldallekérés esetén szükségesek lesznek.

A fejléc a *title* és az oldal főcíme számára is tartalmaz információt:

```
$fejléc = array(
  'cím' => 'Nagy Gusztáv',
  'mottó' => 'Szakmai és személyes oldal'
);
```

Az *$oldalak* tömb a főmenüben szereplő menüpontokat és a hozzájuk tartozó oldalak adatait tartalmazza. Első eleme a kezdőoldalra utal.

```
$oldalak = array(
  '/' =>
    array('fájl' => 'kezdo', 'szoveg' => 'Kezdőlap'),
  'magamrol' =>
    array('fájl' => 'magamrol', 'szoveg' => 'Magamról'),
  'szakmai_oneletrajz' =>
    array('fájl' => 'szakmai_oneletrajz', 'szoveg' =>
      'Szakmai önéletrajz'),
  'galeriak' =>
    array('fájl' => 'galeriak', 'szoveg' => 'Galériák'),
  'kapcsolat' =>
    array('fájl' => 'kapcsolat', 'szoveg' => 'Kapcsolat'),
);
```

Végül a hibaoldal számára is definiálunk információkat:

```
$hiba_oldal =
  array ('fájl' => '404',
    'szoveg' => 'A keresett oldal nem található');
```

Természetesen ha további, a főmenün kívüli oldalak is lesznek a honlapunkon, akkor újabb tömb segítségével azon is definiálhatók lennének. (Persze ekkor az *index.php* is összetettebb kell legyen.)

Az index.tpl.php

Az index.tpl.php fájl a honlap oldalról oldalra változatlan sablonját, valamint a változó részek logikáját valósítja meg. A következő *head* rész példája jól mutatja a két rész együttesét:

```
<head>
  <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
  <title>
    <?= $fejléc['cím'] .
      ( (isset($fejléc['mottó']))
        ? ('|' . $fejléc['mottó']) : '' )
    ?>
  </title>
  <link rel="stylesheet" href="style.css" type="text/css" />
</head>
```

A minden oldalon egyformán megjelenő HTML tagok között szereplő PHP rész a *config.inc.php*-ből származó adatokat helyettesíti be.

Persze egyszerű behelyettesítés mellett a PHP valamivel összetettebb feladatot is megoldhat. Például a következő ciklus és feltételes utasításokkal pont az igényeknek megfelelő eredményt produkálhatunk:

```
<div id="header">
  <h1>
    <?= $fejléc['cím'] ?>
  </h1>
<? if (isset($fejléc['mottó'])) { ?>
  <h2>
    <?= $fejléc['mottó'] ?>
  </h2>
<? } ?>
</div>

<div id="topnav">
  <ul>
<? foreach ($oldalak as $url => $oldal) { ?>
    <li<?= (($oldal == $keres) ? ' class="aktiv"' : '') ?>>
      <a href="<?= ($url == '/') ? '.'
                                 : ('?oldal=' . $oldal['fájl'])
        ?>"><?= $oldal['szoveg'] ?></a>
    </li>
<? } ?>
  </ul>
</div>
```

Az egyes oldalak

Az egyes oldalak változó törzse jelen példánkban az *oldal* könyvtárban található. Az *index.tpl.php* következő része illeszti be az oldalba:

```
<div id="body">
<?
  include("oldal/{$keres['fájl']}.tpl.php");
?>
</div>
```

Példánk végén érdemes átgondolni e verzió sablon elvű megközelítését.

Természetesen érdemes a példa hiányosságait is áttekinteni:

- az adatok nem adatbázisból, hanem fájlrendszerből származnak, és
- csak egyszerű főoldal és főmenü struktúrát valósít meg.

Objektumorientált példa

Chris Corbyn[67] megoldása[68] egy egyszerű vendégkönyv alkalmazás vázlatát mutatja be.

Először nézzük meg, milyen URL-eket kellene feldolgoznunk:

```
http://localhost/index.php?page=guestbook&action=index
http://localhost/index.php?page=guestboook&action=createPost
```

Az első kérés a bejegyzések listázását, a második egy új bejegyzés létrehozása esetén lesz használatos.

Először az *index.php* oldal kapja meg a vezérlést. A *FrontController* osztály betöltésén túl a példányosítás (*createInstance*) és az oldal feldolgozása (*dispatch*) is megtörténik:

```
<?php
  define("PAGE_DIR", dirname(__FILE__) . "/pages");
  require_once "FrontController.php";
  FrontController::createInstance()->dispatch();
?>
```

A *FrontController.php*-t vizsgálva láthatjuk, hogy miért nem a konstruktort használjuk a példányosításhoz:

```
<?php
class FrontController {
  public static function createInstance() {
    if (!defined("PAGE_DIR")) {
      exit("Critical error: Cannot proceed without PAGE_DIR.");
    }
    $instance = new self();
    return $instance;
  }
```

A példányosítás csak definiált PAGE_DIR esetén hajtódik végre.

Az osztály lényege a következő *dispatch* metódusban van. Az URL *page* és *action* értékeit kinyerjük, illetve az alapértelmezett értéket töltjük be:

---

[67] http://www.w3style.co.uk

[68] http://www.w3style.co.uk/a-lightweight-and-flexible-front-controller-for-php-5

```
  public function dispatch() {
    $page = !empty($_GET["page"]) ? $_GET["page"] : "home";
    $action = !empty($_GET["action"]) ? $_GET["action"] :
        "index";
```

Összeállítjuk az osztályt tartalmazó PHP oldal nevét (*$class*), majd próbáljuk betölteni:

```
    $class = ucfirst($page) . "Actions";
    $file = PAGE_DIR . "/" . $page . "/" . $class . ".php";
    if (!is_file($file)) {
      exit("Page not found");
    }
    require_once $file;
```

Végül betölti az oldalnak megfelelő osztályt, példányosírja, és – ellenőrzötten – meghívja az akciót végrehajtó metódust:

```
    $actionMethod = "do" . ucfirst($action);
    $controller = new $class();
    if (!method_exists($controller, $actionMethod)) {
      exit("Page not found");
    }
    $controller->$actionMethod();
    exit(0);
  }
}
```

A két korábbi példa URL-ünk esetén a *pages/guestbook/GuestbookActions.php* fájlt tölti be, és meghívja a két példa metódusunkat:

```
<?php
class GuestbookActions {
  public function doIndex() {
    echo "Index action called…";
  }

  public function doCreatePost() {
    echo "CreatePost action called…";
  }
}
```

Érdemes tovább nézni Chris Corbyn példáját, konplex MVC megoldáshoz közelít.

9.2. Strategy

A stratégia egy objektumba zárt algoritmus. Az egységbezárás miatt az algoritmus jól elkülöníthető a kód más részeitől, az öröklődés miatt pedig a stratégia könnyen lecserélhető egy másikra.

Validátor példa

A fejezet példája[69] segítségével az űrlap által küldött adatok érvényességének ellenőrzését (validálását) fogjuk elvégezni.

Nézzünk egy hagyományos, nem túl praktikus megközelítést:

[69] A példa a http://www.phppatterns.com/docs/design/strategy_pattern oldalról származik

```
if ( isset ($_POST['submit']) ) {
    if ( $_POST['user'] < '6' ) {
        echo ('Username is too short');
    } else if ( $_POST['pass'] != $conf ) {
        echo ('Passwords do not match');
    } else if ( $_POST['fullmoon'] == true ) {
        // ...
    }
}
```

Látszik, hogy a kód kissé zavaros, nehezen áttekinthető és módosítható.

Nézzük meg egy jobb megoldást. A *Validator* (logikailag[70]) egy absztrakt osztály, a lényegi munka a leszármazottakban fog történni.

```
class Validator {
    var $errorMsg;

    function Validator () {
        $this->errorMsg=array();
        $this->validate();
    }

    function validate() { }

    function setError ($msg) {
        $this->errorMsg[]=$msg;
    }

    function isValid () {
        if ( isset ($this->errorMsg) ) {
            return false;
        } else {
            return true;
        }
    }

    function getError () {
        return array_pop($this->errorMsg);
    }
}
```

Jól látszik, hogy az ellenőrzés során a hibák felismerése a hibaüzenetek megfogalmazását is jelenti. A hibák az *$errorMsg* tömbben gyűlnek.

A következő osztály a felhasználónév érvényességét ellenőrzi:

```
class ValidateUser extends Validator {
    var $user;

    function ValidateUser ($user) {
        $this->user=$user;
        Validator::Validator();
    }
```

[70] A PHP 4-es verziója még nem ismeri az absztrakt osztály fogalmát

```
    function validate() {
        if (!preg_match('/^[a-zA-Z0-9_]+$/',$this->user )) {
            $this->setError(
                'Username contains invalid characters');
        }
        if (strlen($this->user) < 6 ) {
            $this->setError('Username is too short');
        }
        if (strlen($this->user) > 20 ) {
            $this->setError('Username is too long');
        }
    }
}
```

Érdemes megfigyelni, hogy ez a kód épít a PHP 4-es verziójának azon specialitására, ami szerint az ősosztály konstruktorát nem hívja meg automatikusan a leszármazott konstruktora. (Ebben az esetben az osztály logikája nem is működne.)

A jelszó és az e-mail cím ellenőrzése hasonló logikára épül:

```
class ValidatePassword extends Validator {
    var $pass;
    var $conf;

    function ValidatePassword ($pass,$conf) {
        $this->pass=$pass;
        $this->conf=$conf;
        Validator::Validator();
    }

    function validate() {
        if ($this->pass!=$this->conf) {
            $this->setError('Passwords do not match');
        }
        if (!preg_match('/^[a-zA-Z0-9_]+$/',$this->pass )) {
            $this->setError(
                'Password contains invalid characters');
        }
        if (strlen($this->pass) < 6 ) {
            $this->setError('Password is too short');
        }
        if (strlen($this->pass) > 20 ) {
            $this->setError('Password is too long');
        }
    }
}

class ValidateEmail extends Validator {
    var $email;
    function ValidateEmail ($email){
        $this->email=$email;
        Validator::Validator();
    }
```

```
    function validate() {
        $pattern=    "/^([a-zA-Z0-9])+([\.a-zA-Z0-9_-])*@([a-zA-
Z0-9_-])+(\.[a-zA-Z0-9_-]+)+/";
        if(!preg_match($pattern,$this->email)){
            $this->setError('Invalid email address');
        }
        if (strlen($this->email)>100){
            $this->setError('Address is too long');
        }
    }
}
```

Az ellenőrzés megvalósítása az osztályaink felhasználásával már egyszerű:

```
<?php
if ( $_POST['register'] ) {
 require_once('lib/Validator.php');
 $v['u']=new ValidateUser($_POST['user']);
 $v['p']=new ValidatePassword($_POST['pass'],$_POST['conf']);
 $v['e']=new ValidateEmail($_POST['email']);
    foreach($v as $validator) {
        if (!$validator->isValid()) {
            while ($error=$validator->getError()) {
                $errorMsg.="<li>".$error."</li>n";
            }
        }
    }
    if (isset($errorMsg)) {
        print (
            "<p>There were errors:<ul>n".$errorMsg."</ul>");
    } else {
        print ('<h2>Form Valid!</h2>');
    }
} else { ?>

<h2>Create New Account</h2>
<form action="<?php echo ($_SERVER['PHP_SELF']); ?>"
method="post">
<p>Username: <input type="text" name="user"></p>
<p>Password: <input type="password" name="pass"></p>
<p>Confirm: <input type="password" name="conf"></p>
<p>Email: <input type="text" name="email"></p>
<p><input type="submit" name="register" value="Register"></p>
</form>
<?php } ?>
```

Általános eseménynaplózás

Nagyon szép példa található a Patterns For PHP oldalán:

http://www.patternsforphp.com/wiki/Strategy

9.3. Data Access Object

Az adatbázis elérés megvalósítása nem triviális feladat. Sokféle megoldás létezik, ezért sokszor nem könnyű a választás sem. A DAO tervezési minta egy igen sokrétű megoldá-

sának csupán egy részével fogunk foglalkozni. A teljes példa megtalálható a phpPatterns oldalán[71].

Nézzük először az adatbázis-kapcsolat felépítéséért és az SQL parancsok futtatásáért felelős *DataAccess* osztályt:

```
class DataAccess {
    var $db;
    function DataAccess ($host,$user,$pass,$db) {
        $this->db=mysql_pconnect($host,$user,$pass);
        mysql_select_db($db,$this->db);
    }

    function & fetch($sql) {
        return new DataAccessResult(
                $this,mysql_query($sql,$this->db));
    }

    function isError () {
        return mysql_error($this->db);
    }
}
```

Érdemes megfigyelni, hogy SQL lekérdezés futtatása esetén *DataAccessResult* típusú objektumot fog kiszolgálni a *fetch* metódus. A visszaadott objektumból kinyerhető a lekérdezés hibái vagy eredményei is.

```
class DataAccessResult {
    var $da;
    var $query;

    function DataAccessResult(& $da,$query) {
        $this->da=& $da;
        $this->query=$query;
    }

    function getRow () {
        if ( $row=mysql_fetch_array(
                $this->query,MYSQL_ASSOC) )
            return $row;
        else
            return false;
    }

    function rowCount () {
        return mysql_num_rows($this->query);
    }

    function isError () {
        $error=$this->da->isError();
        if (!empty($error))
            return $error;
        else
            return false;
    }
}
```

Nézzük meg a névadó osztályt, amely a lekérdezések típusai között is különbséget tesz:

[71] http://www.phppatterns.com/docs/design/data_access_object_pattern_more_widgets

```
class Dao {
    var $da;

    function Dao ( & $da ) {
        $this->da=$da;
    }

    function & retrieve ($sql) {
        $result=& $this->da->fetch($sql);
        if ($error=$result->isError()) {
            trigger_error($error);
            return false;
        } else {
            return $result;
        }
    }

    function update ($sql) {
        $result=$this->da->fetch($sql);
        if ($error=$result->isError()) {
            trigger_error($error);
            return false;
        } else {
            return true;
        }
    }
}
```

A DAO osztály leszármazottai képesek arra, hogy az egyes konkrét feladatok esetén a saját igényeiknek megfelelő adatbázis-eléréseket valósítsanak meg.

Példánk utolsó bemutatott osztálya (LogDAO) egy konkrét feladatot lát el: log információkat tartalmazó táblával tud kommunikálni. A tábla szerkezete:

```
CREATE TABLE log (
  id int(11) NOT NULL auto_increment,
  host char(100) NOT NULL default '',
  address char(100) NOT NULL default '',
  agent char(100) NOT NULL default '',
  date datetime default NULL,
  country char(50) NOT NULL default '',
  provider char(100) NOT NULL default '',
  os char(50) NOT NULL default '',
  wb char(50) NOT NULL default '',
  PRIMARY KEY  (id),
  KEY id (id)
) TYPE=MyISAM;
```

A leszármazott először is az ős konstruktorát hívja:

```
class LogDao extends Dao {
function LogDao ( & $da ) {
        Dao::Dao($da);
    }
```

A további függvények a konkrét feladathoz szükséges lekérdezés-típusokat rejtik el a kód további részétől. Például a teljes listázást (esetleg intervallum megadással) végző *searchAll* függvény a paraméterei alapján összeállítja az SQL parancsot, majd (mivel SELECT típusú lekérdezésre van szükség) az őse *retrieve* függvényét kéri a tényleges munkavégzésre.

```
function & searchAll ($start=false,$rows=false) {
    $sql="SELECT * FROM log ORDER BY date DESC";
    if ( $start ) {
        $sql.=" LIMIT ".$start;
        if ( $rows )
            $sql.=", ".$rows;
    }
    return $this->retrieve($sql);
}
```

Az áttekintett négy osztály közül az első kettő (DataAccess és DataAccessResult) tartalmazza az adatbázisszerver-típusából (jelen esetben MySQL) adódó specialitásokat, de a további osztályoknak már ezzel nem kell foglalkozni. A DAO osztály leszármazottai pedig a konkrét problémákra tudnak koncentrálni. A phpPatterns eredeti példája teljes MVC integrációt is tartalmaz, további érdekes megoldásokkal egy komplex honlap-motor fejlesztéséhez.

9.4. MVC

Az MVC[72] (model – view – controller, modell – nézet - vezérlő) egy jól használható, kipróbált módszer arra, hogy hogyan válasszuk szét a felhasználói felületet és az üzleti logikát. Az elsődleges cél az, hogy a felhasználói felület megjelenítéséért felelős kódot teljesen elkülönítsük. Ezáltal annak módosítása, kiegészítése nem vonja maga után az üzleti logikát megtestesítő kód módosítását, vagy megismétlését.

A módszer lényege az, hogy a hagyományos eljárás alapú programok adatbevitel-adatfeldolgozás-eredmény megjelenítése feladatokat leképezzék a grafikus felhasználói felülettel rendelkező programokban:

adatbevitel→	adatfeldologzás→	eredmény megjelenítése
Controller (Vezérlő) →	Model (Modell) →	View (Nézet)

A **vezérlő** dolgozza fel a felhasználói adatbevitelt. Parancsokká és függvényhívásokká képezi le azokat. Ezek a parancsok fogják előidézni az adatok módosítását, törlését, vagy a nézetek megváltozását. Például ha a felhasználó kiválasztja a menü egyik elemét, akkor egy controller fogja meghatározni, hogy ennek hatására mi is történjen.

A **modell** reprezentálja az üzleti logikát, feladata értelmet adni a nyers adatoknak. Ez az egység felelős pl. egy számla áfa tartalmának és végösszegének kiszámolásáért. A Model tudja, hogy melyik vevő fogja kifizetni a számlát, és ha szükséges, azt is, hogy éppen születésnapja van-e ennek a vevőnek.

A **nézet** feladata a felhasználói felület megjelenítése. Ez az űrlapokat, táblázatokat, linkeket, gombokat, szövegeket jelent. Ezek az elemek megjelenhetnek egy szabványos HTML oldalon, azonban ha egy Java nyelvű alkalmazásról van szó, akkor az AWT vagy a Swing könyvtárra épülő grafikus felületet jelent.

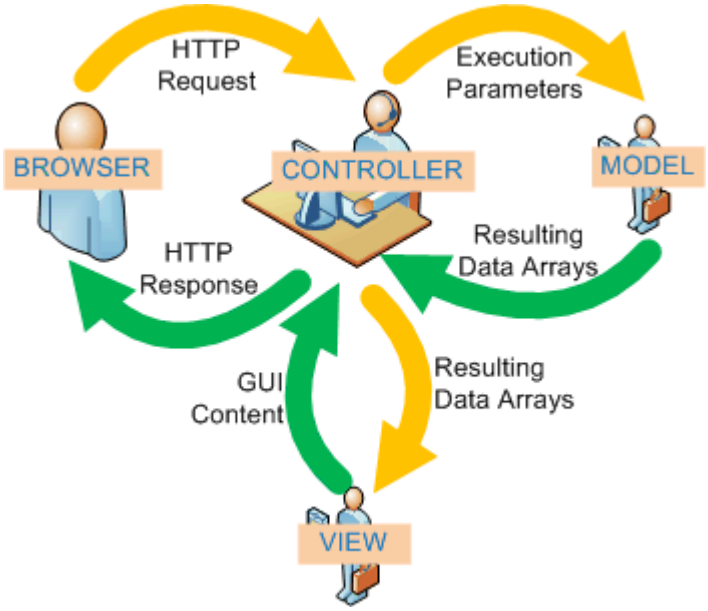
Vegyük egy konkrét gyakorlati PHP fejlesztői példát. Feladatunk egy felmérés eredményeinek megjelenítése táblázatban.

Hagyományos módszerrel a PHP forrásunk tartalmaz egy SQL lekérdezést, majd az adatokat egy ciklussal kiírja egy táblázatba. Ha az ügyfelünk ugyanezen adatokat egy grafikonon is látni szeretné, akkor létrehozunk egy másik fájlt, ami szintén tartalmazni fogja az SQL kérést, és az adatok megjelenítését.

---

[72] Barkóczi Roland szakdolgozata (http://blog.aer.hu/) alapján

Az MVC tervezési minta ezzel szemben tartalmazni fog egy modellt, ami felelős az adatbázis kezeléséért, és két nézetet, ami felelős az adatok megjelenítéséért. Azt, hogy melyik nézetet kell megjeleníteni, a vezérlő dönti el a felhasználó kívánsága szerint. Ha további nézetekre van szükségünk (pl. kördiagramm vagy esetleg egy szöveges magyarázatokkal ellátott kimutatásra), akkor csak a megfelelő nézetfájlokat kell elkészíteni, amelyek a modellből származó adatokat használják fel.



Ezt a tervezési mintát Trygve Reenskaug írta le 1979-ben, amikor grafikus felületű SmallTalk alkalmazásokon dolgozott a Xerox-nál. A SmallTalk MVC nagy sikert aratott, rengeteg grafikus keretrendszerben került felhasználásra.

9.4.1 Nem objektum-orientált megvalósítás

Bár az MVC mintát objektum-orientált módon szokás megvalósítani, a megértésben segíthet egy nagyon egyszerű (már-már primitív) megvalósítás[73]. Természetesen az MVC hármas felépítése így is tökéletesen bemutatható.

Simple-Non-OO-MVC-Controller.php

A másik két „tag” betöltése után a vezérlő a *Front Controller*-hez hasonlóan működik:

```
// copyright Lawrence Truett and FluffyCat.com 2006,
// all rights reserved
  include_once('Simple-Non-OO-MVC-Model.php');
  include_once('Simple-Non-OO-MVC-View.php');

  $page = getPage();
  if ("TITLE" == $page) {
    $title = getTitle();
    showTitlePage($title);
  } elseif ("AUTHOR" == $page) {
    $author = getAuthor();
    showAuthorPage($author);
  }
```

[73] http://www.fluffycat.com/PHP-Design-Patterns/Non-OO-MVC/

```
function getPage() {
   if (isset($_GET["page"])) {
     $pageIn = $_GET["page"];
     switch($pageIn) {
       case "title":
         $page = "TITLE";
       break;

       case "author":
         $page = "AUTHOR";
       break;

       default:
         $page = "TITLE";
       break;
     }
   } else {
     $page = "TITLE";
   }

   return $page;
}
```

Simple-Non-OO-MVC-Model.php

```
function getAuthor() {return 'Larry Truett';}
function getTitle() {return 'PHP for Cats';}
```

A modell függvények akár adatbázisból is olvashatnának.

Simple-Non-OO-MVC-View.php

```
function showAuthorPage($authorIn) {
  $page = htmlBegin() . 'AUTHOR: '.$authorIn.htmlEnd();
  echo $page;
}

function showTitlePage($titleIn) {
  $page = htmlBegin() . 'TITLE: '.$titleIn.htmlEnd();
  echo $page;
}

function htmlBegin() {
  return '<'.'HTML><'.'HEAD><'.'TITLE><'
      '/TITLE><'.'/HEAD><'.'BODY>';
}

function htmlEnd() {
  return '<'.'/BODY><'.'/HTML>';
}
```

A nézet a kapott adatok és a változatlan szövegek felhasználásával a kimenetet állítja elő.

9.4.2 Objektum-orientált megvalósítás

Az MVC elvét számtalan (többé-kevésbé eltérő) módon megvalósították már. A jegyzet jelen verziójában nincs lehetőségünk ezek alaposabb megismerésére :-(

9.5. További források

- Patterns for PHP
  http://www.patternsforphp.com/
- PHP Design Patterns
  http://www.fluffycat.com/PHP-Design-Patterns/
- Stig Bakken, Andi Gutmans, Derick Rethans:
  PHP 5 Advanced OOP and Design Patterns
  http://www.informit.com/articles/article.aspx?p=346958&seqNum=4

10.Sablonrendszerek

Köszönet: E fejezet jelentősen épít Boda Csaba *PHP alapú sablon rendszerek áttekintése* című szakdolgozatára.

Az asztali alkalmazásfejlesztésben az 1980-as évektől kezdve meghonosodott a MVC (Modell-View-Controller) tervezési minta.

A webfejlesztésben sokszor egy ennél egyszerűbb módszert alkalmazunk, vagy a nézet felépítését is a sablonozásra bízhatjuk. Ennek lényege, hogy az alkalmazás logikát és a megjelenítési kódot a lehető legtisztábban válasszuk el egymástól. Így bármelyik komponens változása esetén a másik komponenshez csak kis részben, vagy egyáltalán nem kell hozzányúlni.

Az alkalmazás logika és a megjelenítési kód szétválasztásának további előnye, hogy az alkalmazás logikáját megvalósító kódon dolgozó fejlesztő, és az oldal megjelenésével foglalkozó dizájner akár párhuzamosan, egymás zavarása nélkül is végezheti a dolgát. Ráadásul egymás munkáját is elég alapfokon ismerni az együttműködéshez.

10.1. Smarty

A Smarty[74] az elmúlt években méltán vált az első számú (és legtöbb fejlesztő számára az egyetlen) sablon-motorrá. A mai változata már több mint egy sablonmotor, talán pontosabb lenne sablon-keretrendszernek nevezni.

Jellemzők

Bevezetésül a Smarty néhány fontos jellemzőjét fogjuk megvizsgálni.

Gyorstárazott

A Smarty lehetőséget nyújt az oldalaink lefordított állapotának eltárolására két különböző szinten is. Természetesen ez a lehetőség finoman konfigurálható, nem csak globálisan, hanem egyes oldalakra nézve is ki-be kapcsolható.

Konfigurációs állományok

Egy vagy több oldalon is felhasználhatók a külön állományokban tárolt konfigurációs információk. Ráadásul a konfigurációs beállításokat a dizájner önállóan is menedzselni tudja, nincs szükség a programozó beavatkozására.

Biztonságos

A sablonok nem tartalmaznak PHP kódot. Ezzel csökkentjük ugyan a dizájner lehetőségeinek körét, de ez a legtöbb esetben inkább előnyös, mint hátrányos. Ráadásul van arra is lehetőség, hogy ellenőrzött módon kiterjesszük a Smarty lehetőségeit, tehát a „probléma” áthidalható.

Könnyen kezelhető és fenntartható

A dizájnernek nem kell a PHP bonyolult szintaxisával megismerkedni, helyette egy (a HTML-hez sok tekintetben közelebb álló) sablon-nyelvet kell alkalmazni.

---

[74] http://smarty.php.net/

Változó módosítás

A használt változók értékét könnyedén módosíthatjuk a felhasználásuk előtt. Például nagybetűssé konvertálás, csonkítás, szóközök szűrése stb.

Sablon függvények

A sablon függvények segítségével összetettebb lehetőségeket is kap a dizájner az egyszerű megjelenítés helyett.

Szűrők

A programozó teljesen kontroll alatt tudja tartani a kimenetet. Ezt elő- vagy utószűrők segítségével teheti meg. Pl. a JavaScript fejezetben bemutatott e-mail cím elrejtés csak akkor működőképes, ha a @ karakter nem jelenik meg a kimenetben. Egy utószűrővel meg lehet oldani, hogy az esetleg a szövegben levő @ karaktereket automatikusan – a korábbiakban leírtaknak megfelelően – elkódolja.

Kiegészítések

Az alaprendszerhez sokféle kiegészítő tölthető le a webről, és alkalmazható, vagy akár magunk is fejleszthetünk kiegészítőt.

Munkafolyamat Smartyval

A dizájner és a programozó sablonrendszer használatával nagyjából a következő módon dolgozhatnak a követelményspecifikáció elkészítése után:

- A dizájner minden egyes oldaltípushoz készít egy tisztán HTML mintát.
- A programozó PHP-ben megvalósítja az üzleti logikát (alkalmazás logikát).
- Kettőjük kommunikációjából előáll a dizájn és az üzleti logika csatolófelülete (interfésze), ami a változók formájában megjelenő adatok és a dizájn-elemek kapcsolatát fejezi ki.
- Ez után a kisebb (vagyis az interfészt nem befolyásoló) változások keretén belül a programozó és a dizájner ismét egymástól függetlenül végezheti a feladatát.

10.1.1 Smarty alapok

A Smarty fő célja az üzleti logika és a prezentációs logika szétválasztása.

Nézzünk példaként egy cikkeket publikáló honlapot. Az egyes cikkeket megjelenítő oldalak szerkezete hasonló lesz. Lesz az oldalnak fejléce, lábléce, címe, írója, megjelenési dátuma, tartalma stb. Érdemes megfigyelni, hogy az előző példa logikailag fogalmazta meg az oldalt alkotó elemeket. Nem beszélt *table* és *p* elemekről, se a megjelenítés vizuális, esztétikai részéről. Ráér majd a dizájner azzal foglalkozni, hogy ez HTML (vagy akár WAP) kód szintjén hogyan fog megvalósulni.

A Smarty a sablon fájl alapján egy PHP állományt fog előállítani. Mivel ez költséges művelet, érdemes gyorstárazni az eredményt, hogy a későbbi kérések gyorsabban kiszolgálhatók legyenek.

Telepítés

A Smarty futtatásához PHP 4-es változata is elegendő.

A Smarty magja

Ha letöltjük a Smarty legfrissebb változatát a letöltési oldalról[75], akkor egy tömörített állományt találunk. Ennek *libs* alkönyvtára tartalmazza a rendszer magját. Ennek a könyvtárnak a rendszerünkbe másolása jelenti a telepítés első lépését. A könyvtár tartalmát soha ne módosítsuk!

A Smarty *lib* könyvtárában található állományait az alkalmazásunkból el kell tudni érnünk. Éppen ezért vagy egy globálisan elérhető helyre, vagy pl. a portál gyökérmappájába célszerű helyeznünk. Fontos szerepet tölt még be a *SMARTY_DIR* változó is. Szokás szerint ez a változó tartalmazza a Smarty telepítés helyét.

Nézzük meg a lehető legegyszerűbb példát:

```
<?
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

A Smarty használatáról csak akkor beszélhetünk, ha létrehozunk egy példányt a Smarty osztályból. Az osztály kódja pedig (a beszédes nevű) *Smarty.class.php* nevű állományban található.

A *SMARTY_DIR* változó használatával a példánk így néz ki:

```
<?
define('SMARTY_DIR', 'c:/webroot/libs/Smarty-v.e.r/libs/');
require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```

További könyvtárak

A Smarty számára további négy könyvtárra lesz még szükség:

```
templates/
templates_c/
configs/
cache/
```

Ezek a könyvtárak tipikusan a weboldalunk gyökérmappájába kerülnek. A *templates* és *configs* könyvtárak állományait tipikusan a dizájner hozza létre: sablonok és konfigurációs állományok kerülnek beléjük.

A webszervert futtató felhasználónak írási joggal kell rendelkezni a *templates_*c és *cache* könyvtárakra, ugyanis ezekben tárolja a sablon állományokból fordított PHP állományokat, valamint a gyorstárazott HTML oldalainkat.

Megjegyzés: A Smarty belsőleg kétszintű átmeneti tárolást tartalmaz. Az első szintet az jelenti, hogy amikor először tekintünk meg egy sablont, a Smarty tiszta PHP kódra fordítja azt, és menti az eredményt. Ez a tárolási lépés megakadályozza, hogy a sablonkódokat az első kérelem után is minden alkalommal fel kelljen dolgozni. A második szint, hogy a Smarty az éppen megjelenített (jellemzően HTML) tartalom átmeneti tárolását is megengedi.

Hello Világ!

Eljutottunk az első teljes, működő alkalmazásunkhoz. Nézzük először a sablon állományt (*templates/index.tpl*):

[75] http://smarty.php.net/download.php

```
{* Smarty *}

Hello {$name}, welcome to Smarty!
```

A sablont megjelenítő kód:

```
<?php
  require_once(SMARTY_DIR . 'Smarty.class.php');
  $smarty = new Smarty();
  $smarty->assign('name','Ned');
  $smarty->display('index.tpl');
?>
```

A példán már látszik is a sablonelvű fejlesztés legalapvetőbb eleme: a sablon oldal *{$name}* változóját szeretnénk kicserélni egy aktuális névre. Erre a *smarty* objektum *assign* metódusánál láthatunk egy hozzárendelést: az objektum tulajdonképpen gyűjti az ilyen típusú hozzárendeléseket. A sablon állomány feldolgozása és a sablonváltozók beillesztgetése történik meg a display metódus segítségével.

10.1.2 Változó módosítók

Nemcsak kiírathatjuk a változók tartalmát, hanem meg is lehet változtatni a kiíratás előtt azokat. Az érthetőség kedvéért mindegyik módosítót egy rövid példával ismerhetjük meg. A teljesség igénye nélkül néhány módosító:

Megjegyzés: a fejezet következő példái sorrendben a PHP állományt, a sablon állományt és az eredményt mutatják.

capitalize

Ez a módosító minden szó első betűjét átteszi nagybetűvé.

```
$smarty->assign('valtozo', 'következő film, próbálkozás, x3');
```

```
{$valtozo|capitalize}
```

```
Következő Film, Próbálkozás, x3
```

count_characters

Egy adott változó változóban megszámolja a karakterek számát.

```
$smarty->assign('valtozo', 'Tegnap nagyon hideg volt.');
```

```
{$valtozo|count_characters}
```

```
22
```

date_format

Különböző dátum és idő formátumok.

```
$smarty->assign('tegnap', strtotime('-1 day'));
```

```
{$smarty.now|date_format}
{$smarty.now|date_format:"%I:%M %p"}
```

```
Mar 21, 2006
03/21/06
```

default

Ha a használt változó üres vagy olyan változóra hivatkozunk ami nem létezik, akkor ennek a tartalma lesz felhasználva.

```
$smarty->assign('email', '');
{$email|default:'No email address available'}
No email address available
```

escape

A szöveget különböző kódolásnak megfelelően lehet konvertálni.

```
$smarty->assign('valtozo', "'Ez lesz a következő nagy terv!'");
{$valtozo|escape:"url"}
%27Ez%20lesz%20a%20k%F6vetkez%F5%20nagy%20terv%21%27
```

lower

Minden karaktert kisbetűsé alakít.

```
$smarty->assign('valtozo', "'Ez Lesz A Következő NAgY Terv!'");
{$valtozo|lower}
'ez lesz a következő nagy terv!'
```

nl2br

A „\n”-et átalakítja a HTML nyelvben használatos
 elemre.

```
$smarty->assign(
    'valtozo', "'Ez lesz a következő\n nagy terv!'");
{$valtozo|nl2br}
'Ez lesz a következő<br /> nagy terv!'
```

replace

A megadott a stringet megkeresi és kicseréli.

```
$smarty->assign('valtozo', "Ez lesz a következő nagy terv!");
{$valtozo|replace:'nagy':'kis'}
Ez lesz a következő kis terv!
```

***strip_*tags**

Az formázó tagokat kicseréli helyközre vagy teljesen eltünteti őket.

```
$smarty->assign('valtozo', "Ez lesz a következő <font
face=\"helvetica\">nagy terv</font>");
{$valtozo|strip_tags}
Ez lesz a következő nagy terv, amelyet megvalósítok az
elkövetkező évben.
```

10.1.3 Vezérlési szerkezetek

Az egyszerű változóbehelyettesítés használata a Smarty-t látszatra rendkívül erőteljessé teszi. A sablonok világosak és egyszerűek, és a háttérkód sem bonyolult.

Elágazások

Nézzünk egy szituációt: Ha webhelyünk látogatójától csak az első alkalommal akarunk bejelentkezést kérni, a későbbiekben (szütitől függően) már nem, akkor két lehetőségünk van. Az első, hogy ezt a PHP kódban végezzük el, az alábbihoz hasonló módon:

```
require('../smarty/Smarty.class.php');
$smarty = new Smarty;

$name = array_key_exists('name', $_COOKIE) ? $_COOKIE['name'] :
'Stranger';

if($name == 'Stranger') {
  $login_link = "<a href=\"/login.php\">belépés</a>";
} else {
  $login_link = ' ';
}

$smarty->assign('name', $name);
$smarty->assign('login_link', $login_link);
$smarty->display('hello2.tpl');
```

Ezután a sablonnal meg kell jelentetnünk a *$login_link*-et, ami lehet, hogy be sincs állítva:

```
<html>
<body>
Helló {$name}.<br>
{php}{$login_link;}{/php}
</body>
</html>
```

Ez a módszer sajnos megsérti az alkalmazási és megjelenítési kód szétválasztásának elvét.

A második megoldás, hogy a megjelenítési rétegre bízzuk a döntést, hogy megjeleníti-e a bejelentkezési hivatkozást, és ha igen, hogyan:

A *hello3.tpl* tartalma:

```
<html>
<body>
Hello {$name}.<br>

{ if $name == "Stranger" }
  <a href="/login.php">Belépés</a>
{ /if }

</body>
</html>
```

A *hello3.php* tartalma:

```
require('../smarty/Smarty.class.php');
$smarty = new Smarty;

$name = array_key_exists ( 'name' ,
    $_COOKIE) ? $_COOKIE ['name' ] : 'Stranger';

$smarty->assign('name',$name);
$smarty->display('hello3.tpl');
```

Ciklusok

A teljes feltételrendszert nyújtó *if-elseif-else* utasításokon kívül a Smarty a *foreach*-en keresztül a tömbök ciklussal való feldolgozását is támogatja. Egy egyszerű sablon, amely kiírja az éppen érvényes környezeti változókat. A *getenv.tpl* tartalma:

```
<html>
<body>
<table>

{foreach from=$smarty.env key=key item=value }
  <trxtd>{$key}</tdxtd>{$value}</tdx/tr>
{/foreach}

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

A *getenv.php* tartalma:

```
require('../smarty/Smarty.class.php');
$smarty = new Smarty;
$smarty->display('getenv.tpl');
```

A fenti példában láthatjuk a *$smarty* változót is, ami egy asszociatív tömb, melynek segítségével elérhetjük a PHP szuperglobálisait (például *$_GET*), illetve a Smarty beállítási változóit. A szuperglobálisok elérése a *$smarty.cookie*, *$smarty.get* stb. formában történhet.

Hol legyen a logika?

Egyesek ez ellen azzal érvelhetnek, hogy magában a sablonban egyáltalán nem ajánlott működési logikát elhelyezni. Az nem biztos, hogy jó, ha a megjelenítésből teljesen kivonjuk a logikát, vagy azt jelenti, hogy a kimenet előállításához valójában nem tartozik logika. A megjelenítési kódnak az alkalmazásba helyezése pedig nem jobb, mint az alkalmazáslogika beépítése a megjelenítési kódba. A sablonrendszerek használatának éppen az a célja, hogy mindkét említett helyzetet elkerüljük.

Mindazonáltal a sablonokban logikát elhelyezni sok veszélyt rejt. Minél több szolgáltatás érhető el a sablonokban, annál nagyobb a kísértés, hogy maga az oldal tartalmazzon nagy mennyiségű kódot. Amíg ez a megjelenítésre korlátozódik, tartjuk magunkat az MVC mintához.

Az MVC nem arról szól, hogy a nézetből eltávolítunk minden logikát - a cél az, hogy az területre jellemző működési kódot vegyük ki belőle. A megjelenítési és működési kód között azonban nem minden esetben könnyű különbséget tenni. Számos fejlesztő számára nem csupán az a cél, hogy elválasszák a megjelenítést az alkalmazástól, hanem az, hogy a megjelenítési kód is minél kisebb legyen. A Smarty ezt a problémát nem oldja meg.

Függvények

Az alapvető vezérlési szerkezetek mellett a Smarty lehetőséget ad arra is, hogy beépített, illetve felhasználói függvényeket hívjunk meg. Ez nagyobb rugalmasságot ad abban, hogy mit tehetünk meg magán a sablonkódon belül, de az az ára, hogy a sablonok bonyolulttá válnak.

include

A leghasznosabb beépített függvény az *include*. A PHP *include* függvényével azonos módon azt teszi lehetővé, hogy egy sablonba egy másikat építsünk be. De ez csak egy a sok tucat esetből, amelyre használhatjuk. Például közös fejlécet és láblécet adjunk a sablonokhoz, mint az alábbi egyszerű példában:

A *fejlec.tpl* tartalma:

```
<html>
  <head>
  <title>{$title}</title>

{if $css}
  <link rel="stylesheet" type="text/css" href="{$css}" />
{/if}

</head>
<body>
<br>
```

A *lablec.tpl* tartalma:

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

Ha ezután egy sablonban fejlécre vagy láblécre van szükség, a következőképpen építjük be azokat. Az *include.tpl* tartalma:

```
{include file="fejlec.tpl"}

Hello {$name}.

{include file="lablec.tpl"}
```

php

A Smarty támogatja a *php* függvényt is, melynek segítségével sablonon belüli PHP-kódot írhatunk. Az *include2.tpl* tartalma:

```
{include file="fejlec.tpl"}

Helló {php}print $_GET['name'];{/php}

{include file="lablec.tpl"}
```

A programozási nyelvek keverése egyetlen dokumentumon belül szinte soha nem jó ötlet. Feleslegesen bonyolítja az alkalmazást, és megnehezíti, hogy megállapítsuk, hol található egy adott szolgáltatás megvalósítása. A Smarty támogatja az egyéni függvényeket és változómódosítókat is. Az egyéni függvények az összetett feladatokat automatizáló segéd-függvények készítésében lehetnek segítségünkre.

Külső függvények

A külső függvényeket nem a Smarty fejlesztői készítették, de a telepítőcsomag is tartalmaz néhányat. Ismét a teljesség igénye nélkül következzék néhány függvény, amelyik űrlatok tömbből való generálását könnyítik meg:

Html_checkboxes

Jelölőnégyzetetek állíthatunk elő vele. Egy tömbben meg kell adni az értékét, és a szerint létrehozza őket.

```
$smarty->assign('cust_ids', array(1000,1001,1002,1003));
$smarty->assign('cust_names',
    array('Joe Schmoe','Jack Smith','Jane Johnson',
          'Charlie Brown'));
$smarty->assign('customer_id', 1001);

{html_checkboxes name="id" values=$cust_ids
    output=$cust_names selected=$customer_id separator="<br />"}
```

A kimenet:

```
<label><input type="checkbox" name="id[]" value="1000" />Joe
Schmoe</label><br />

<label><input type="checkbox" name="id[]" value="1001"
checked="checked" />Jack Smith</label><br />

<label><input type="checkbox" name="id[]" value="1002" />Jane
Johnson</label><br />

<label><input type="checkbox" name="id[]" value="1003" />Charlie
Brown</label><br />
```

10.1.4 Gyorstárazás

A lefordított sablonok használatánál is gyorsabb a sablonok kimenetének átmeneti tárolása, így a sablont egyáltalán nem kell végrehajtani. Az átmeneti tárolás (gyorstárazás, caching) általában véve is igen hatékony módszer.

A Smarty-ban a tartalom átmeneti tárolásához először engedélyeznünk kell a tárolást az objektumban, mégpedig a következő sorral:

```
$smarty->cache = true;
```

Ezután amikor csak meghívjuk a *display* függvényt, az oldal teljes kimenete

```
$smarty->cache_lifetime
```

ideig (ez általában 3600 másodperc) tárolódik.

Megjegyzés: Még pontosabban fogalmazva az ezen az időintervallumon belül érkező újabb kérések az eltárolt kimenetet fogják kapni.

A legtöbb oldalon a beágyazott PHP program igényli a legtöbb időt, hiszen az adatokból elő kell állítani az oldalt. A folyamatot rövidre zárhatjuk, ha az *is_cached* függvénnyel ellenőrizzük, hogy az oldalnak létezik-e tárolt másolata. A PHP programon belül ez a következőképpen történik:

```
$smarty = new Smarty;
if(!is_cached('index.tpl')) {

  /* beállítás */
}
$smarty->display('index.tpl');
```

Ha az oldal bármilyen felhasználóhoz köthető információt tartalmaz, ez a módszer számunkra nem megfelelő, mert csak az első felhasználóra vonatkozó információk tárolódnak, és minden további felhasználó ugyanazt kapja.

Ha feltételekkel szeretnénk tárolni az adatokat, a *display* -nek egy második paramétert kell átadnunk. Ezt a tárolási rendszer kulcsként használja, hogy azonos kulccsal rendelkező kérelem esetén visszaadhassa a tárolt tartalmat. Ha a *sajatoldal.tpl* sablont például

10 percig minden kérelmező számára egyedileg szeretnénk tárolni, a felhasználókat a nevük alapján előállított MD5 kivonat alapján azonosíthatjuk:

```
$smarty = new Smarty;
if(!is_cached('sajatoldal.tpl', md5($_C00KIE['name'])) ) {
  /* beállítás */
  $smarty->assign('name', $_C00KIE['name']);
}$
smarty->display('sajatoldal.tpl', md5($_COOKIE['name' ] )) ;
```

Láthatjuk, hogy az *is_cached* továbbra is használható, csak át kell neki adni az azonosítót is.

Legyünk óvatosak: a Smarty nem rendelkezik beépített szemétgyűjtéssel, így minden tárolt oldal egy fájlt jelent a tároló fájlrendszerben. Ez lehetőséget teremt a véletlen vagy szándékos túlterhelésre, amikor is tárolt oldalak ezrei gyűlnek fel a rendszerben. Ehelyett azt javasoljuk, hogy viszonylag kevés értéket felvehető kulcs alapján válasszuk ki a tárolandó tartalmat.

További források

http://smarty.php.net/crashcourse.php

http://smarty.php.net/manual/en/

http://smarty.php.net/sampleapp/sampleapp_p1.php

10.2. A PHP, mint sablonnyelv

Vitathatatlan, hogy a sablonrendszerek nagyon fontos szereket töltenek be a webfejlesztésben. A Smarty mellett sok kisebb rendszer is létezik, amelyek a Smarty bonyolultsága, vagy éppen egy más típusú sablonnyelv igénye miatt más utakon járnak. Vannak azonban olyan fejlesztők is – köztük a szerző is -, akik szerint olyan esetben, amikor a dizájner ismeri a PHP nyelvet, felesleges lehet egy külön sablonnyelvet megtanulni és alkalmazni.

Ezzel nem a sablon logika különválasztásának, hanem csupán a külön sablon nyelv szükségességét kérdőjelezzük meg. A tanulás szempontjából tehát igenis nagyon hasznos, ha megismerjük pl. a Smartyt, de ezután akár egy sablonnyelv, akár a PHP is alkalmas lehet a feladat megoldására, ha a fejlesztők fegyelmezetten külön tudják választani a megjelenítési és alkalmazási kódot.

Érdemes itt megjegyezni, hogy pl. a Drupal és a Wordpress tartalomkezelő rendszerek estén is lehetőség van PHP alapú sablon (vagy más néven smink) kezelésre. Van azonban arra is példa, hogy tartalomkezelő rendszer a Smartyt használja beépített módon.

10.2.1 Sablon osztály

Ha PHP alapokon akarunk sablon-elvű módon programozni, akkor nagyon hasznos lehet számunkra a következő *Sablon* osztály. (Ez az osztály massassi[76] megoldása alapján egy kissé testreszabott megoldás.)

```
<?
class Sablon {
  var $dir = 'sablon/';
```

[76] http://www.massassi.com/php/articles/template_engines/

A sablon állományok helye a fájlrendszerben.

```
  var $fajl;
  var $cserek = array();
```

A sablon fájl neve és a cserélendő párok.

```
  function Sablon ($f) {
    $this -> fajl = $f;
  }
```

A konstruktor mindössze a fájl nevét tárolja el.

```
  function berak($nev, $ertek) {
    if (is_object($ertek)) {
      $this -> cserek[$nev] = $ertek->cserel();
    } else {
      $this -> cserek[$nev] = $ertek;
    }
  }
```

A leggyakrabban használt metódus, hiszen ezzel definiálhatók a cserélendő párok.

```
  function berakTomb($tomb) {
    if (is_array($tomb)) {
      $this -> cserek = ($this -> cserek) + $tomb;
    }
  }
```

Hasznos függvény, ha a cserélendő párok már eleve tömbben helyezkednek el.

```
  function cserel() {
    extract($this -> cserek);
    $cimsor;
    ob_start();
    include($this -> dir . $this -> fajl . '.tpl.php');
    $tartalom = ob_get_contents();
    ob_end_clean();
    return $tartalom;
  }
```

A tényleges sablon műveletet, a cserék megvalósítását végzi. Érdemes megfigyelni, mennyire egyszerűvé teszi a kódot a kimenet pufferelése (*ob_* függvények).

```
  function torol() {
    $this -> cserek = array();
  }
}
?>
```

Végül a töröl függvény lehetővé teszi, hogy újabb sablon objektum példányosítása nélkül is újra használhassuk a sablonállományunkat, pl. több azonos sablon alapján gyártott blokk generálására.

10.2.2 Blogbejegyzés

Nézzünk egy konkrét példát a nagygusztav.hu oldal 2006-os kódjából. A következő sablon állomány (*ujBejegyzesekBlokk.tpl.php*) egy blogbejegyzés előzetest fog készíteni a kezdőoldalra.

```
<? /* Csak az előzetes megjelenítésére, hozzászólások számával
*/ ?>
<div class="hir">
<? if ($lehethozza || $tartalom) { ?>
  <h2><a href="<?= $index ?>?<?= $blognev ?>/<?=
        $url ?>"><?= $hirCim ?></a></h2>
<? } else { ?>
  <h2><?= $hirCim ?></h2>
<? }?>

<? if ($datum) {?>
  <acronym class="datum" title="<?= $datum['hosszu'] ?>">
    <span class="honap"><?= $datum['honap'] ?></span>
    <span class="nap"><?= $datum['nap'] ?></span>
  </acronym>
<? } ?>

<?= $hirSzoveg ?>

<? if ($lehethozza || $tartalom) { ?>
<p class="hozzaszolasszam">
<? if ($tartalom) { ?>
  <a href="<?= $index ?>?<?= $blognev ?>/<?= $url ?>">
         Teljes bejegyzés&raquo;</a>
<? }?>

<? if ($lehethozza) { ?>
Eddig
<?  if ($hozzaszolasSzam == 0) { ?>
 nincs
<?  } else {?>
  <?= $hozzaszolasSzam ?>
<?  } ?>
<a href="<?= $index ?>?<?= $blognev ?>/<?=
    $url ?>#hozza">hozzászólás</a>.
<? }?>
</p>
<?  } ?>
</div>
```

A következő kód használja a sablont:

```
$ujBejegyzesek = $this -> adatbazis -> ujBejegyzesek(10);
foreach ($ujBejegyzesek as $k => $e) {
  $ujBejegyzesek[$k]['blogcim'] = $this ->
      joOldalak[$e['blognev']]['cim'];
}

$sablon = new Sablon('ujBejegyzesekBlokk');
$sablon -> berakTomb($GLOBALS['portal_adatok']);
$sablon -> berak('ujBejegyzesek', $ujBejegyzesek);

$szoveg .= $sablon -> cserel();
```

Források

- George S. Schlossnagle: PHP fejlesztés felsőfokon (http://www.kiskapu.hu/)
- deadlime: PHP, mint sablonnyelv (http://deadlime.hu/2006/07/28/php-mint-sablonnyelv/)

- phpPatterns: PHP and Templates (http://www.phppatterns.com/docs/design/templates_and_template_engines)
- massassi: Template Engines (http://www.massassi.com/php/articles/template_engines/)