

Nagy Gusztáv
Web programozás

0.5. verzió

2007. május

Jogi nyilatkozat



Nevezd meg! - Ne add el! 2.5 Magyarország

A következőket teheted a művel:

- szabadon másolhatod, terjesztheted, bemutathatod és előadhatod a művet
- származékos műveket (feldolgozásokat) hozhatsz létre

Az alábbi feltételekkel:



Jelöld meg! A szerző vagy a jogosult által meghatározott módon kell megjelölni a művet (pl. a szerző és a cím feltüntetésével).



Ne add el! Ezt a művet nem használhatod fel kereskedelmi célokra.

A szerzői jogok tulajdonosának írásos engedélyével bármelyik fenti feltételtől eltérhetsz. A fentiek nem befolyásolják a szabad felhasználáshoz fűződő, illetve az egyéb jogokat.

Ez a *Legal Code* (Jogi változat, vagyis a teljes licenc) szövegének közérthető nyelven megfogalmazott kivonata.

Ez a kivonat a <http://creativecommons.org/licenses/by-nc/2.5/hu/> oldalon is olvasható. A teljes licenz a <http://creativecommons.org/licenses/by-nc/2.5/hu/legalcode> oldalon érhető el.

E jegyzet hivatalos honlapjáról (<http://nagyusztav.hu>) tölthető le a mindenkori legfrissebb verzió.

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 mellett további segítséget ad a Web kliens és szerver oldali nyelvei, technológiai 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 megoldását javaslom. 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.

Verzió

A jelenlegi 0.5-ös verzió a félév során folyamatosan készült. A jegyzet második felében sajnos még sok téma csak vázlatosan, hiányosan szerepel. Ezek finomítása, kiegészítése – az ellenőrző kérdésekkel és gyakorló feladatokkal együtt - a következő félévben fog folytatódni.

Köszönet

A jegyzet elkészítéséhez elsősorban a W3Schools¹ oktatóanyagait használtam fel. Az anyagok fordításában résztvevő hallgatókat szintén köszönet illeti.

Végül szeretnék köszönetet mondani a Weblabor² szakmai közösségének, akiktől a legtöbbet tanultam – többek között – szemléletmódban.

Kecskemét, 2007. május

a szerző

¹ <http://www.w3schools.com/>

² <http://weblabor.hu/>

Tartalomjegyzék

1. Fejlesztőkörnyezet kialakítása.....	8
1.1. Szerver operációs rendszer.....	8
1.2. Szerveralkalmazások.....	9
1.2.1 XAMPP integrált telepítő csomag.....	9
1.2.2 Az Apache telepítése.....	11
1.2.3 A PHP telepítése.....	13
1.3. Fejlesztőeszközök.....	14
1.4. Grafikus programok.....	14
2. A webfejlesztés alapjai.....	16
2.1. Tervezési, fejlesztési szempontok.....	16
2.1.1 Honlaptervezés.....	16
2.1.2 Navigációs struktúra.....	16
2.1.3 Oldaltervezés.....	18
2.1.4 Figyeljünk a látogatók visszajelzéseire.....	21
2.1.5 Milyen monitort használnak a látogatók?.....	21
2.1.6 Milyen böngészőt használnak a látogatók?.....	21
2.1.7 Mi van a sérültekkel?.....	22
2.2. A web szabványai.....	22
2.2.1 A World Wide Web Consortium (W3C).....	22
2.2.2 Validátorok.....	23
2.3. Ellenőrző kérdések.....	23
3. HTML.....	24
3.1. Bevezetés.....	24
3.2. Tagok.....	26
3.3. Alapvető HTML tagok.....	27
3.4. Hogy nézzük meg egy oldal HTML kódját?.....	31
3.5. Formázás.....	33
3.6. Karakter entitások.....	36
3.7. Linkek.....	37
3.8. Keretek.....	38
3.9. Táblázatok.....	41
3.10. Listák.....	44
3.11. Űrlapok.....	46
3.12. Képek.....	50
3.13. Oldal háttere.....	53
3.14. Színek.....	53
3.15. Szabványosság.....	53
3.16. Stílusok.....	54
3.17. Fejrész.....	55
3.18. Szkriptek.....	56
3.19. Általános tulajdonságok.....	57
3.20. Esemény tulajdonságok.....	58
3.21. URL-kódolás.....	60
3.22. Szemantikus HTML.....	60
3.23. Validátor.....	60
3.24. Ellenőrző kérdések.....	60
3.25. További források.....	64
4. XHTML.....	65

4.1. XML alapok.....	65
4.1.1 Mire használjuk az XML-t?.....	65
4.1.2 XML szintaxis.....	66
4.2. XHTML a gyakorlathoz képest.....	67
4.3. Dokumentumtípus-deklaráció.....	69
4.3.1 XHTML 1.0 DTD-k.....	69
4.3.2 XHTML 1.1 DTD.....	69
4.4. Visszafelé kompatibilitás.....	70
4.5. További források.....	70
5. CSS.....	71
5.1. Bevezetés.....	71
5.2. A CSS nyelvtana.....	73
5.3. A background tulajdonságok.....	75
5.3.1 Háttérszín.....	75
5.3.2 Háttérkép.....	76
5.4. Szövegek megjelenítése.....	77
5.5. Betűk formázása.....	78
5.6. Szegélyek.....	79
5.7. Térközök a szegélyen belül és kívül.....	81
5.8. Listák.....	81
5.9. Méretek.....	83
5.10. Megjelenítés.....	83
5.10.1 A lebegtetés.....	84
5.10.2 Pozícionálási sémák.....	88
5.10.3 Láthatóság.....	89
5.10.4 Z-index.....	90
5.11. Látszólagos osztályok.....	90
5.11.1 Linkek viselkedése.....	90
5.11.2 Első gyermek.....	90
5.11.3 Első betű és első sor.....	91
5.12. Média típusok.....	91
5.13. Ellenőrző kérdések.....	92
5.14. Validátor.....	94
5.15. További források.....	95
6. JavaScript.....	96
6.1. Bevezetés a JavaScript nyelvbe.....	96
6.1.1 Változók.....	97
6.1.2 Elágazások.....	98
6.1.3 Operátorok.....	99
6.1.4 Dialógusablakok.....	102
6.1.5 Függvények.....	103
6.1.6 Ciklusok.....	104
6.1.7 Eseménykezelés.....	106
6.1.8 Kivételkezelés.....	108
6.2. Objektorientált programozás.....	109
6.2.1 Fontosabb objektumok röviden.....	110
6.3. HTML DOM.....	112
6.3.1 getElementById.....	112
6.3.2 A Document objektum.....	113
6.3.3 Az Event objektum.....	113
6.4. Diszkrét JavaScript.....	114
6.4.1 Előugró ablak példa.....	114

6.4.2 E-mail cím elrejtése.....	116
6.5. Gyakran használt függvények.....	117
6.5.1 Sütik kezelése.....	117
6.5.2 getElementByClass.....	119
6.5.3 addEvent.....	119
6.5.4 addLoadEvent.....	120
6.6. Kódtárak.....	120
6.7. Alkalmazások.....	121
6.8. Felhasználói élmény.....	122
6.8.1 Kliens oldali űrlap ellenőrzés.....	122
6.8.2 Hosszú listák böngészése helyett.....	124
6.9. Elavult technikák.....	128
6.10. További források.....	128
7. AJAX.....	129
7.1. Bevezető példa.....	130
7.2. A böngészők AJAX támogatása.....	131
7.3. XML-kommunikáció.....	132
7.4. További források.....	132
8. PHP nyelv és környezet.....	133
8.1. Alapok.....	133
8.1.1 Szintaxis.....	133
8.1.2 Változók.....	135
8.1.3 Operátorok.....	135
8.1.4 Elágazások.....	137
8.1.5 A switch szerkezet.....	139
8.1.6 Tömbök.....	140
8.1.7 Ciklusok.....	143
8.1.8 Függvények.....	145
8.1.9 Űrlapok és felhasználói adatbevitel.....	148
8.1.10 A \$_GET tömb.....	149
8.1.11 A \$_POST tömb.....	150
8.2. Haladó témák.....	151
8.2.1 Dátumok kezelése.....	151
8.2.2 Az include és társai.....	152
8.2.3 Fájlkezelés.....	154
8.2.4 Fájl feltöltése.....	156
8.2.5 Sütik kezelése.....	158
8.2.6 Munkamenet-kezelés.....	160
8.2.7 Levélküldés.....	161
8.2.8 PHP hibakezelés.....	164
8.2.9 Kivételkezelés.....	169
8.3. Adatbázis-kapcsolat kezelése PHP-ben.....	174
8.3.1 MySQL alapok.....	174
8.3.2 Kapcsolódás egy MySQL adatbázishoz	175
8.3.3 Adatbázisok és táblák létrehozása.....	176
8.3.4 Adatok bevitele adatbázisba.....	179
8.3.5 Lekérdezés.....	180
8.3.6 A WHERE záradék.....	182
8.3.7 Az ORDER BY kulcsszó.....	183
8.3.8 Adatok módosítása.....	183
8.3.9 Adatok törlése az adatbázisból.....	184
8.3.10 ODBC kapcsolat létesítése.....	185

8.4. XML kezelés.....	187
8.4.1 Expat XML elemező.....	187
8.4.2 XML DOM.....	187
8.4.3 SimpleXML.....	189
9. PHP megoldások.....	191
9.1. Egy portál váza.....	191
9.1.1 GAMF honlap.....	191
9.1.2 Egy másik megközelítés.....	196
9.1.3 MVC megközelítés.....	198
9.2. Biblia.....	203
9.3. Beléptető-rendszer.....	208
9.4. Tervezési minták.....	214
9.4.1 Front Controller.....	214
9.4.2 Strategy.....	215
9.4.3 Data Access Object.....	217
9.4.4 MVC.....	220
10. Sablonrendszerek.....	222
10.1. Smarty.....	222
10.1.1 Smarty alapok.....	223
10.1.2 Változó módosítók.....	225
10.1.3 Vezérlési szerkezetek.....	226
10.1.4 Gyorstárazás.....	230
10.2. A PHP, mint sablonnyelv.....	231
10.2.1 Sablon osztály.....	231
10.2.2 Blogbejegyzés.....	233
11. Keretrendszerek.....	235
11.1. Code Igniter.....	235
11.1.1 Telepítés.....	236
11.1.2 Az URL-ek felépítése.....	236
11.2. CakePHP.....	236
11.2.1 Alapkonceptió.....	236
11.2.2 A CakePHP telepítése.....	238
11.2.3 Konfiguráció.....	239
11.2.4 GyorsVáz azaz Scaffolding képességek.....	242
11.2.5 Nézetek.....	244
11.2.6 Komponensek.....	246
12. További szerver platformok.....	249
12.1. J2EE.....	249
12.2. Microsoft .NET.....	249
12.2.1 A .NET Keretrendszer.....	249
12.2.2 A .NET építőkövei.....	250
12.2.3 .NET Szoftver.....	251
12.2.4 SQL Server 2000.....	252
12.3. Python.....	253
12.4. Ruby on Rails.....	253
13. Tartalomkezelő rendszerek.....	254
14. Álláskereső próbafeladatok.....	255
14.1. Egyszerű beléptető-rendszer.....	255
14.2. Dinamikus űrlapgeneráló osztály.....	255
14.3. Címjegyzék alkalmazás.....	257
14.4. Álláshirdetések.....	257

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

Linux³

Az elterjedtebb, és nem kimondottan asztali (*desktop*) használatra szánt Linux 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 szolgáltatás⁴ igénybe vételével is többnyire Linux alapú szerverekkel találkozhatunk.

BSD⁵

Kevésbé közismert, de egyes rendszergazdák körében stabilitása miatt nagy népszerűségnek örvendő Unix 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 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 nevében nagybetű soha ne szerepeljen a fájlnevében.

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

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. ``.

³ <http://www.linux.hu/>

⁴ <http://tarhely.lap.hu/>

⁵ <http://www.bsd.hu/>

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 beállításába belelátni.

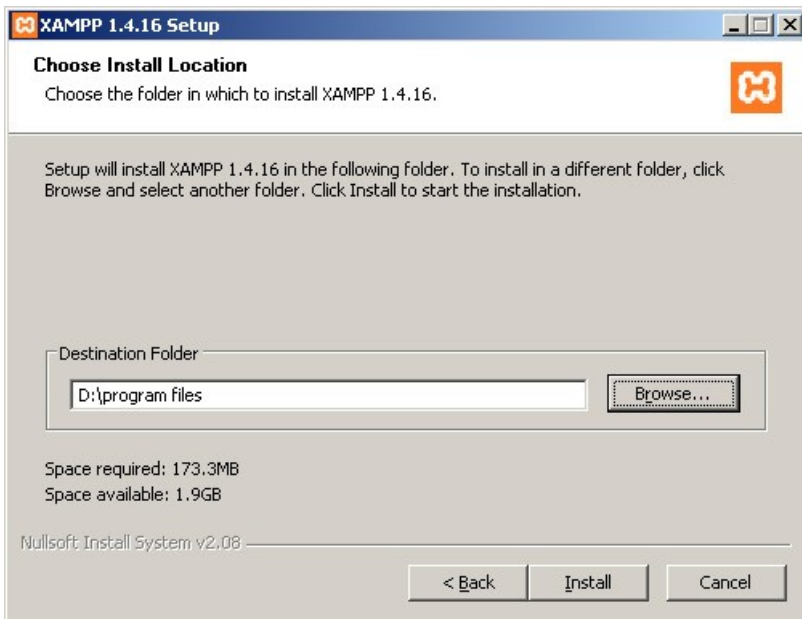
A szerző által leginkább ajánlott csomag az **XAMPP**⁶. Az XAMP for Windows 1.5.3a változata a következő szoftvereket telepíti és konfigurálja:

- Apache 2.2.2
- MySQL 5.0.21
- PHP 5.1.4 + PHP 4.4.2-pl1 + PEAR
- PHP-Switch win32 1.0
- XAMPP Control Version 2.3
- XAMPP Security 1.0
- SQLite 2.8.15
- OpenSSL 0.9.8b
- phpMyAdmin 2.8.1
- ADOdb 4.80
- Mercury Mail Transport System v4.01b
- FileZilla FTP Server 0.9.16c
- Webalizer 2.01-10
- Zend Optimizer 3.0.0

Ha valakinek ez az összeállítás túl sok, akkor a Lite változatot is alkalmazhatjuk.

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

⁶ <http://www.apachefriends.org/>

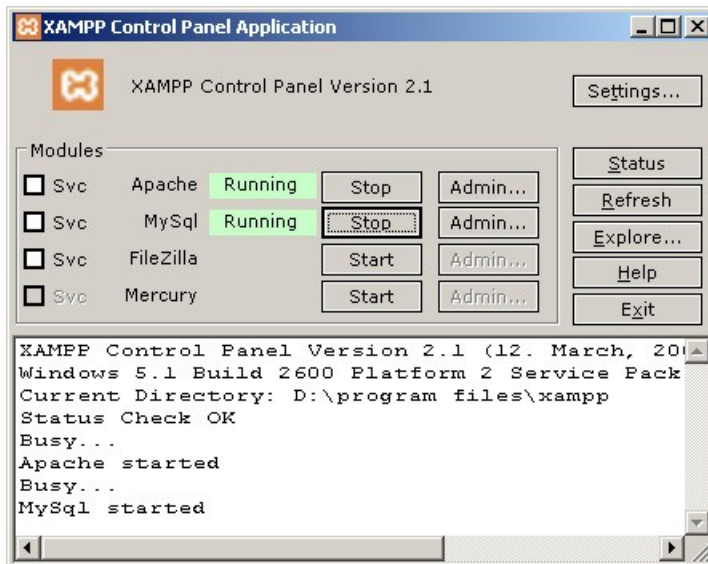


A telepítés végezhető parancssorból is a *setup-xampp.bat* parancsállomány futtatásával:

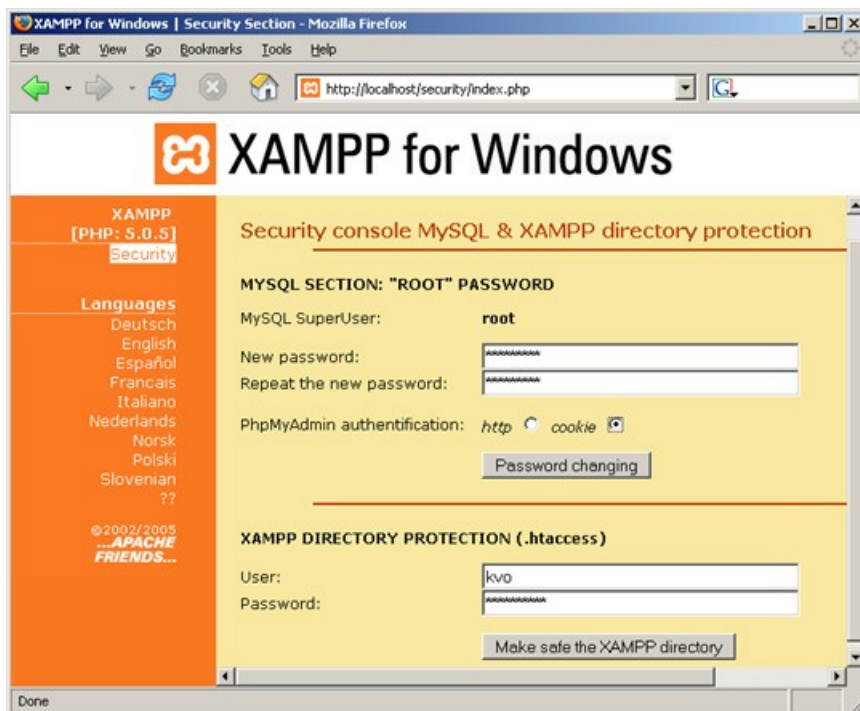
```
C:\WINNT\System32\cmd.exe
#####
# ApacheFriends XAMPP setup win32 Version 1.4.1
# -----
# Copyright (c) 2002-2005 ApacheFriends
# -----
# Authors: Kay Vogelgesang <kvo@apachefriends.org>
# Carsten Wiedmann <webmaster@wiedmann-online.de>
#####
Configure for server 1.4.16
Configure XAMPP with awk for 'Windows_NT'
Please wait ...
Enable AcceptEx Winsocks v2 support for NT systems DONE!

##### Have fun with ApacheFriends XAMPP! #####
```

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



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



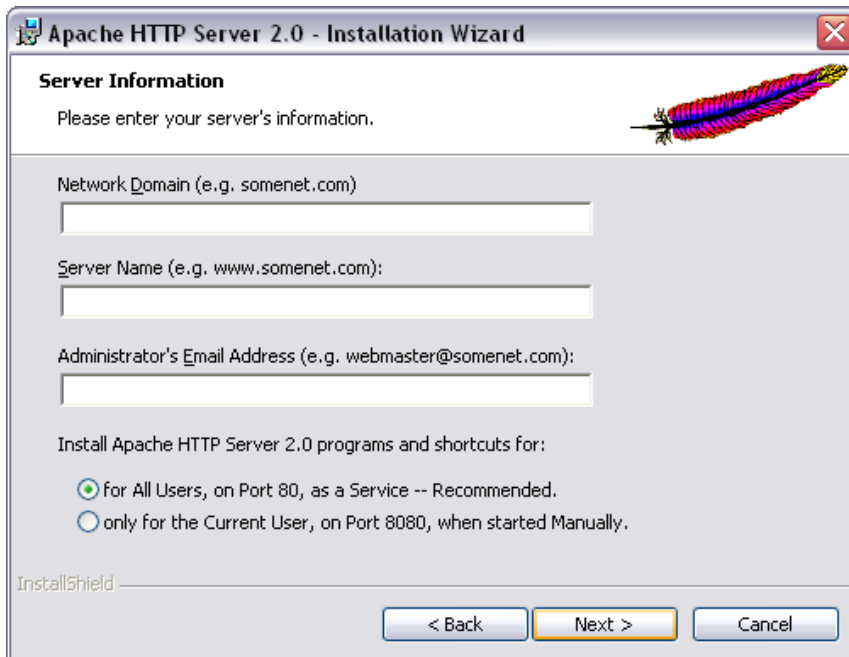
Ha az XAMPP, vagy más hasonló komplex programcsomag alkalmazása mellett döntünk, akkor a fejezet hátralevő részében bemutatott önálló alkalmazástelepítéseket már nem kell elvégeznünk, a rendszerünk kész a webfejlesztés tanulására.

1.2.2 Az Apache telepítése⁷

Az Apache webszerver letöltési oldaláról⁸ töltsük le a megfelelő telepítő állományt. A telepítő varázsló ablakai közül a következőket érdemes kiemelni:

⁷ További forrás: <http://weblabor.hu/cikkek/apachephptelepites>

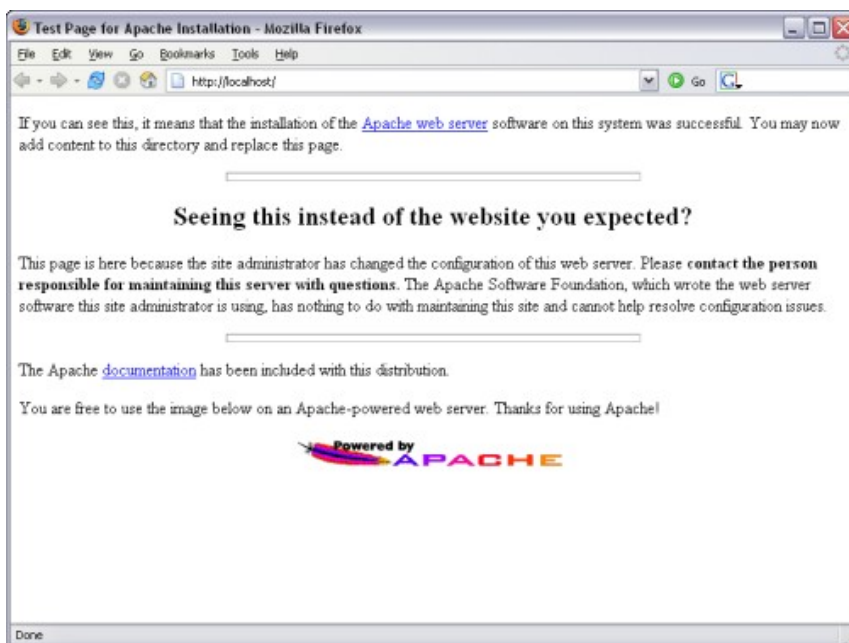
⁸ <http://httpd.apache.org/download.cgi>



Saját gépre telepítés esetén kétszer *localhost* megadása javasolt, az e-mail címnek ebben az esetben nincs jelentősége. Érdeemes szolgáltatásként (service) telepíteni a szervert, ugyanis a 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:



Az óra mellett megjelenik az Apache Monitor:



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

A konfigurációval 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.

1.2.3 A PHP telepítése⁹

A PHP telepítése is a kívánt telepítőkészlet letöltésével kezdődik. A letöltési oldalon¹⁰ 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 elegendő. 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 telepíteni a PHP-t. Ehhez először is a megfelelő *dll* állomány használatát elő kell írunk (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.

⁹ További források: <http://weblabor.hu/cikkek/apachephptelepites>, <http://hu.php.net/manual/hu/install.windows.php>

¹⁰ <http://www.php.net/downloads.php>

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

1.3. Fejlesztőeszközök

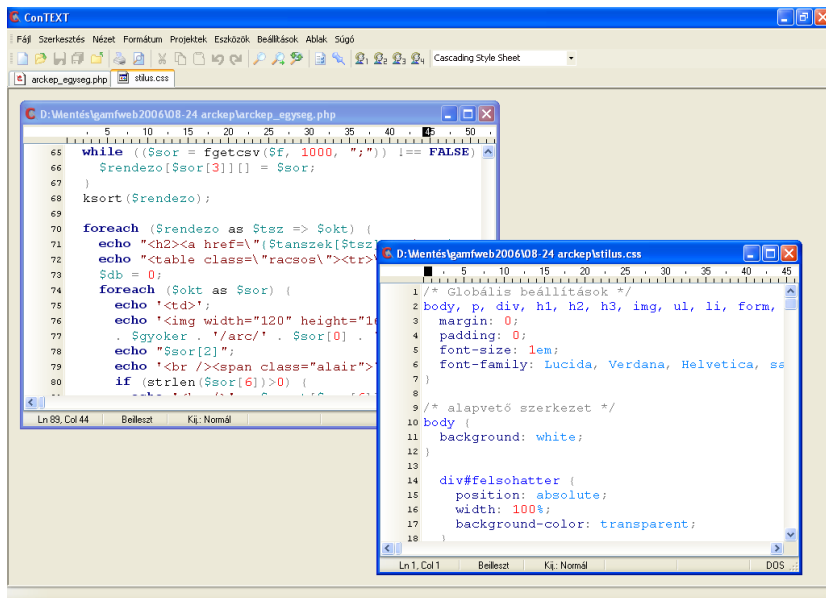
Fejlesztőeszközök témájában érdemes először a szélsőséges példákat áttekinteni.

Az eszközök közül a Jegyzettömb az a minimum, amivel még többé-kevésbé lehet webfejlesztést végezni, bár több okból sem célszerű. Egy valamivel jobb szerkesztő (Például NotePad+¹¹) viszont már használható.

A másik végletet azok a WYSIWYG (What You See Is What You Get) programok képviselik, amelyek úgy teszik lehetővé az oldal elkészítését, hogy lényegében 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.

A kódszínezést nyújtó programok közül a szerző kedvence a Context¹², amely – többek között – magyar felülettel több, mint 200 nyelv színezését tudja megoldani. Ezen 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.



1.4. Grafikus programok

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

¹¹ <http://mypecee.org/rogsoft/notepad.html>

¹² <http://www.context.cx/>

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, színeskorrekciót, világosítást stb. végrehajtani. E feladatok ellátására részben a Paint is megfelel, de érdemes valamivel komolyabb szoftvert alkalmazni.

A szerző sokáig a Jasc Paint Shop Pro híve volt, de a közelmúltban áttért a teljesen ingyenes Paint.NET¹³ programra.



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

¹³ <http://www.getpaint.net/>

2. A webfejlesztés alapjai

Weboldalakat azért készítünk (és készíttetnek velünk), mert szeretnénk valamit nyújtani a felhasználóknak. A webfejlesztés – mint az ipari alkotó tevékenység általában – a felhasználók igényeit nem hagyhatja figyelmen kívül.

Megjegyzés: A művészi önkifejezés, vagy éppen a blogolás látszólag háttérbe szorítja ezt az elvet, de csak látszólag.

2.1. Tervezési, fejlesztési szempontok

Egy weboldal színvonalas elkészítését mindig komoly tervezés előzi meg. A végcél a felhasználók megfelelő kiszolgálása, ezért fontos, hogy a felhasználók viselkedését, igényeit figyelembe vegyük.

2.1.1 Honlaptervezés

A honlaptervezés a célok megfogalmazásával kezdődik. Kinek szól? Mit szeretnénk közölni, átadni? Milyen felhasználói interakciókra lesz szükség?

Egy nagyobb honlap tervezésénél különösen fontos, hogy a látogatók fejével gondolkozunk, és a számukra áttekinthető oldalszerkezetet alakítsuk ki. Tipikus hiba lehet, hogy egy céges honlapon a cég belső szerkezete kap hangsúlyt ahelyett, hogy a látogatók igényeire összpontosítana.

Ha például a honlapon keresztül termékeket szeretnénk eladni, akkor a termékek több szempont szerinti kategorizálása, kereshetősége, valamint alapos bemutatása (képek, videofelvételek, pontos és részletes adatok, hivatkozások a gyártó oldalára, termék-összehasonlítási lehetőség) sokkal fontosabb, minthogy kik a cég vezetői és mi a cég szerkezeti felépítése.

Megjegyzés: Ez az elv sem mindig egyszerűen alkalmazható. A szerző több éve fejleszti a GAMF¹⁴ honlapját, és nem kevés fejfájást okoz az óta sem a kérdés: a központi honlap és az egyes szervezeti egységek honlapjai milyen arányban szerepeljenek, hogyan viszonyuljanak egymáshoz? Vagy még radikálisabban: kellene-e egyáltalán tanszéki honlapok?

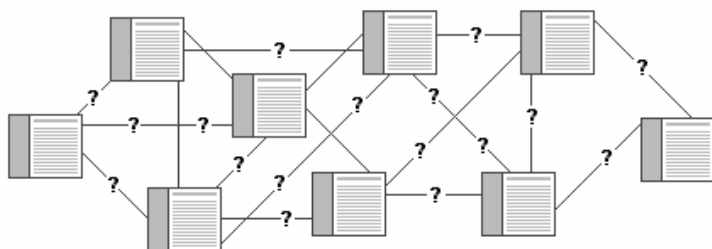
Egy felsőoktatási intézmény esetén a hallgatók hétköznapi életében is sokszor megmutatkozik a kar szerkezeti felépítése. (Például a szakdolgozat jelentkezési lapot annál a tanszéknél kell leadnia a hallgatónak, ahová a választott konzulense tartozik.) Így nem tűnik indokoltnak a teljes központi szervezés. A szerző azonban arra törekszik, hogy – akár redundáns tartalmak árán is – több oldalról megközelítve el lehessen jutni a keresett információig.

Például minden tanszéken van olyan oldal, ahol az oktatók fényképei, elérhetőségei szerepelnek. De mit tegyen a hallgató, ha nem tudja az oktatójának sem a nevét, sem a tanszékét, csupán az arcára emlékszik? Lapozza fel az összes tanszéki oldalt? Nem! Legyen egy oldal ehhez hasonló esetekre, ahol az összes oktató alapvető fényképes tájékoztatója szerepel. Ebből is látszik, hogy az egyes tartalmakat bizonyos esetekben akár több helyen, többféle formában is érdemes nyújtani.

2.1.2 Navigációs struktúra

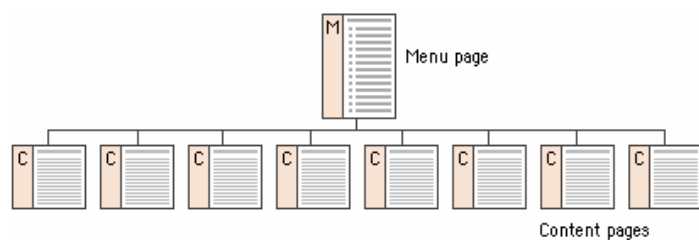
A honlap szerkezetének kialakításában nagyon fontos, hogy a tartalmi szerkezet, az oldalak közötti navigáció és az egyes oldalak navigációs lehetőségei teljes egységet alkossanak. A következő ábra egy zavaros szerkezetű honlapot mutat:

¹⁴ <http://www.gamf.hu/>

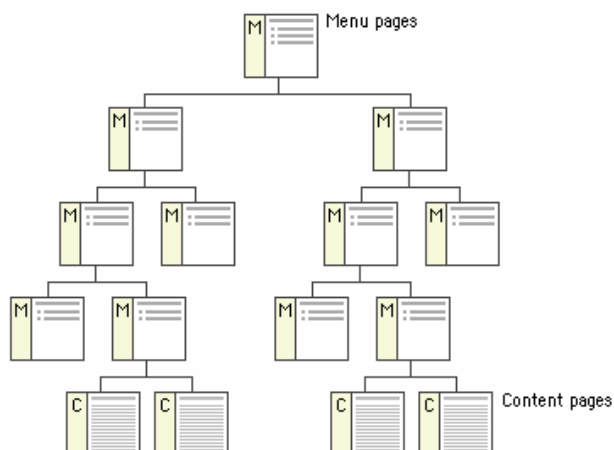


Általában faszerkezetű hierarchia kialakítására érdemes összpontosítani, és szükség esetén gráffá tovább bővíteni.

Nem magától érthető az sem, hogy egy menü mennyi elemet tartalmazzon. Az egyik véglet az egyszintű felépítés, ahol a sok lehetőség közül nehéz választani:



A másik véglet esetén a sokszintű felépítésben nem egyszerű megtalálni az adott tartalmat:

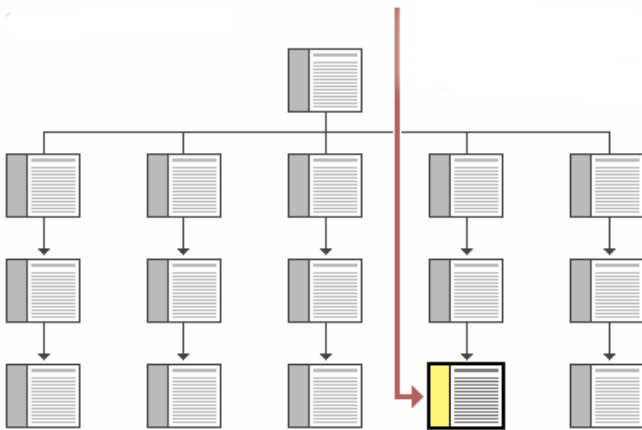


Meg kell tehát keresni az arany középutat a két véglet között, és további alternatív navigációt is lehetővé tenni.

Megjegyzés: Jól megfigyelhető a tipikus blog oldalak többféle navigációt lehetővé tevő felépítése. A más oldalak esetén jellemző (egy vagy többszintű) kategóriába való sorolás és a keresés alap navigáció természetesen itt is jellemző. De ezen kívül még fontosabbá vált az időbeli elhelyezkedés, a címkék (tagok) menti gyors elérési lehetőség, sőt a más blogokkal való kapcsolatok is.

A navigáció irányai

A következő ábra jól mutatja, milyen problémával találja szemben magát a látogató, ha a nyilakkal jelzett módon a navigációs irány csak fentről lefelé létezik.



Külső oldalról (pl. Google keresési oldal) érkező a továbblépés nem igazán lehetséges. Mindig gondoskodni kell tehát arról, hogy bármilyen irányban tovább tudjunk lépni egy adott oldalról, akár felfele és oldalra is:



A „felfelé” irányú elsősorban közvetlen szülő is jelenti, és természetesen a kezdőoldalra való ugrás is alapvető.

2.1.3 Oldaltervezés

Vizsgálódásunkat az egész honlap tervezéséről az egyes oldalak tervezésére helyezzük át.

Az olvasók pásztáznak

Ha azt hisszük, hogy a látogatónk a teljes oldalt el fogja olvasni, akkor tévedünk. Mivel az interneten rengeteg forrás van, a látogató néhány másodperc alatt eldönti, hogy az oldal számára fontos, értékes információt tartalmaz-e. Ha nem látszik első ránézésre, hogy mit is tartalmaz az oldal, akkor a látogató valószínűleg nem kezdi el az oldalt bogarászni.

Megjegyzés: Természetesen itt is vannak kivételek. Ha az oldal tartalma elsődlegesen más médiumon jelenik meg (pl. ez a jegyzet elsődlegesen A4-es méretű nyomtatásra van tervezve, de HTML formában is elérhető), vagy a jellegénél fogva az olvasó tudja, hogy a keresett információ itt van (pl. egy törvény szövegét olvassa), akkor „csupán” annyi a következő szempontok jelentősége, hogy jobban tájékozódik az oldalon. A következő elvek tehát elsődlegesen a webre írt tartalom esetén érvényesek.

Érdeemes átgondolni a saját tapasztalatok alapján, hogy mik is segítik az oldal gyors áttekintését:

- kifejező fő és alfejezetcímek,
- rövid összefoglaló a cím után (szokás félkövéren vagy nagyobb betűmérettel kiemelni),
- linkekkel és egyéb jelölésekkel kiemelt kulcsszavak, gondolatok

- a bekezdéseink legyenek rövidek és lényegre törők
- a fejezetek legyenek rövidek és áttekinthetők
- hagyjunk megfelelő távolságot az egyes bekezdések, fejezetek és címek között
- egy sorba lehetőleg ne kerüljön 70-nél több betű (bár a 70-es számban nincs teljes megegyezés, a hosszra mindenképpen érdemes figyelni)
- az oldalaink ne legyenek túl hosszúak
- ha mégis hosszú az oldal (néha indokolt lehet), akkor segítsük az oldalon belüli navigációt bevezető tartalomjegyzékkel, linkekkel

A témához a Pszichológia Online (<http://www.pszichologia.hu/>) Webdesign alapok I.-III. cikkeit¹⁵ javasoljuk tanulmányozni.

Oldal navigáció

A honlap minden oldalán egységes szerkezetű, jól áttekinthető menürendszer legyen az oldal felső, vagy valamelyik oldalsó részének felső részén. (Hasznos lehet az oldalon elhelyezett másodlagos navigáció is.) A látogató mindig tudja, hogy hol van a honlapon belül, és tudjon a megfelelő irányba navigálni.

Megjegyzés: Az oldalon elhelyezett egyetlen JavaScript-es vissza link (`Vissza`) a lehető legrosszabb megoldás. A látogató lehet, hogy a Google oldaláról jutott egy belső oldalra, és keresné az oldalhoz kapcsolódó egyéb oldalakat, de így nem találja, hiszen a link a Google oldalára fogja visszavinni.

Viszonylag kevés honlapon használják, pedig sokszor célszerű megoldás az ún. **halszálkás menü**, ami az adott oldal tartalmi szerkezetén belüli helyzetét mutatja.

A Weblabor honlapja¹⁶ jól mutatja a fő navigációs elemeket:

- a WL logó a kezdőoldalra visz, bárhol is vagyunk
- kétszintű főmenü (itt túl sok helyet foglalna, ha egyszerre látszana a két szint összes választási lehetősége)
- halszálkás menü mutatja, hogy az éppen aktuális oldal egy cikk (Cikkek) az adatbázis (Adatbázis) kategóriából



Megoszlanak a vélemények arról, hogy a szövegben levő, vagy a szöveg melletti hivatkozások hasznosabbak. Az azonban mindenképpen fontos, hogy legyenek hasznos hivatkozások a szövegben és a szöveg mellett is.

¹⁵ <http://www.pszichologia.hu/cikk/cikk.phtml?id=38>

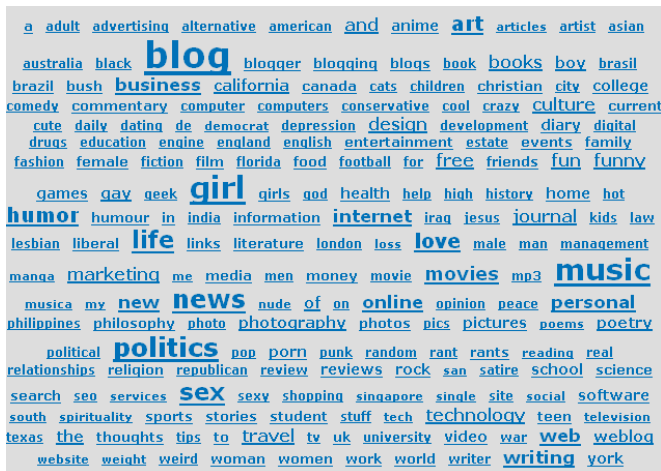
¹⁶ <http://weblabor.hu/>

Megjegyzés: A szerző véleménye szerint a szövegben célszerű azokat a hivatkozásokat elhelyezni, amelyek egy szóhoz, névhez vagy kifejezéshez jól kapcsolhatók. Ezzel szemben olyan hivatkozásokat, amelyek az oldal egészéhez kapcsolódnak, mindenképpen oldalra érdemes tenni.

Az oldal alján levő hivatkozások nem mindig szerencsések, hiszen ha nem olvassa, vagy görgeti végig az oldalt a látogató, akkor nem is veszi észre azokat. Másodlagos szerepet azonban sokszor betölt.

Alternatív navigáció

A hierarchikus felépítés nem minden esetben jó megközelítés. Főleg blog típusú oldalak esetén terjed a címkék (*tag*) vagy kulcsszavak használata, ahol az egyes címkék, és az oldalak között több-több kapcsolat van. A címkéket összefoglaló oldalakon a betűmérettel is szokás jelezni a címkék relatív gyakoriságát:



Talán érdemes lenne ezt is egyre több oldalnál alkalmazni, mint egy újabb lehetőséget, nem elfelejtve a logikai hierarchiából következő lehetőségeket sem.

A letöltési sebesség

Általános hiba, hogy a fejlesztő a saját gépén, hálózati elérés nélkül fejleszti a honlapot, és nem veszi figyelembe, hogy a látogatóknak meg kell majd várni az oldal letöltődését. Soha nem lesz minden felhasználónak olyan sebességű internet kapcsolata, mint esetleg a fejlesztőnek. Itt nem csak a ma már egyre jelentéktelenebb számú modemes elérésre, hanem a több gép között megosztott hálózati kapcsolatra, vagy az egyre elterjedtebb mobil böngészésre is érdemes gondolni. Mindig lesznek olyan felhasználók, akik ilyen hibás tervezés miatt nem fogják az oldalt látogatni. Egy felmérés szerint 7 másodperc után a látogató másik oldalra megy, ha az oldalnak legalább egy része nem lesz használható (olvasható). Néhány tipp arra nézve, hogy mit tehetünk a kevésbé gyors kapcsolattal rendelkező látogatók megtartásáért:

- Az oldal képek nélkül is olvasható legyen, hogy a képek letöltődéséig is tudja a felhasználó hasznosan tölteni az idejét. Az oldal címei tehát ne képpel, hanem szöveggel (is) jelenjenek meg. Ha mindenképpen képet használunk, akkor legalább az *img* tag *alt* tulajdonságot adjuk meg.
- Adjuk meg a képek méretét a HTML forrásban, így nem fog „ugrálni” az oldal a képek tényleges letöltődésekor, és a látvány sem csak a teljes letöltődésre fog összeállni.
- Ne használjunk nagy méretű táblázatot az oldalon (főleg az oldal szerkezetének kialakításához ne), mert a teljes táblázat letöltődéséig nem fog összeállni a látvány. Több kisebb táblázat egyébként is áttekinthetőbb lehet.

2.1.4 Figyeljünk a látogatók visszajelzéseire

Ha egy látogató nem csak az orra alatt zsörtölődik, hanem még veszi is a fáradságot, hogy leírja a véleményét, akkor azt valószínűleg érdemes figyelembe venni. Egy külső szemlélő gyakran könnyebben észreveszi a hibákat, nehezen használható részeket.

Például a Joomla! Magyarország honlapján¹⁷ történt 2006 augusztusi dizájn (és ebből következően navigáció) váltás után a kezdőoldalra linkelt, „*Mi a véleményetek az új oldalról?*” című fórum 24 óra alatt 32 hozzászólást eredményezett, amiből a sok dicséret mellett több mint 10 építő kritikai vélemény volt, aminek nagy részét még aznap be is építették a dizájnba!

Megjegyzés: Ennél a szélsőségesen pozitív példánál azért figyelembe kell venni, hogy a honlap látogatói jelentős részt maguk is webfejlesztők.

2.1.5 Milyen monitort használnak a látogatók?

Alapvetően több kérdést is fel kell vetni: mekkora a képernyő felbontás, mekkora a tényleges képernyő méret, és hogy fog kinézni az oldal a képernyőn.

Az a probléma, hogy ezekről nem sok biztosat tudunk megállapítani. Statisztikai eredményeket egyedül a képernyő felbontásokról lehet tudni, de ez nagyon szegényes információ több okból is. A felhasználó nem biztos, hogy teljes képernyős méretben használja a böngészőt, pl. nagy monitorok (19, 21 col) esetén. Nem tudjuk, hogy a böngésző ablakon belül mennyi eszközsort alkalmaz, vagy éppen a tálcája melyik oldalán és milyen méretben van. Nem tudjuk azt sem, hogy milyen böngésző-beállításokat alkalmaz, és azok pontosan hogyan befolyásolják a megjelenítést.

El lehet azt mondani, hogy nagy hibát követ el a fejlesztő, ha csak a saját környezetén teszteli az oldalt. 2006 nyarán célszerű a 800x600-tól legalább az 1600x1200-as felbontásig tesztelni többféle operációs rendszer és többféle böngésző alatt. Sőt azt is érdemes figyelembe venni, hogy 1-2 éven belül a mobil telefonról böngészők számottevő résztvevői lehetnek a látogatóknak. Ráadásul (már a drágább technológia használatából sejtetően) fizetőképesebb látogatói rétegről lehet szó.

Végül azt a súlyos (tév)hitet is érdemes megvizsgálni, miszerint az oldalnak minden böngészőn és minden felbontásban pixelre pontosan ugyanúgy kell kinézni. A szerző még nem találkozott egyetlen felhasználóval sem, aki képernyőképeket összehasonlítgatva pixel-eltéréseket keresne egy oldalon :). A felhasználónak egyetlen igénye az, hogy az ő esetében jól működjön az oldal: átlátható és szép legyen.

2.1.6 Milyen böngészőt használnak a látogatók?

Sok fejlesztő az utóbbi években kizárólag egy böngészőre (Microsoft Internet Explorer) és egyetlen képernyőfelbontásra „optimalizálta” az elkészült oldalt. Pontosabb úgy fogalmazni, hogy nem vette a fáradságot, hogy foglalkozzon a látogatók igényeivel. Az utóbbi egy év szerencsére pozitív tendenciákat mutat: a kb. 30% részesedéssel rendelkező egyéb böngészőt alkalmazó felhasználóiról is kezdenek tudomást venni a fejlesztők.

Komolyabb problémáik főleg azoknak a fejlesztőknek vannak, akik az Explorerre írt kódjukat szeretnék a többi böngészőre is használhatóvá alakítani. A gyakorlatban azonban sokkal jobban beválik, ha a szabványok pontos ismerete alapján felépített oldal készí-

¹⁷ <http://joomla.hu/>

tünk. Innen általában kisebb lépés az egyes böngészők speciálisabb igényeinek kielégítése.

2.1.7 Mi van a sérültekkel?

Vannak emberek, akiknek a látásuk, hallásuk sérült, vagy teljesen hiányzik. Ők is szeretnék használni a webet. Sőt érdemes belegondolni, hogy egy – a számítógép kezelését ismerő – vak vagy látássérült több esetben rá van kényszerülve, hogy a teendőit interneten végezze, mint a jól látók, hiszen már a fizikai közlekedés is problémás lehet számára.

Vakok és látássérültek számára felolvasó szoftverek vagy Braille perifériák teszik a weboldalak szöveges tartalmát hozzáférhetővé. Ráadásul ez azt is jelenti, hogy a szöveg az elejétől a végéig sorosan olvasható, de nem áttekinthető vizuálisan, és a szövegben történő navigáció is elég körülményes. Ezt a helyzetet látókként úgy próbálhatjuk ki, hogy egy weboldalt szöveges böngészővel nézünk meg (semmi kép, semmi flash, semmi formázás:), például a közismert Total Commander Nézőkéjével (F3) egy HTML állományt megnyitunk, majd olyan kicsire méretezzük az ablakot, hogy egyszerre csak egy szót lássunk.

Tudatos odafigyeléssel lényeges plusz munka nélkül elérhető, hogy a tartalmaink a sérült emberek számára is elérhetőek legyenek.

A témához további információk találhatóak pl. a Weblabor Akadálymentes weboldalak¹⁸ cikksorozatában.

2.2. A web szabványai

A webfejlesztők gyakran küzdenek a különböző böngészőkben és böngészőverziókban is **használatos** 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 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 legalapvetőbb módszer.

2.2.1 A World Wide Web Consortium¹⁹ (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:

¹⁸ <http://weblabor.hu/cikkek/akadalymentesweb1>

¹⁹ <http://www.w3.org/>

- IBM
- Microsoft
- America Online
- Apple
- Adobe
- Macromedia
- Sun Microsystems

Akár személyesen is bekapcsolódhatunk az ajánlások kidolgozásába a W3C Magyar Iroda²⁰ által koordinált módon. (2006. október 18-án például W3C Mobilweb Műhelykonferencia²¹ lesz.)

2.2.2 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.

2.3. Ellenőrző kérdések

- Melyik az utolsó HTML (nem XHTML!) verzió?
- Minek a rövidítése: HTML, W3C?
- Mi a W3C feladata?
- Melyek jelenleg a szabványkövető böngészők?
- Milyen verziónál tartanak ma a népszerűbb böngészők?

²⁰ <http://www.w3c.hu/>

²¹ <http://www.w3c.hu/rendezvenyek/2006/mobilweb/index.html>

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 lehetőséget 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.
- 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

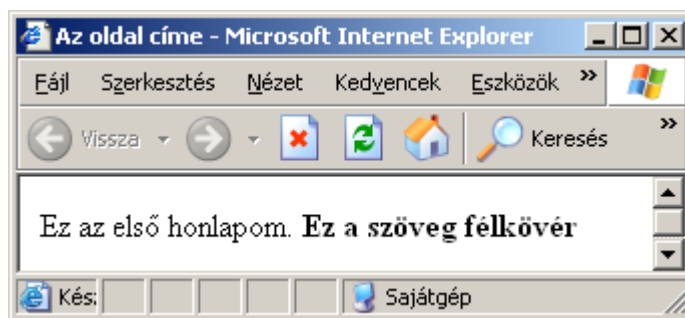
Hogyan kezdjük neki?

Windows operációs rendszer alatt indítsuk el a Jegyzettömböt, 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!

Nyissuk meg a böngészőnket, majd a Fájl menü megnyitás parancsát választva keressük meg az előbb elmentett *oldal.html* állományt! A következőhöz hasonlóan kell látnunk a böngészőnkben:



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 `` és `` 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 begépelte szöveget. Miért?

Válasz: A Windows alapértelmezett beállítása szerint a Jegyzetomb 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 alkalmazni, vagy a Mappa beállításai között a bűvös „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. 2006 májusában a következő böngészőt használják a látogatók:

56.7% Internet Explorer 6
25.7% Mozilla Firefox 1-1.5
4.5% Internet Explorer 5
2.3% Mozilla
1.5% Opera 7-8
1.1% Internet Explorer 7

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 és
- 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 , a tartalmazott szöveg *Ez a szöveg félkövér*, és a záró tag .

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.

3.3. Alapvető HTML tagok

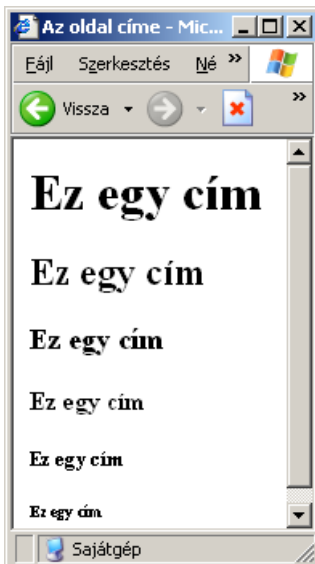
A HTML legalapvetőbb és leggyakrabban használt tagjai azok, amelyek segítségével címeket, bekezdéseket és sortöréseket lehet létrehozni.

A HTML tanulásához a legjobb módszer, ha példákkal dolgozunk.

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 *cimek.html* oldal 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>
```

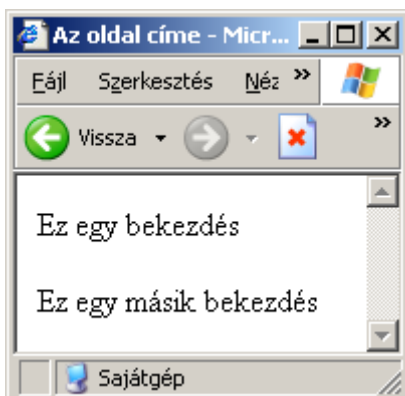


A címsorokhoz a böngésző alapértelmezetten térközöket is kapcsol.

Bekezdések

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

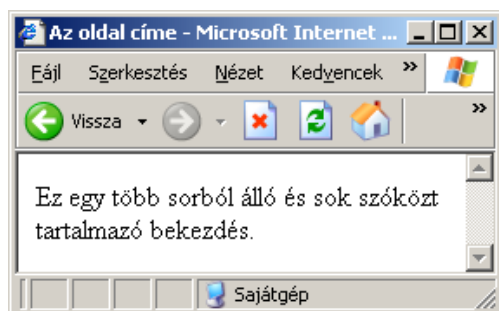
```
<p>Ez egy bekezdés</p>
<p>Ez egy másik bekezdés</p>
```



A bekezdésekhez a böngésző alapértelmezetten térközöket is kapcsol.

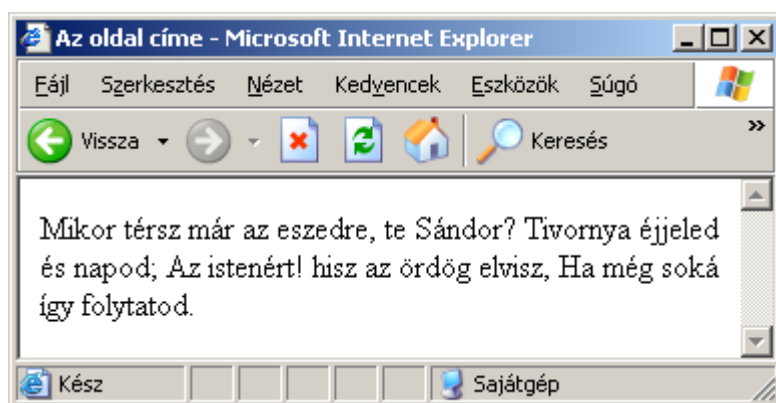
A következő példában hiába szerepel az újsor és a több szóköz karakter, a böngésző minden elválasztó karakter-sorozatot egy szóközként értelmez. A tényleges tördelést mindig a böngésző mérete és a benne levő szöveg határozza meg.

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



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

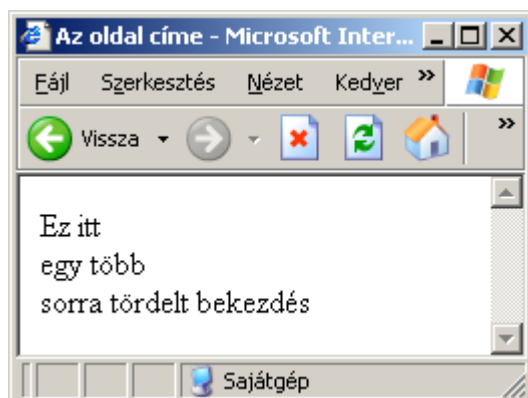
```
<p>
  Mikor térsz már az eszedre, te Sándor?
  Tivornya éjjeled és napod;
  Az istenért! hisz az ördög elvisz,
  Ha még soká így folytatod.
</p>
```



Sortörések

A `
` 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>
```



A `br` tag üres, vagyis nincs záró párja.

Megjegyzések

A megjegyzés tagot megjegyzések elhelyezésére használjuk. A böngésző nem veszi figyelembe a megjegyzésbe írt szöveget.

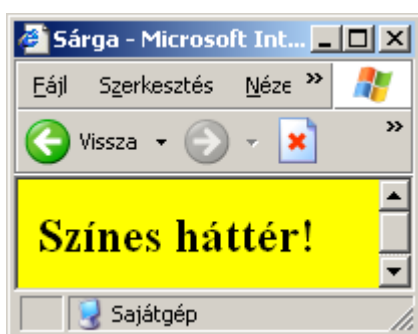
```
|<!--Ez egy megjegyzés -->
```

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>
```



Hasznos tippek

- Amikor HTML oldalakat hozunk létre, mindig nézzük meg más böngészőben, más képernyő (ablak) felbontásban is.
- Soha ne szóközzel és újsor karakterekkel (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
<code><html></code>	HTML dokumentumot definiál
<code><body></code>	A dokumentum törzsét definiálja
<code><h1> ... <h6></code>	Címsorokat definiál
<code><p></code>	Bekezdést definiál

<code>
</code>	Egyszerű sortörést szúr be
<code><hr></code>	Vízszintes elválasztó vonalat szúr be
<code><!-- ... --></code>	Megjegyzést definiál

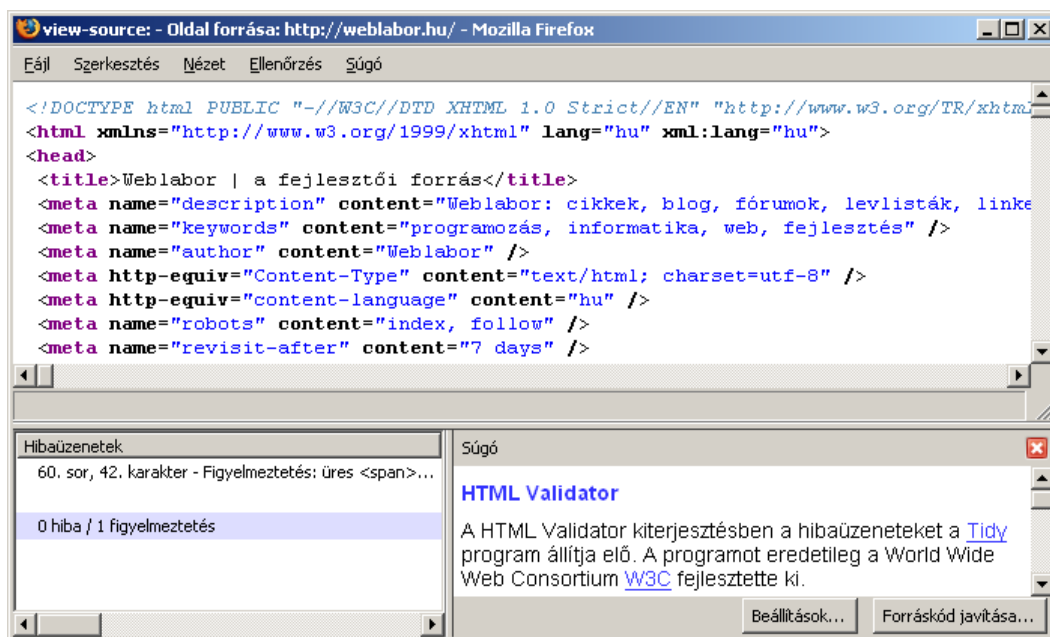
3.4. Hogy nézzük meg egy oldal HTML kódját?

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 fejlesztők 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*²²



A képen látható, ahogy a színelmelés alatt a szintaktikai hibák és a hibák részletes leírása is megtekinthetők.

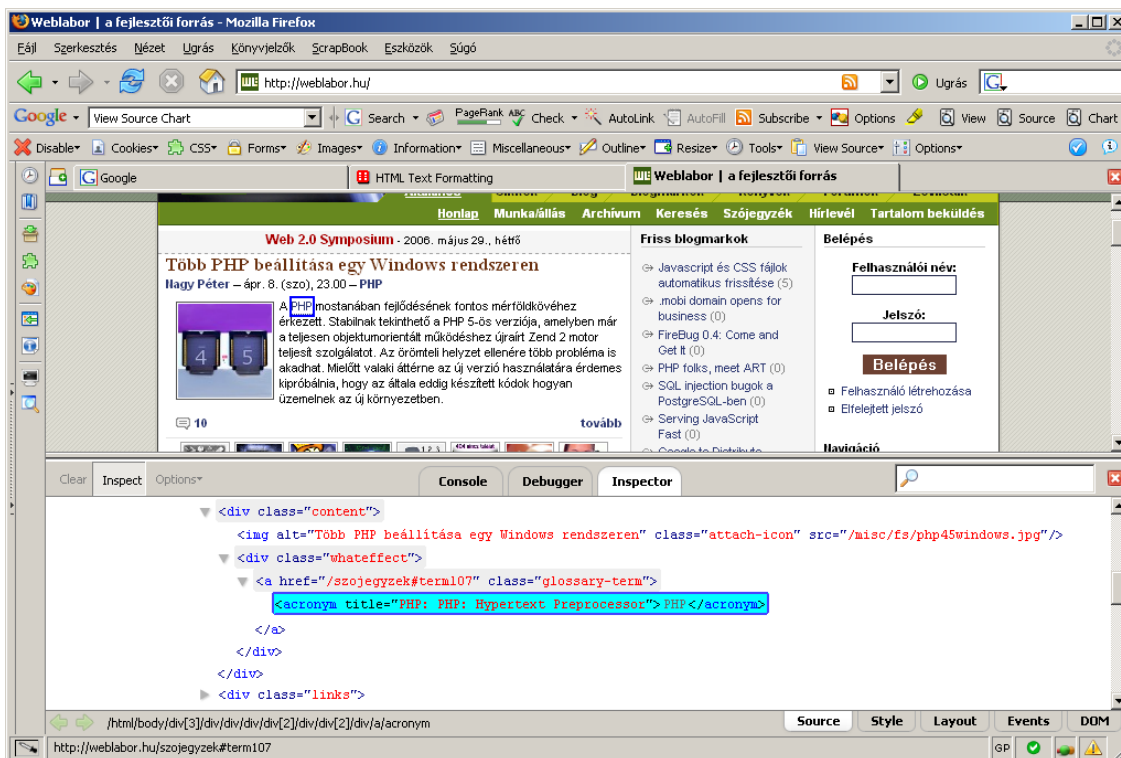
²² <http://users.skynet.be/mgueury/mozilla/>

View Source Chart²³



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.

FireBug²⁴



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

²³ <http://jennifer-madden.com/scripts/ViewRenderedSource.html>

²⁴ <http://www.joehewitt.com/software/firebug/>

3.5. Formázás

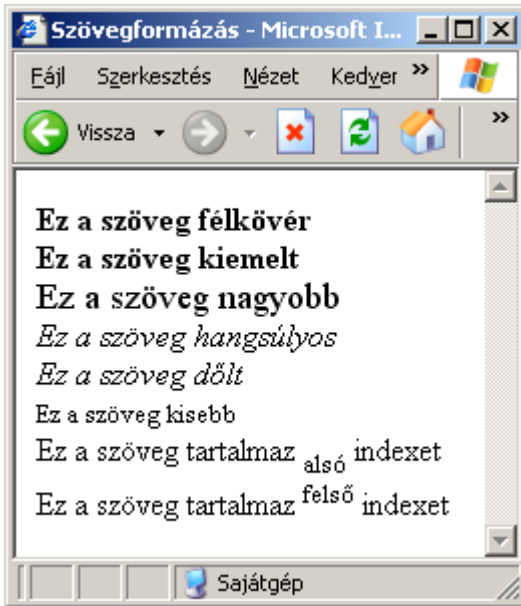
A HTML definiál néhány formázásra szolgáló elemet, mint pl. félkövér vagy dőlt szöveg formázásához.

Szövegformázó tagok

Tag	Leírás
<code></code>	Félkövér szöveget definiál
<code><big></code>	Nagyobb szöveget definiál
<code></code>	Kiemeli a szöveget
<code><i></code>	Dőlt szöveget definiál
<code><small></code>	Kisebb szöveget definiál
<code></code>	Erősebb kiemelés
<code><sub></code>	Alsó indexet definiál
<code><sup></code>	Felső indexet definiál
<code><ins></code>	Beszúrt szöveget jelöl
<code></code>	Törölt szöveget jelöl

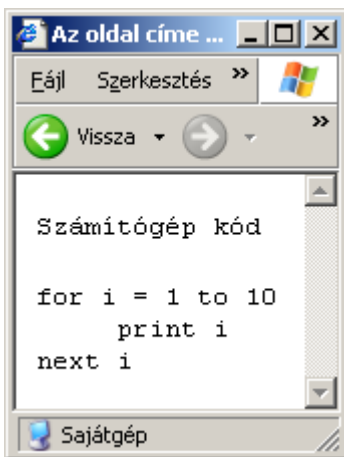
A következő példa mutatja a tagok hatását:

```
<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
```



Számítógép kimenetet definiáló tagok

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



Idézet, kiemelés és definíciós tagok

Tag	Leírás
<code><abbr></code>	Rövidítést definiál
<code><acronym></code>	Mozaikszót definiál
<code><address></code>	Cím elemet definiál
<code><bdo></code>	Szöveg írásirányt határoz meg

<code><blockquote></code>	Hosszú (akár több bekezdéses) idézetet jelöl
<code><q></code>	Rövid (általában bekezdésen belüli) idézetet jelöl
<code><cite></code>	Idézetet jelöl
<code><dfn></code>	Definíciót jelöl

```

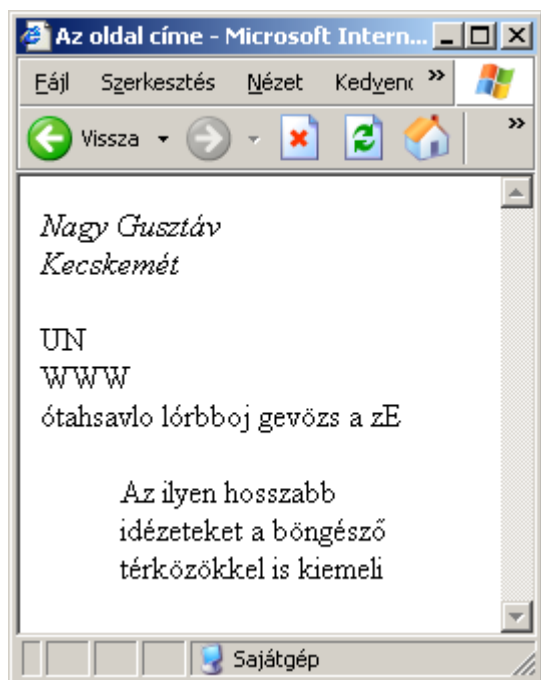
<address>
  Nagy Gusztáv<br>
  Kecskemét
</address>
<br>

<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 térközőkkel is
  kiemeli
</blockquote>

```



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 kód # után.

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 szóköz-ként 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 kódlap meghatározásra 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 másik 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 klikkelés 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 a klikkelhető, 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 akarja eldönteni, hogy mi hol jelenjen meg. Nem szükséges, hogy erről a webfejlesztő helyette döntsön.

A name tulajdonság

Egy hivatkozás alapértelmezetten a HTML oldal legelejét 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.)

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 az egy hátránya, hogy a vissza gomb hatására is az oldalon belül fog a felhasználó ugrálni.

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. (Érdeemes belegondolni, hogy nagy portálok szinte egyáltalán nem alkalmazzák.) A teljesség igénye nélkül néhány hátrány, mielőtt belekezdzenénk 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 jeleltsen 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 fáradságot, hogy a keretes változat mellett keretek nélkül is tökéletes funkcionalitással bíró változatot is elkészítsen. Akik mégis vették a fáradságot, 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 soron belüli keretet definiál. Ezt a tagot is érdemes elkerülni.

Hasznos tippek

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

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 felélben pedig vízszintesen 1:3 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 HTM kódjában megadjuk a linkek esetén alapértelmezett cél keret nevét a head szakasz base 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

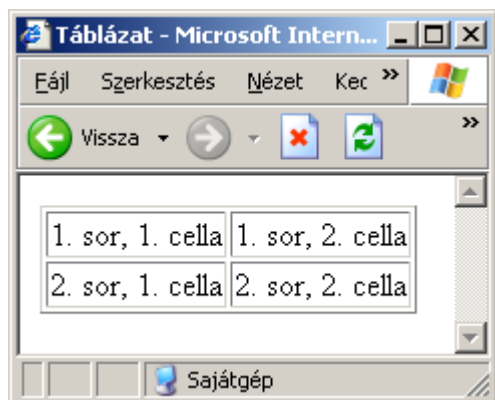
Tag	Leírás
<code><frame></code>	keretet definiál
<code><frameset></code>	keret-csoportot definiál
<code><noframes></code>	a keretet nem ismerő böngésző esetén megjelenő tartalom
<code><iframe></code>	szoron belüli keretet definiál

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, úrlapokat, 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>
```



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>
```

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:



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:

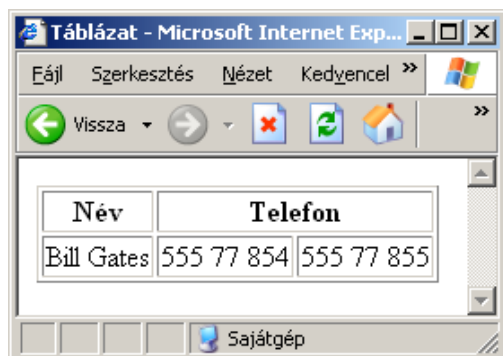
```
<table border="1">
  <caption>Táblázat címe</caption>
  <tr>
    <th>1. fejléc</th>
    <th>2. fejléc</th>
  </tr>
```



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>
```



Tippek

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

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

Összefoglaló táblázat

Tag	Leírás
<i><table></i>	táblázatot definiál
<i><th></i>	fejléctet definiál
<i><tr></i>	sort definiál
<i><td></i>	cellát definiál
<i><caption></i> >	címet definiál
<i><thead></i>	fejléctet definiál
<i><tbody></i>	táblázat törzset definiál
<i><tfoot></i>	lábéléctet definiál

3.10. Listák

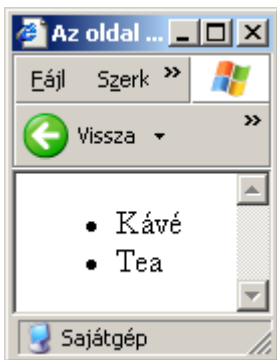
A HTML támogatja a számozott, felsorolt és definíció-listák létrehozását.

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 `` taggal írható le, a lista elem pedig `` taggal.

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

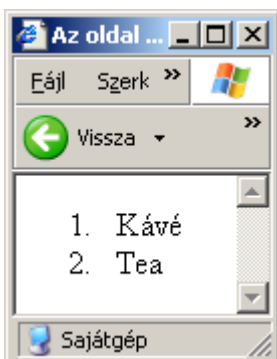


Számozott lista

A számozott listák elemei (többnyire) számmal azonosítottak.

A számozott listát `` taggal kell létrehozni, a lista elemek az előzőhöz hasonlóan `li`-vel definiálhatók.

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



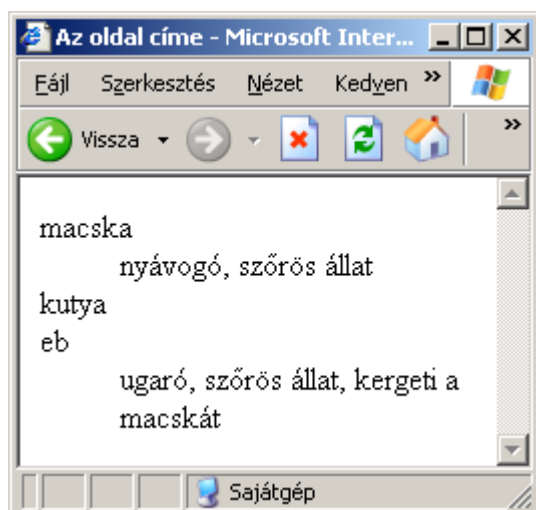
Definíciós lista

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>
```



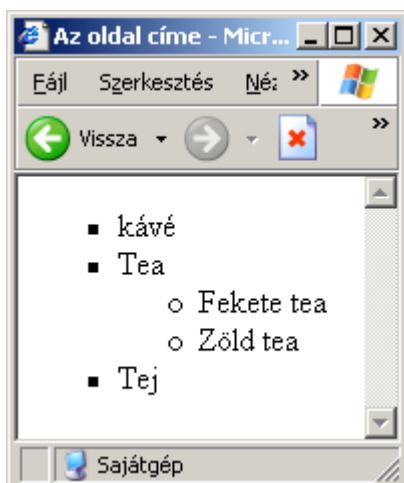
További példák

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>
```



Összefoglaló táblázat

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

3.11. Űrlapok

Az űrlapokat arra használhatjuk, hogy különböző módokon lehetőséget adjunk a látogatónak 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.

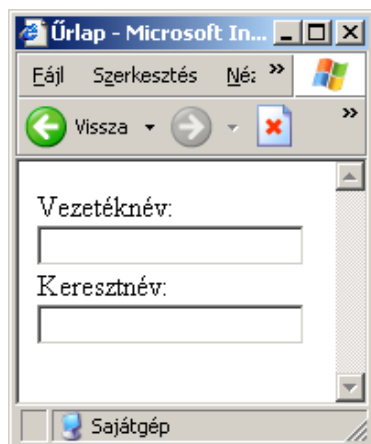
Szöveges mezők

A szöveges mezők lehetővé teszik, hogy betűkből, számokból, írásjelekből álló karakter-sorozatot lehessen begépelni.

```

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

```



Érdeemes 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 begépeltek szöveg helyett * karakterek jelennek meg.

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

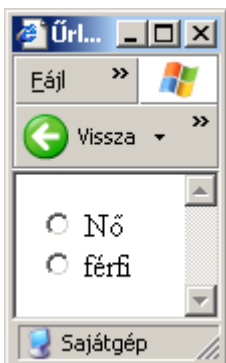
Rádiógombok

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

```

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

```



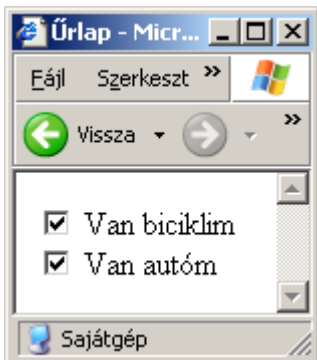
Ha valamelyik választási lehetőséget alapértelmezettnek tekintjük, akkor ezzel könnyíthetjük is az űrlap kitöltését:

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

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>
```



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

Érdeemes 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 klikkelni 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 *id* alapján éri el az elemeket.

Űrlap adatok elküldése

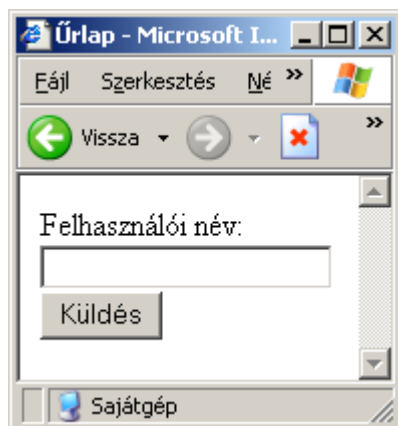
Az esetek jelentős részében a felhasználó azért tölt ki egy űrlapot, hogy adatokat tudjon a szerver felé küldeni valamilyen hatás érdekében.

Az eddigi példákból két fontos rész kimaradt. Először is a felhasználó számára szokás biztosítani egy küldés (vagy valami hasonló funkcionalitású) 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>

```



Ha a felhasználó begépel 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

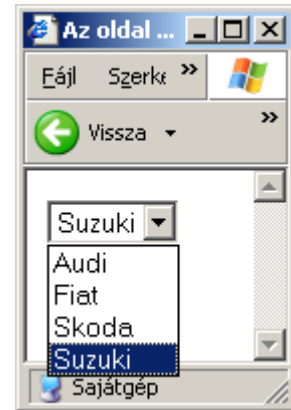
Bizonyos esetekben rádiógombok helyett célszerű inkább a lenyíló listák alkalmazása. (Itt elsősorban terjedelmi, áttekinthetőségi okokra kell gondolni.)

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>

```



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
<code><form></code>	úrlapot definiál
<code><input></code>	beviteli mezőt definiál
<code><textarea ></code>	többsoros beviteli mezőt definiál
<code><label></code>	címkét definiál
<code><select></code>	lenyíló listát definiál
<code><option></code>	lenyíló lista elemet definiál
<code><button></code>	nyomógombot definiál

3.12. Képek

A HTML nyelvben az `` 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:

```
| 
```

Az *url* lehet abszolút vagy relatív megadású is.

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.

```
| 
```

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”.

```
| 
```

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">
    
  </a>
</p>
```



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.

```


<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>

```



Érdeemes 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 szöveg-gel akár körbefuttathatóvá is válik. (Maga a körbefuttatás szintén CSS-el végzendő.)

Összefoglaló táblázat

Tag	Leírás
	kép elemet definiál

`<map>` képtérképet definiál

`<area>` területet definiál

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 három féle módon lehet megadni: hexadecimális számként, RGB értéként 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 több féle módon közelíthetjük meg. Itt most a monitorok képmegjelenítésénél használatos, éppen ezért a webfejlesztésben is általános RGB megközelítést alkalmazunk.

Az *RGB* színek három összetevőből állnak, mint piros (*red*), zöld (*green*), és kék (*blue*). Mindhárom érték egy bájtban tárolt előjel nélküli számként határozható meg, vagyis értéke 0 és 255 között (hexában 0 és FF között) lehet. Ezzel a módszerrel tehát a 3 bájtban 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* harcának egyik mellékterméke, hogy a böngészők felismernek, értelmeznek olyan HTML oldalakat is, amelyek nem felelnek meg a szabvány-

nak. Sőt a webfejlesztők ezekre a pontatlanságokra rá is szoktak, és ami az informatika-oktatás 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 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">
```

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 Validator²⁵

3.16. Stílusok

A HTML 4.01 minden formázási információt elkülönített stíluslapokra 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>
```

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.

²⁵ <http://users.skynet.be/mgueury/mozilla/>

Beágyazott stíluslap

Beágyazott stíluslapot akkor szokás alkalmazni, ha az oldal egyedi (a külső stíluslapot nem lehetne másik oldalhoz felhasználni), vagy nagyon egyszerű stílusról van szó.

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 egyetlen 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 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
<code><style></code>	stílus definíciót hoz létre
<code><link></code>	külső stíluslapot kapcsol az oldalhoz
<code><div></code>	blokk szintű elemet definiál
<code></code>	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* 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.

A *link* elem többek között a CSS, hanem JavaScript fájl linkelésére is alkalmas.

Összefoglaló táblázat

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

3.18. Szkriptek

Kliens oldali szkriptek segítségével az oldal interaktívabbá tehető.

A `<script>` tag 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 öngölt rügött.

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

3.19. Általános tulajdonságok

A HTML tagoknak lehetnek tulajdonságaik. 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
-------------	--------

<i>class</i>	az elem osztálya
<i>id</i>	az elem egyedi azonosítója
<i>style</i>	soron belüli stílus definíció
<i>title</i>	rövid segítség definiálása

Nyelvi tulajdonságok

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

Tulajdonság	Leírás
<i>dir</i>	az írás irányát definiálja
<i>lang</i>	a nyelvet adja meg

Billentyűzet tulajdonságok

Tulajdonság	Leírás
<i>accesskey</i>	gyorsbillentyűt rendel az elemhez
<i>tabindex</i>	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 klikkelés hatására az oldal egy része megváltozik.

Ablak események

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

Tulajdonság	Leírás
<i>onload</i>	akkor fut le a szkript, ha az oldal betöltődött
<i>onunload</i>	akkor fut le a szkript, ha a felhasználó az oldalt bezárja, vagy más oldalra lép tovább

Úrlap elemek eseményei

Egyedül úrlap elemek esetén használhatók.

Tulajdonság	Leírás
<i>onchange</i>	akkor fut le a szkript, ha az elem megváltozott

<i>onsubmit</i>	akkor fut le a szkript, ha az űrlap elküldését kéri a felhasználó
<i>onreset</i>	akkor fut le a szkript, ha az űrlap kezdőállapotba áll
<i>onselect</i>	akkor fut le a szkript, ha az elemet kiválasztja a felhasználó
<i>onblur</i>	akkor fut le a szkript, ha az elem elveszti a fókuszt
<i>onfocus</i>	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
<i>onkeydown</i>	akkor fut le a szkript, ha egy billentyűt lenyomott a felhasználó
<i>onkeypress</i>	akkor fut le a szkript, ha egy billentyűt lenyomott és felengedett a felhasználó
<i>onkeyup</i>	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
<i>onclick</i>	akkor fut le a szkript, ha egérklikkelés történt
<i>ondblclick</i>	akkor fut le a szkript, ha dupla egérklikkelés történt
<i>onmousedown</i>	akkor fut le a szkript, ha az egérgombot lenyomta a felhasználó
<i>onmousemove</i>	akkor fut le a szkript, ha a felhasználó az adott elem felett mozgatja a kurzort
<i>onmouseout</i>	akkor fut le a szkript, ha a felhasználó az adott elemre mozgatja a kurzort
<i>onmouseover</i>	akkor fut le a szkript, ha a felhasználó az adott elemről elmozgatja a kurzort
<i>onmouseup</i>	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 begépett 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
 elemmel, hanem valóban jelezzük a bekezdés végét és egy új kezdetét.
- A különböző listákat a lista elemekkel definiáljuk. (, , <dl>). Talán nem triviális, de a menüket is így érdemes megadni. (A menüpontok is egyfajta listát jelentenek.)

3.23. Validátor

A HTML kód ellenőrzésére a korábban említett Tidy mellett az online is használható W3c Markup Validation Service²⁶ is nagyon jó eszköz.

3.24. Ellenőrző kérdések

Melyik a helyes HTML dokumentum felépítés?

- a. `<html><head>...</head><title>Cím</title><body>A dokumentum szövegtörzse</body></html>`

²⁶ <http://validator.w3.org/>

- 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. ``
- c. ``
- d. `<list>`

Melyik tag segítségével készíthetünk egyszerű felsorolás listát?

- a. ``
- b. `<dl>`
- c. ``
- 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. ``
- c. ``
- d. ``

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,

hogya 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. kírja az utolsó frissítés dátumát
- c. 5 másodperc múlva betölti a megadott url-ről a dokumentumot

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. XHTML

Az XHTML (*EXtensible HyperText Markup Language* – kiterjeszhető HTML) a HTML pontosabb és tisztább verziója. A HTML 4.01 leváltására 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.

4.1. XML alapok

Ebben az alfejezetben röviden áttekintjük az XML alapjait, hogy az XHTML újonságait 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 az adatok leírásával, míg a HTML a megjelenítéssel foglalkozik. Az XML – bár nem kizárólag a webbel áll kapcsolatban, az utóbbi évek sok webes alkalmazást hoztak létre.

4.1.1 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 leírására, továbbítására, átadására van szükség.

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

Példa: Képzeliünk el egy olyan vállalatot, amelyik önálló informatikai infrastruktúrával, vállalatirányítási rendszerrel rendelkező kisebb cégeket kebelez be. A bekebelezés folyamatának végcélja lehet a bekebelezett cég teljes integrációja, de ez sokszor eléggé nehézkes lehet, és nem is biztos, hogy célszerű. Sokszor célravezetőbb megoldás lehet, ha „csupán” a rendszerek közötti aktív adatcsere-t teremtjük meg. Erre pedig a legegyszerűbb megoldás, ha definiálunk egy XML nyelvtant az adatcsere definiálására, és a két rend-

szerhez csupán egy XML csatolófelületet adunk hozzá. Így a két rendszer költséges újrainása helyett jóval kisebb költséggel megoldható az integráció.

Természetesen integrációs törekvések más esetben is előfordulhatnak. Jellemző eset még az is, hogy egy vállalatban 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.

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 megvalósulásának 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.

4.1.2 XML szintaxis

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

Nézzünk egy egyszerű példát, egy jegyzetomb 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-bités ASCII kódolással mindig leírható, a további sorok olvasásánál már ismert lesz a karakterkódolás.

A `<note>` és `</note>` tagok határolják az egész dokumentumot kitöltő gyöker 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).

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, és ezekből melyek kötelezőek, stb.

A következő példa nem külső állományban, hanem helyben definiálja a DTD-t, és mindössze a lehetséges tagokkal, valamint a gyökérelém 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>

```

4.2. 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>
```

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 
```

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 tartalmazzanak 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:

```

```

Helyes:

```
| 
```

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

```
| 
```

4.3. Dokumentumtípus-deklaráció

Bár a HTML is előírja a DTD meghatározását, ezt alig néhány oldalon használták 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.

4.3.1 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">
```

4.3.2 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.

4.4. Visszafelé kompatibilitás

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

4.5. További források

Magyar nyelven a HTMLInfo oldalán²⁷ olvasható a W3C ajánlásának fordítása.

Ugyanitt megtalálható az XML igen alapos definíciója²⁸ is.

²⁷ <http://htmlinfo.polyhistor.hu/xhtml11se/cover.html>

²⁸ http://htmlinfo.polyhistor.hu/xml_ref/rec-xml.html

5. CSS

5.1. Bevezetés

Mi a CSS?

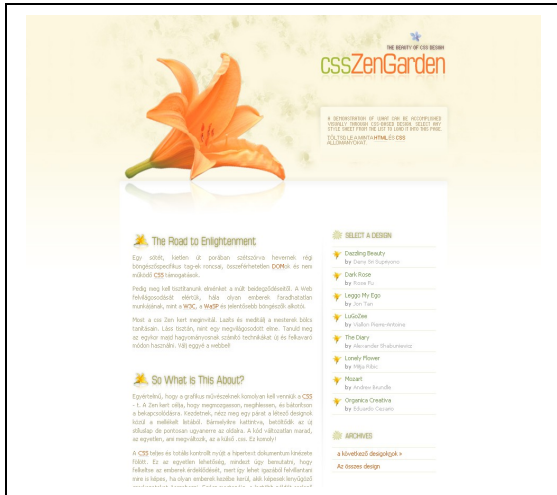
- A CSS a *Cascading Style Sheets* rövidítése
- A stílusok a HTML megjelenítési elemei és attribútumai helyett használhatók, azoknál jóval több lehetőséget biztosítva
- A stílusok meghatározzák, hogy hogyan jelenjenek meg vizuálisan a HTML elemek
- A stíluslapok segítségével könnyen szét lehet választani 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*²⁹ oldalát meglátogatni. Ez az oldal azzal a céllal jött létre, hogy a CSS-el szembeni ellenérzéseket lerombolja, és bemutassa, milyen óriási lehetőségek vannak a dizájnner 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:



²⁹ Magyar oldal: <http://www.csszengarden.com/tr/magyar/>



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*³⁰ (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.

³⁰ <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 stílus, 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.

5.2. 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.

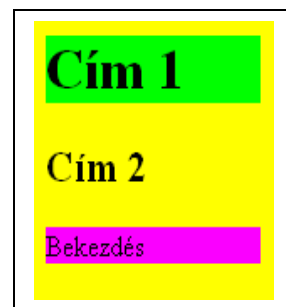
5.3. A *background* tulajdonságok

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

5.3.1 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)}
```



5.3.2 Háttérkép

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

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

Ismétlődés

A háttérkép alapértelmezés szerint kitapétázva jelenik meg, ha az elem mérete ezt szükségessé 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.

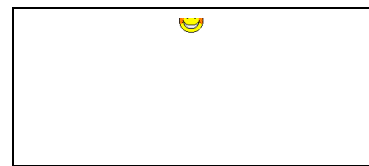
Pozíció

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 napocskánk:

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



Az érték megadásánál először a függőleges, majd a vízszintes pozíciót kell megadnunk. A 3-3 konstans (*top*, *center*, *bottom* és *left*, *center*, *right*) mellett százalékos és pixeles pozícionálás 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 háttérkép nem csak a görgetősávval együtt mozogva, hanem fixen pozícionálva 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;
```

5.4. Szövegek megjelenítése

A CSS szöveg tulajdonságai segítségével a szövegek vizuális megjelenítését lehet testreszabni.

A szöveg színe

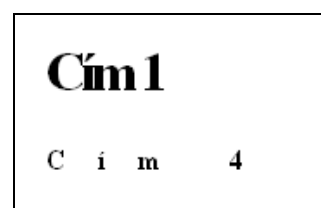
A szöveg színét a *color* tulajdonság határozza meg:

```
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 fölhúzott, áthúzott, aláhúzott 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 1cm-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.

5.5. 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 betűformázás összevonható akár egyetlen deklarációvá:

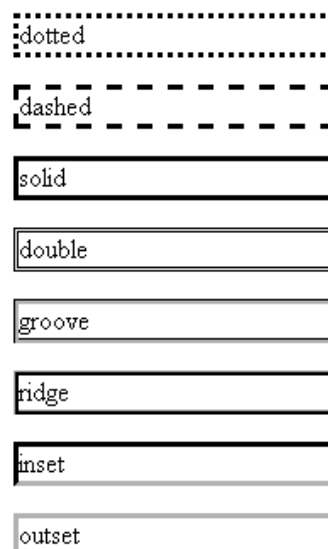
```
| p {font: italic small-caps 900 12px arial}
```

5.6. 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 oldalt:

```
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

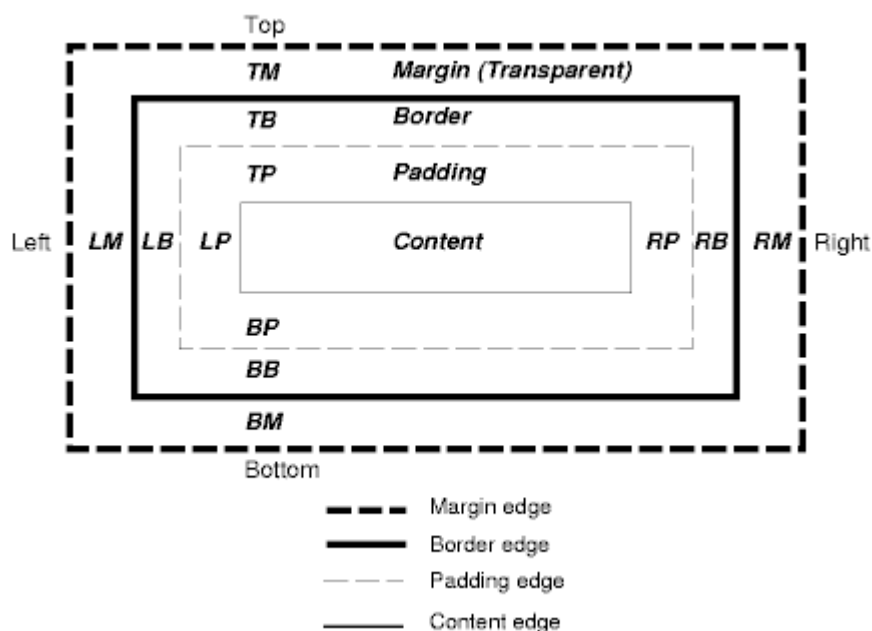
Az eddigi szegély tulajdonságok (hasonlóan a korábbi tulajdonságcsoporthoz) összevonhatók a *border* tulajdonság segítségével. Például az alsó szegély több tulajdonságának beállítása:

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


5.7. Térközök a szegélyen belül és kívül

A CSS szintaktikailag két nagyon hasonló tulajdonság-csoportot tartalmaz. A *margin* tulajdonsággal a szegélyen 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}
```

Mindden 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;
| }
```

5.8. 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 listaelemes sorrendisége nem tartalmaz információt, ezért minden listaelem előtt ugyanazt a jelölőt szokás alkalmazni. Érdekes megfigyelni a következő példában, hogy a formázás a lista (*ul*) és nem az egyes listaelemek (*li*) tekintetében történik.

<code>ul.disc {list-style-type: disc}</code>	◆ Kávé
<code>ul.circle {list-style-type: circle}</code>	● Tea
<code>ul.square {list-style-type: square}</code>	◇ Kávé
<code>ul.none {list-style-type: none}</code>	◇ Tea
	■ Kávé
	■ Tea
	Kávé
	Tea

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

<code>ol.decimal {list-style-type: decimal}</code>	1. Kávé
<code>ol.lroman {list-style-type: lower-roman}</code>	2. Tea
<code>ol.uroman {list-style-type: upper-roman}</code>	
<code>ol.lalpha {list-style-type: lower-alpha}</code>	i. Kávé
<code>ol.ualpha {list-style-type: upper-alpha}</code>	ii. Tea
	I. Kávé
	II. Tea
	a. Kávé
	b. Tea
	A. Kávé
	B. Tea

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

<code>list-style-image: url('arrow.gif')</code>	▶ Kávé
	▶ Tea

A listajelölő pozíciója

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

<code>ul.inside { list-style-position: inside }</code>	◆ Egyik listaelem ◆ Másik, hosszabb listaelem
<code>ul.outside { list-style-position: outside }</code>	◆ 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')
```

5.9. Méretek

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

Több felhasználási lehetőség közül talán a bekezdések formázását érdemes megemlíteni: bekezdések esetén a sorok *height* tulajdonságával befolyásolható a sorok közötti távolság.

5.10. Megjelenítés

A megjelenítés módja

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

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}                               Első bekezdés. Második bekezdés.  
| div {display: none}
```



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

5.10.1 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.
```

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.
```



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

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

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



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 trükkkel 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;
}
```

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



```
<body>
  <div>
    <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 egérkurzort 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 érhetőek el. Érdeemes 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 menüön kívüli háttérszín és a jobb oldali szegély színe itt meg kell, hogy egyezzen.

```

a:hover { background-color:#ff3300}

```

Az egérkurzorra színváltással reagál a menüpont. Érdeemes 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 bonyolalmat 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. pozícionálásához 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 dizájn 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). A következőt szeretnénk elérni:

Praesent lorem ispum	
Phasellus wisi nulla, bibendum in, eleifend vitae, mollis tinci dunt, tortor lorem ispum lorem ispum. Pellentesque tempus. Lorem ipsum dolor sit amet.	<h3>Aenean nummy odio orci</h3> <p>Phasellus wisi nulla, bibendum in, eleifend vitae, mollis tinci dunt, tortor lorem ispum lorem ispum. Pellentesque tempus. Lorem ipsum dolor sit amet.</p> <p>Adipiscing elit praesent ut mi ut nibh dignissim pretium.</p>
Praesent lorem ispum	

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. Érdeemes 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.

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 megszkott 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 háttérét kell beállítanunk.

5.10.2 Pozícionálási sémák

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

Relatív pozíció

Az elemeket alapértelmezett (*static*) helyzetüktől el tudjuk mozgatni a relatív pozícionálás 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 pozícionálás lehetőségeit. A megoldás lényege, hogy a cím szövege két példányban szerepel a HTML forrásban, de vizuálisan majdnem egymást elfedve, és más színnel. Nézzük a példát:

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

Főcím

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

Bekezdés. Bekezdés. Bekezdés.
 Bekezdés. Bekezdés. Bekezdés.
 Bekezdés. Bekezdés. Bekezdés.
 Bekezdés. Bekezdés.

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 pozícionált 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 pozícionálással 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;
}
```

Főcím

Bekezdés. Bekezdés.
Bekezdés...

Fix pozíció

A fix pozíció az előző speciális esete. Az elem itt is kikerül a pozícionálási sémából, de itt nem a tartalmazott doboz, hanem a látótér (képernyő) a viszonyítási pont.

Nagyszerűen alkalmazható, pl. a hagyományos keretek (*frame*-k) kiváltására.

5.10.3 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 szkript 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 megemlíteni a *Wufoo* ide vonatkozó demóját³¹.

³¹ <http://wufoo.com/demo/>

5.10.4 Z-index

Legtöbb esetben az elemek nem takarják egymást. Ha mégis, akkor alapértelmezetten a későbbi elem takarja el a korábbi. Ha ez nem megfelelő, akkor a *z-index* értékek meghatározásával manipulálhatjuk a vizuális takarást.

5.11. 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.

5.11.1 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 egérkurzot alatt levő és a klikkelés 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.

5.11.2 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>Nézzze meg a
  <a href="http://www.gamf.hu">GAMF</a>
  és az
  <a href="http://informatika.gamf.hu">
    Informatika tanszék</a>
  honlapját!</p>
```

Nézzze meg a [GAMF](http://www.gamf.hu) és az [Informatika tanszék](http://informatika.gamf.hu) honlapját!

Ha az adott elemben (itt *p*) nem link (*a*) az első 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>Nézzze meg</strong> a
  <a href="http://www.gamf.hu">GAMF</a>
  ...
```

Nézzze meg a [GAMF](http://www.gamf.hu) és az [Informatika tanszék](http://informatika.gamf.hu) honlapját!

5.11.3 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;
}
```

5.12. Média típusok

Ha már egy weboldalt az eredetihez képest több féle környezetben használhatjuk. Egy weboldalt megnézhetünk képernyőn, kézisámítógépen, 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 álmányokat 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 módszer)
screen	képernyő

5.13. 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*

5.14. Validátor

A CSS kód ellenőrzésére az online is használható W3C CSS Validation Service³² egy nagyon jó eszköz. Arra azért figyelni kell, hogy csak érvényes HTML kóddal érdemes a CSS érvényességét vizsgálni.

³² <http://jigsaw.w3.org/css-validator/>

5.15. 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.

6. JavaScript

A JavaScript ma az egyetlen jelentős kliens oldali szkriptnyelv. Weboldalak milliói használják űrlapok helyes kitöltésének ellenőrzéséhez, sűtik 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.

6.1. 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 nics szoros kapcsolatba a Java nyelvvel
- C++ és Java szintaxisra alapoz
- Ma már a szabványosított ECMAScript specifikációt valósítja meg

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 kódoktól eltekintve nem szokás 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:


```
<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 szükséges:

```
| <script type="text/javascript" src="message.js"></script>
```

6.1.1 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 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 változóknak** 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 foglalkotnunk 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).

6.1.2 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 aktíválni. A *break* pedig azért szükséges, hogy a programunk ne tudjon a másik esetre rácsorogni.

A következő példa 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>

```

6.1.3 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
```

Feltételes operátor

A háromoperandusú feltételes operátor egyszerű esetekben egy *if-else* utasítást is kiválthat. Szintaxis:

```
változónév=(feltétel)?kifejezésaigaz:kifejezésahamis
```

A *feltétel* teljesülése esetén *változónév* értéke *kifejezésaigaz* értéke lesz, különben pedig *kifejezésahamis* 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>
```

6.1.4 Dialógusablakok

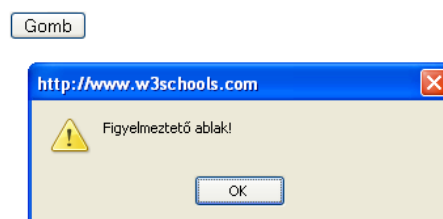
Időnként szükségünk lehet egy honlapon arra, hogy a felhasználó figyelmét ráirányítsuk valamilyen fontos információra egy dialógusablak segítségével. Ezek a dialógusablakok típusuktól függően más-más függvénnyel érhetők el.

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>
```

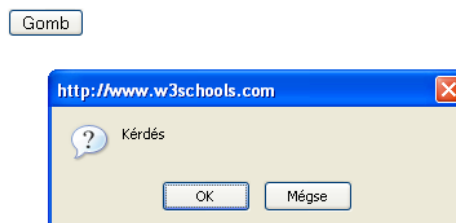


Kérdés

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>
```

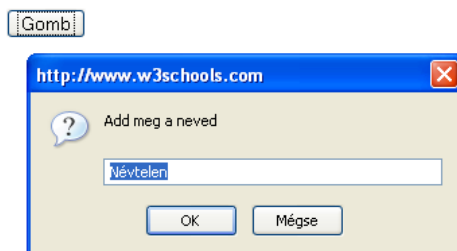


```
| </body>
```

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>
```



6.1.5 Függvények

A függvények az újrafelhasználható kódolás egyszerű eszközei. Egy egyszer megírt függvényt a kód különböző helyein tetszőlegesen sokszor meghívhatjuk. 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.

6.1.6 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 adig 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) {
    ciklusmag
}
```


A *do-while* ciklus logikája csak abban más, hogy mindenképpen lefut egyszer a ciklusmag, és csak ez után értékelődik ki a feltétel:

```
do {
  ciklusmag
} 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);
```

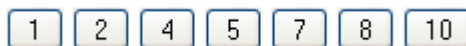
Vezérlésátadás

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>
```

6.1.7 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:

- egérekattintás
- 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 amikor 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 begépelte adat, és véletlen klickelek egy linkre, elveszhet a begépelte 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 klikkelés vagy 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 klikkelve 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>
```

Dologozok
Munkahely:

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 vázolata:

```




<br>


```

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 cserélni. (CSS-el is megoldható a csere, másrészt az aktivitást jelző kép lassú kapcsolat esetén nem jelenik meg azonnal, ami nagyon zavaró.)

6.1.8 Kivételkezelés

A szkriptjeink futása közben előfordulnak különböző hibák, amelyeket kezelniük kell. A látogatók se a felbukkanó (számukra) értelmetlen hibaiüzeneteket, 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 elgépeztük: *alert* helyett *adlert*-et gépeztünk. 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 {
      alert("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!")
}
```

6.2. Objektorientált programozás

A JavaScript objektorientált nyelv. Mielőtt azonban a részletekbe mennénk, érdemes megjegyezni, hogy a C++, Java, PHP stb. nyelvekkel szemben itt valódi objektorientáltságról 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 képest, 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 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() {
  this.tulajdonsag = 5;
}
var objektum = new osztaly();
alert(objektum.tulajdonsag);
```

A kód eredménye **5**.

További forrás

A téma háttéréhez Hodicska Gergely *Objektum orientált JavaScript programozás a felszín alatt*³³ című cikkét javasoljuk.

6.2.1 Fontosabb objektumok röviden

Ez a fejezet csupán néhány alapvető objektum felületes bemutatására vállalkozhat, elsősorban példákon keresztül. Részletes referencia található pl. a HTMLInfo³⁴ oldalon.

³³ <http://weblabor.hu/cikkek/oojsafelszinalatt>

³⁴ <http://htmlinfo.polyhistor.hu/js13ref/contents.htm>

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 paraméternélküli hívással tudjuk megtenni.

```
| document.write(Date())
```

Eredmény:

```
| Fri Aug 18 2006 15:15:11 GMT+0200
```

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 />")  
| }
```

Érdeemes 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.

6.3. 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 klikkelés hatására sárga háttérrel állít be:

```
<head>
  <script type="text/javascript">
    function ChangeColor() {
      document.body.backgroundColor="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.backgroundColor* az oldal (*body*) háttérszínét képviseli. Az objektum eseménykezelőit hasonló egyszerűséggel használhatjuk: az *onclick="ChangeColor()"*-hoz hasonlóan.

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.backgroundColor="yellow"
```

A fejezet hátralévő részében különböző objektumok fontosabb szolgáltatásait (metódusait) fogjuk áttekinteni példákon keresztül.

6.3.1 *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 fókuszot a *focus* metódus meghívásával, vagy éppen elvesztheti a *blur* metódus hatására. A *focus* praktikus lehet, ha a JavaScriptet űrlap adatok ellenőrzésére használjuk. Ekkor a hibüzenet megjelenítése után vissza lehet küldeni a kurzort, a hibát tartalmazó beviteli mezőre, így egy klikkelést 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.

6.3.2 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>
```

Érdeemes 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 keressük meg (*document.getElementById('azon')*), akkor ez soha nem fog nehézséget okozni. Másrészt a kódot később olvasni is egyszerűbb ezzel a megközelítéssel.

6.3.3 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.

6.4. Diszkrét JavaScript³⁵

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 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.

6.4.1 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.

³⁵ Nagyon jó cikk a témához: <http://weblabor.hu/cikkek/diszkretjavascript>

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:

```
| 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 gyűjti 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 klikkeléskor 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.)

6.4.2 E-mail cím elrejtése³⁶

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 lényege, 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 visszacsereéljük 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 problémái miatt fenálló 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.

³⁶ A témáról részletesebben: <http://weblabor.hu/cikkek/what>

```
    replaces++;  
  }  
}
```

Végül a csere elindítása:

```
addEvent(window, 'load', whatCorrector);
```

6.5. Gyakran használt függvények³⁷

Ebben a fejezetben olyan függvények következnek, amelyek sokféle feladat esetén alkalmazhatók.

6.5.1 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:

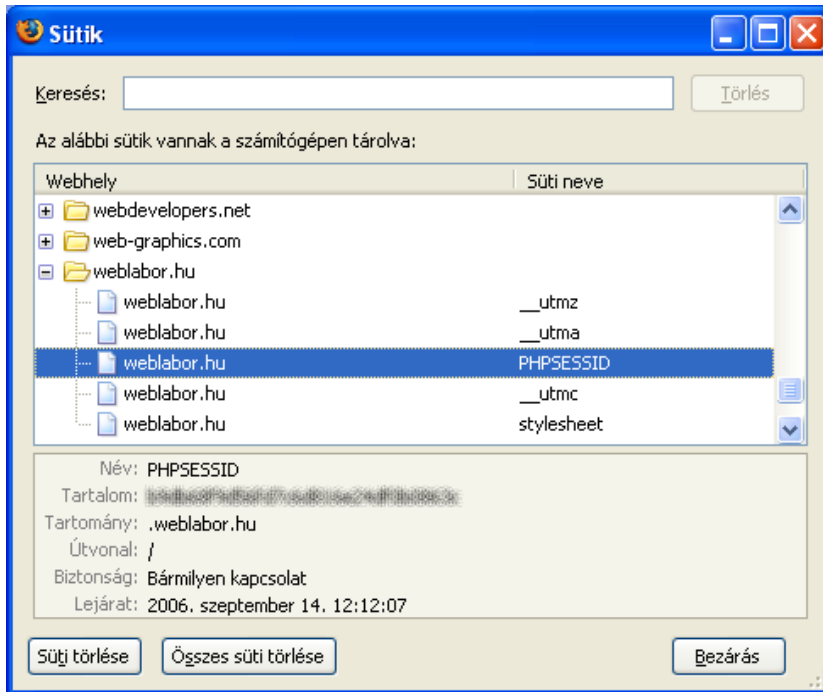
```
HTTP/1.x 200 OK  
Server: Apache/2.4.12 (Unix) Debian GNU/Linux PHP/6.0.0  
Last-Modified: Sun, 25 Apr 2006 13:57:03 GMT  
Set-Cookie: sessionId=krixkrax  
Content-Length: 177  
Content-Type: text/html; charset=iso-8859-2  
  
<html>  
  <head>  
    <title>Cikkek</title>  
  </head>  
  <body>  
    <a href="cikk.php?id=1">1. cikk</a>  
    <a href="cikk.php?id=2">2. cikk</a>  
  </body>  
</html>
```

Minden további kérés:

³⁷ Forrás: pl. <http://www.dustindiaz.com/top-ten-javascript/>

```
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 kisírozva látszik. Ennek megszerzése ugyanis – hozzáértő kezekben – a szerveren levő „személyiséggel” kapcsolatos visszaélésekre adnak lehetőséget³⁸.

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 szó-kás 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 darbolja fel a sztringet sütikre.

³⁸ 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 sűtik közül a keresettet, és visszaadja az értékét.

6.5.2 *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 megszámlálásához 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);
}

```

6.5.3 *addEventListener*

Ez a függvény egy adott böngésző objektum egy adott eseményéhez csatol hozzá egy adott függvényt, melyeket paraméterül 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.)

6.5.4 *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.

6.6. Kódtárak

Az elmúlt néhány év a JavaScript nagykorúvá válásáról szólt. Elkészültek olyan nyílt forrású kódtárak, amelyek a programozók számára többé-kevésbé jól dokumentáltak, és komoly funkcionalitást (más megközelítésben felhasználói élményt) nyújtanak a felhasználónak.

Megjegyzés: A Javascript első néhány évében rengeteg olyan kódot készítettek, ami mai felhasználói szemmel nézve inkább hátráltatja az oldal használatát, vagy éppen van jobb megoldás is a használatánál (jellemzően a CSS váltott ki sok korábbi „effektet”, például a *:hover* látszólagos kiválasztóval).

A JavaScript komoly felhasználását sokáig az is hátráltatta, hogy a kódot (minden ellenkező irányú törekvés ellenére) viszonylag egyszerűen „el lehet lopni”. Így a JavaScript fejlesztésére fordított idő nem igazán gazdaságos anyagi oldalról.

prototype.js

A *prototype.js*³⁹ egy nagyon jól használható JavaScript keretrendszer, mely főként AJAX/Web 2.0 fejlesztésekhez készült (számos program/rutingyűjtemény épít rá), de egyéb környezetekben is vannak nagyon jól használható részei. Gazdag felhasználói élményt nyújtó alkalmazások készíthetők a segítségével.

A rendszer függvényeinek, osztályainak fejlesztői leírása magyarul is olvasható⁴⁰.

Yahoo! User Interface Library

A Yahoo fejlesztői honlapján⁴¹ nagyon sok hasznos kódot bocsájt a rendelkezésünkre. A rutinkönyvtár animációk, XMLHttpRequest, DOM manipulációs, drag&drop és ese-

³⁹ <http://prototype.conio.net/>

⁴⁰ <http://prototype.rajmi.hu/prototype.js.hu.html>

⁴¹ <http://developer.yahoo.com/yui/>

ménykezeléssel kapcsolatos lehetőségeket kínál, illetve naptár, csúszka és faszerű lista komponenseket is közlétesz.

MooTools

A MooTools⁴² egy olyan platformfüggetlen, moduláris, objektumorientált keretrendszer, amely gyors és egyszerű lehetőséget ad látvány-effektusok alkalmazására.

6.7. Alkalmazások

Manapság a kódtárakon kívül nagyszerű alkalmazások is várnak arra, hogy az oldalainkon felhasználjuk. Következzen itt egy példa, amelyet a szerző előszeretettel alkalmaz.

Megjegyzés: Ehhez a témához nincs értelme képet mutatni, érdemes inkább online megtekinteni.

Lightbox

A Lightbox JS v2.0⁴³ a Prototype Framework és a Scriptaculous Effects Library⁴⁴ felhasználásával egy igen látványos, diszkrét JavaScriptet alkalmazó fényképalbumot bocsát a rendelkezésre. A Rövid telepítési útmutató segítségével 10 perc alatt hasonlót varázsolhatunk egy honlapra.

Google Analytics

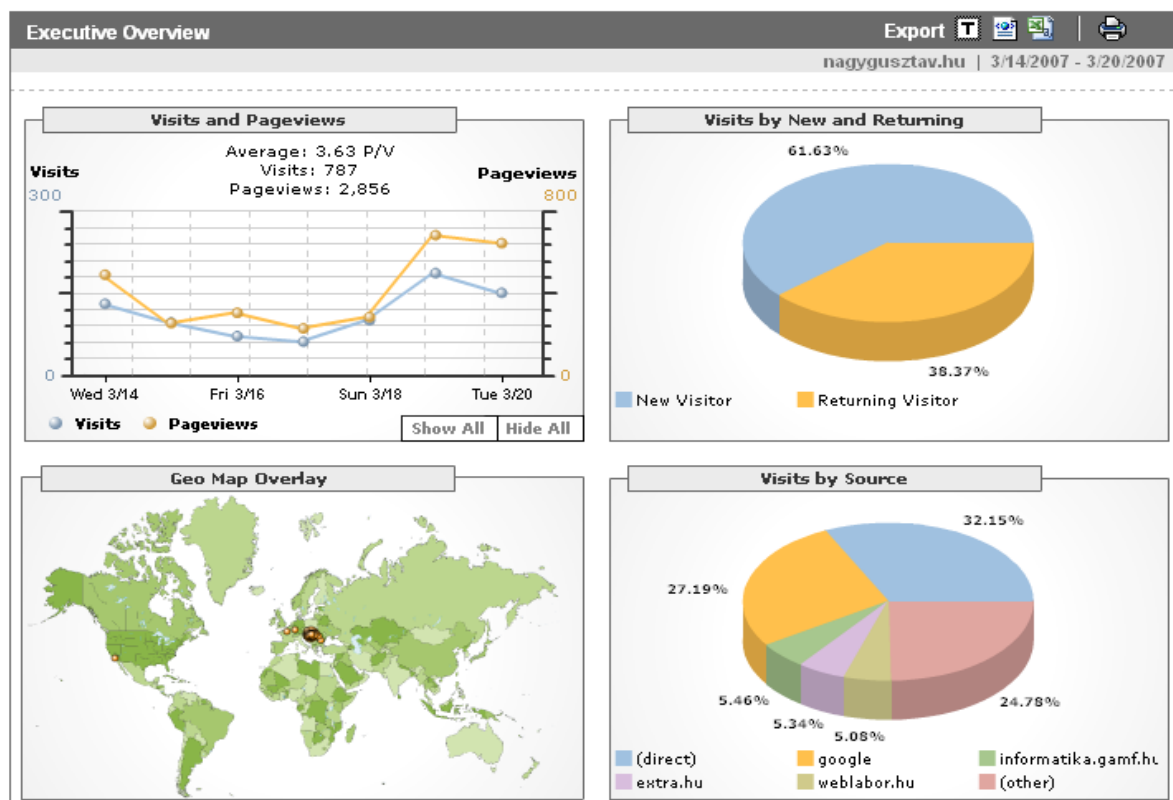
A Google Analytics⁴⁵ szolgáltatása nemcsak egy komplex statisztikai rendszert nyújt, hanem a honlappal kapcsolatos marketing célok elérését is segíti. A következő ábrán csupán az alapvető statisztikai információkat tudjuk megtekinteni:

⁴² <http://mootools.net/>

⁴³ <http://www.huddletogether.com/projects/lightbox2/>

⁴⁴ <http://script.aculo.us/>

⁴⁵ <http://www.google.com/analytics/index.html>



6.8. 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 dolgát megkönnyíteni.

6.8.1 Kliens oldali űrlap ellenőrzés

Felhasználóként rendkívül kiábrándító tud lenni, ha egy komolyabb űrlap kitöltése 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 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:

Új hozzászólás

(kötelező, minimum 10 betű):

Név (kötelező, minimum 3 betű):

E-mail cím:

(Az e-mail cím megadása nem kötelező. Megadása esetén megjelenik az oldalon, de nem lesz kiszolgáltatta a spam robotok számára.)

Humán ellenőrzés. **Negyvenhét** számmal:

Küldés

Az ábrán nem 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 kiszolgáltatta 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>
```

Érdeemes 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:

```
addEventListener(document.getElementById('szoveg'),
  'keypress', urlapEllenoriz);
addEventListener(document.getElementById('nev'),
  'keypress', urlapEllenoriz);
addEventListener(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);
}
```

6.8.2 Hosszú listák böngészése helyett

A GAMF-on működő BME Nyelvvizsgahely⁴⁶ 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	lb idő	lb óra	lb hely	Magnó idő	Magnó óra	Magnó hely
2	Házaudató és Történet Történet	Angol	Alap	A				2007.03.30	14:00	GAMF Főép.4'
3	Ház... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
4	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
5	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
6	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
7	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
8	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
9	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
10	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
11	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
12	Órálat... ..	Angol	Alap	B	2007.03.30	15:00	GAMF Főép.411			
13	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
14	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
15	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
16	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
17	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
18	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
19	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'
20	Órálat... ..	Angol	Alap	C	2007.03.30	15:00	GAMF Főép.411	2007.03.30	14:00	GAMF Főép.4'

Néhány hagyományos megoldási lehetőség:

⁴⁶ <http://lekt.gamf.hu/BME/>

- valamilyen offline médium használata (pl. egy Excel táblázat), ebben történik a szervezési információk előállítása és publikálása is
- 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:

Írja be a neve első betűit, majd válassza ki a listáról.

Név:

Kovács András

Kovács Györgyi

Kovács László Imre

Kovács-Nüszl Botond

Nyelv: **angol**

Szint: **alapfok**

Írásbeli vizsga

Időpont: **2006. november 10. 12:00**

Hely: **GAMF Főépület 411.**

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 megvizsgálánk, 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="' + 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>

6.9. 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 szöveggként 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 választó⁴⁷ 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.

6.10. 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>)

⁴⁷ <http://weblabor.hu/cikkek/betumeretvalaszto>

7. AJAX

Jesse James Garrett 2005 elején publikált cikkében⁴⁸ alkalmazta először az AJAX kifejezést. Az azóta eltelt időben nem volt más, ami ennyire felpezsdítette volna a webes szakmai közösséget.

Az AJAX nem egy új nyelv, hanem a létező szabványok (bizonyos értelemben) újszerű alkalmazása. A felhasználók számára jobb, gyorsabb és barátságosabb webes alkalmazások hozhatók létre a segítségével.

Az AJAX az *Asynchronous JavaScript And XML* kifejezés rövidítéseként jött létre. Lényege, hogy az AJAX alkalmazás az oldal letöltődése után további kéréseket küld a szerver felé a HTTP protokollon keresztül. A kérések küldése és fogadása a JavaScript felhasználásával és aszinkron módon történik. Míg a hagyományos navigáció esetén az egész oldal minden esetben újratöltődik, az AJAX alkalmazásával az adatcsere a minimumra csökkenthető.

Megjegyzés: Ezen a ponton érdemes kiemelni, hogy nem az adatmennyiség csökkentése az AJAX célja, ez inkább csak egy „mellékhatásként” jelentkezik. Tévedés tehát azt hinni, hogy a régi keretes oldalak helyett az AJAX oldja meg a menük újratöltésének problémakörét.

A webfejlesztők régóta kísérleteznek azzal, hogyan lehetne az asztali alkalmazásokhoz hasonló, rugalmasan használható webes felületeket készíteni. Az AJAX segítségével könnyedén lehet ilyen „gazdag” alkalmazásokat készíteni.

Nézzünk meg egy nagyon egyszerű helyzetet. A látogató regisztrálni szeretne, és ezért felhasználói nevet választ magának. Hagyományos JavaScript módszerrel itt annyi kényelmi szolgáltatást adhatunk, hogy a szintaxis ellenőrzést elvégezhetjük (például a minimum hosszúság megvan-e, és csak megengedett karaktereket adott-e meg). Azt azonban, hogy a név már foglalt, csak az űrlap elküldése után, vagy pedig AJAX segítségével akár azonnal jelezhető.

AJAX kommunikáció

A hagyományos webalkalmazások a felhasználó által bevitt adatokat egy űrlap formájában küldik el a kiszolgálónak. Miután az feldolgozta az adatokat, egy teljesen új oldalt küld vissza a böngészőnek. Mivel minden felhasználói adatbevitel után a kiszolgáló új oldalt küld, ezek az alkalmazások általában lassúak és kevésbé felhasználóbarátok.

AJAX-szal azonban a weboldalak úgy küldhetnek és fogadhatnak adatokat a kiszolgálónak, hogy nem szükséges az egész oldalt újra letölteni. Ez úgy lehetséges, hogy az oldal a háttérben küld egy HTTP kérést a kiszolgáló felé, majd JavaScript segítségével csak a weboldal egy részét módosítjuk, ha megérkezik a válasz.

Habár bármilyen formátumot használhatunk az adatok küldésére/fogadására (akár egyszerű szöveget is), legelterjedtebb az XML használata.

AJAX szerver nem létezik. Az AJAX egy technológia, ami a böngészőn fut. Aszinkron adatkapcsolat segítségével kér a kiszolgálótól éppen annyi információt amennyire szükség van, ahelyett, hogy az egész oldalt töltenénk le újra.

⁴⁸ <http://www.adaptivepath.com/publications/essays/archives/000385.php>

7.1. Bevezető példa

A következő példa azt mutatja be, hogyan tud egy weboldal kommunikálni a szerverrel az oldal teljes újratöltése nélkül. Nézzünk egy egyszerű szöveges beviteli mezőt, amelyik egy név begépelését teszi lehetővé.

HTML

A kezdetben üres javaslatlista a *txtHint span* mezőbe fog kerülni, az éppen begépelteknek megfelelően. (Például az „A” begépelésére megjelenik az összes „A” betűvel kezdődő név.)

```
<form>
  Keresztnév:
  <input type="text" id="txt1" onkeyup="showHint(this.value)">
</form>
<p>Javaslat: <span id="txtHint"></span></p>
```

showHint

Minden egyes billentyűleütés (pontosabban felengedés: *onkeyup*) esetén lefut a *showHint* JavaScript függvény. Érdekes megfigyelni az átadott paramétert: a *this.value* az aktuális *input* elem (*this*) begépeltek értékét (*value*). A függvény néhány kötelező ellenőrzéssel kezd, a lényegi kód a függvény végén található:

```
function showHint(str) {
  if (str.length==0){
    document.getElementById("txtHint").innerHTML=""
    return
  }
  xmlhttp=GetXmlHttpRequest()
  if (xmlhttp==null) {
    alert ("Browser does not support HTTP Request")
    return
  }
  var url="gethint.php"
  url=url+"?q="+str
  xmlhttp.onreadystatechange=stateChanged
  xmlhttp.open("GET",url,true)
  xmlhttp.send(null)
}
```

Ha a beviteli mező üres (*str.length==0*), akkor nincs javaslat (*innerHTML=""*).

Megjegyzés: ha ilyenkor a teljes listát kiadná, akkor a lényeg veszne el. Egy ilyen helyzetben pont az az AJAX megoldás előnye, hogy nem kell előre letölteni a több ezer választási lehetőséget egy *select* tagba, hanem minden esetben csak egy jóval szűkebb találati listát.

Következő lépés az AJAX működés lelkét adó (példánkban *xmlHttp*) objektum létrehozása. (Mivel ez a lépés böngészőfüggő, később részletesen visszatérünk erre.)

Ha létrejött a kommunikációs objektum, akkor az URL összeállítása következik. Itt egyenlőre annyi lényeges, hogy a megszólításra kerülő *gethint.php* a paraméterek alapján tud arról, hogy itt mi a kérés: az „A” betű begépelése esetén az url a következő lesz:

```
| gethint.php?q=A
```

Természetesen itt a tényleges kódolás előtt az összes lehetséges helyzet alapján rögzíteni kell valamilyen paraméterátadási módszert, és ehhez kell alkalmazkodni a kérés kliens

oldali összeállításakor és szerver oldali feldolgozásakor is. (Ezt úgy is mondhatjuk, hogy a kliens és szerver közti kommunikációra egy interfészt vagy protokollt dolgozunk ki.) A példában jól látszik, hogy a szerver oldalon a *q* paraméter értéke alapján kell a javaslatokat összeállítani.

Megjegyzés: Itt még nem témánk a szerver oldal tárgyalása, de könnyen látható, hogy például egy adatbázis táblában tárolt nevek esetén egy egyszerű, a következőhöz hasonló lekérdezésre lesz szükség:

```
| SELECT nev FROM nevek WHERE nev like "A%"
```

Az utolsó három sor kicsit több magyarázatot igényel. Az AJAX aszinkron működése azt jelenti, hogy a szerver felé küldött kérésre a JavaScript nem várja (nem is várhatja) meg a választ, hanem a felhasználót hagyja tovább dolgozni (esetünkben gépelni). A kommunikációs objektumunk számára tehát egy ún. *callback* (visszahívható) függvény nevét adja meg az *xmlHttpRequest.onreadystatechange=stateChanged* sor. Amikor a szervertől visszaérkezik a válasz, akkor a JavaScript kódunk erről úgy fog értesülni, hogy a *stateChanged* függvény kapja meg a vezérlést. Másként fogalmazva: a *stateChanged* függvényt kell képessé tennünk arra, hogy a visszaérkező választ feldolgozza.

Az *xmlHttpRequest.open* függvény a kérést írja le: *GET* metódussal szeretnénk a kérést küldeni az *url* számára, a *true* paraméter pedig az aszinkronitást írja elő.

Megjegyzés: Aszinkronitás nélkül a böngésző lemerevedne a válasz megérkezéséig. Ez (különösen lassabb hálózati kapcsolat esetén) nem a felhasználói élményt növelné, hanem inkább korlátozná a munkát.

Végül az *xmlHttpRequest.send* függvény végzi a kérés tényleges elküldését.

stateChanged

```
| function stateChanged() {
|     if (xmlHttpRequest.readyState==4 || xmlHttpRequest.readyState=="complete"){
|         document.getElementById("txtHint").innerHTML=
|             xmlHttpRequest.responseText
|     }
| }
```

A függvény többször is meghívódik a válasz teljes megérkezéséig, de az esetek túlnyomó többségében elég akkor foglalkozni vele, amikor a teljes válasz megérkezett (*readyState==4*).

Ahogy eddig is látszott, ez a példa az alapok rövid áttekintését tűzi ki célul, és nem a teljes és komplex megoldást. A válasz esetén is látható, hogy itt nem XML válasz érkezik a szervertől, hanem szöveges (akár HTML). Egyszerűbb esetekben ez tökéletes megoldás lehet.

7.2. A böngészők AJAX támogatása

Az XMLHttpRequest objektum nélkül nincs AJAX.

Sajnos a böngészők nem teljesen egységesen kezelik az (egyébként szabványban⁴⁹ rögzített) XMLHttpRequest objektumot. Ez azonban nem olyan nagy probléma, a következőhöz hasonló kóddal a probléma jól kézben tartható:

⁴⁹ <http://www.w3.org/TR/XMLHttpRequest/>

```
function GetXmlHttpRequest(handler) {
    var objXMLHttp=null
    if (window.XMLHttpRequest) {
        objXMLHttp=new XMLHttpRequest()
    } else if (window.ActiveXObject) {
        objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
    }
    return objXMLHttp
}
```

A függvény először a Mozilla alapú böngészőkön alkalmazható megoldással kezd, majd Windows ActiveX vezérlőt próbál létrehozni. A kódból az is következik, hogy az AJAX használatához a JavaScripten túl az ActiveX engedélyezésére is szükség van a böngészőben.

7.3. XML-kommunikáció

Az előző példa egyszerű szöveges kommunikációt valósított meg. Összetettebb esetekben mindenképpen XML alapú kommunikáció szükséges .

7.4. További források

- HTMLInfo: AJAX
(<http://htmlinfo.polyhistor.hu/ajax/index.html>)
- ajaxian
(<http://ajaxian.com/>)

8. PHP nyelv és környezet

A PHP (**PHP: Hypertext Preprocessor**) 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 webserverral 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 webservice 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 webservice 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 – helytakarékosági 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>
```

Minden PHP utasításnak ;-vel kell végződnie. Két alapvető 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 változókat). 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 ($lehetozza || $tartalom) { ?>
  <h2><a href="<?=$index ?><?=$blognev ?>/<?=$url ?>">
    <?=$hirCim ?></a></h2>
<? } else { ?>
  <h2><?=$hirCim ?></h2>
<? }?>
```

8.1.2 Változók

A PHP nyelvben minden változó \$ 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 szöveg között nem tartalmazhatnak, így az összetett változónevek esetén az aláhúzás karaktert (`$my_string`), vagy a közbülső szó nagy kezdőbetűjét (`$myString`) alkalmazzuk.

8.1.3 Operátorok

Az operátorok műveleteket végeznek az operandusaikkal.

A fejezetben egy gyors összefoglaló következik az operátorokról.

Aritmetikai operátorok

Megjegyzés: A táblázat példái nem tartalmazzak `;-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 maradék	$5\%2$ $10\%8$ $10\%2$	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

8.1.4 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 mellett, a sorokat kapcsos zárójelek közé kell írunk:

```
<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>
```

8.1.5 A *switch* szerkezet

A *switch* utasítást a PHP-ben akkor használjuk, hogyha más-más kódot szeretnénk végrehajtani különböző feltételek alapján. A *switch* használatával 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 végrehajtottik
- 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.

8.1.6 Tömbök

A tömb egy vagy több értéket tárolhat egyetlen változó névvel.

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 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 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.

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:

```
<?php
$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 mellel 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 oldalán a honlap szerkezete a következő tömb alapján generálódik:

```
$oldalak = array (
  'Személyes' => array(
    'elérhet' => array(
      'cim' => 'Elérhetőségek',
      'tipus' => 'statikus'
    ),
    'magamrol' => array(
      'cim' => 'Magamról',
      'tipus' => 'statikus'
    )
  )
  // ...
),
```

```

    'Oktatás' => array(
        'java' => array(
            'cim' => 'Java programozás',
            'tipus' => 'blog',
            'felirat' => 'Tantárgyi információk',
        ),
        // ...
    ),
    // ...
);

```

8.1.7 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, ami a ciklust vezérli. 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 egygyel 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>

```

8.1.8 Függvények

A PHP igazi ereje a függvényeiből ered. A PHP-ben több mint 700 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 keresztnéveket 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 oldalán a következő függvény segítségével kereshetők meg az egy hírhez (blog-bejegyzéshez) 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);
}
```

8.1.9 Úrlapok 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.

8.1.10 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 akadályja 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.

8.1.11 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

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 hibüzenetek 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 osztálydefiniáció 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/Írás. 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 tartalmát és csak a végére ír.
x	Csak írás. Készít egy új fájlt. Hibát jelez (<i>false</i>) ha már létezik.
x+	Olvasás/Írás. Készít egy új fájlt. Hibát jelez (<i>false</i>) 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.

```
<?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 lehetőség szerint.

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:

<code>\$_FILES["file"]["name"]</code>	a feltöltött fájl neve
<code>\$_FILES["file"]["type"]</code>	a feltöltött fájl típusa
<code>\$_FILES["file"]["size"]</code>	a feltöltött fájl mérete byte-okban
<code>\$_FILES["file"]["tmp_name"]</code>	a szerveren tárolt fájl másolatának átmeneti neve
<code>\$_FILES["file"]["error"]</code>	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 fájl méretnek 20 Kb alatt kell maradnia:

```

<?php
    if ((($_FILES["file"]["type"] == "image/gif")
        || ($_FILES["file"]["type"] == "image/jpeg"))
        && ($_FILES["file"]["size"] < 20000)) {
        if ($_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/jpeg")
        && ($FILES["file"]["size"] < 20000)) {
        if ($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 sütitet (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. fejléct küld, ennek a fejlécnek lesz része a süti 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 fejléct.

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 lejáratnak állítjuk be.

Példa törlésre (az érvényesség lejáratának 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 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 változóban 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 egyel. 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 szűrni 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-mailbefecskendezéses 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.

Azonban 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 opcionálisok: fájl, a sor száma és a hibakörnyezete.)

Szintaxis:

```
| error_function(error_level,error_message,
                error_file,error_line,error_context)
```

Paraméter	Leírás
<i>error_level</i>	Kötelező. Megadja, hogy milyen súlyos hibákat tekintünk error-nak. Egy számértéknek kell lennie.
<i>error_message</i>	Kötelező. Megadja a hibaüzenetet.
<i>error_file</i>	Opcionális. A fájlnevet írja le, ahol a hiba felmerült.
<i>error_line</i>	Opcionális. A sor számát határozza meg, ahol a hiba felmerült.
<i>error_context</i>	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 <i>trigger_error()</i> függvényt használva.
512	E_USER_WARNING	Nem végzetes felhasználó által generált figyelmeztetés. Ez olyan, mint egy E_WARNING, melyet a programozó állít be a <i>trigger_error()</i> függvényt használva.
1024	E_USER_NOTICE	Felhasználó által generált értesítés. Ez olyan, mint egy E_NOTICE, melyet a programozó állít be a <i>trigger_error()</i> 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 kiírni, 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 bevétel 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(2) #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 őst. Mivel ez egy leszármazottja az ősoosztálynak, olyan metódusokat is tudunk használni, mint: `getLine()` és `getFile()` vagy `getMessage()`.

A fenti kód egy kivételt dob, és a `CustomException` osztállyal kapja el:

- A `CustomException()` osztály valójában az ősoosztály kiterjesztése. Ez magába foglalja a metódusokat és annak tulajdonságait az ősoosztálynak.
- 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 *CustomException* 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 *CustomException* 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 másodsorra 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 ősz 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 adatbázisszerveret alkalmaznak. Ezért ebben a fejezetben a MySQL használatának alapjait fogjuk áttekinteni.

8.3.1 MySQL alapok

A MySQL a legnépszerűbb 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
Hansen	Ola	Tímoteivn 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 szerveret, amihez kapcsolódni akarunk. Az alapbeállítás a következő: <i>localhost:3306</i>
username	Opcionális. Meghatározza a MySQL felhasználói nevet, amivel bejelentkezünk.
password	Opcionális. Meghatározza a MySQL felhasználó jelszavát . Az alapbeállítás: "" (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 végrehajtsódjon.

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 <i>size</i> paraméterben
decimal(size,d) double(size,d) float(size,d)	Tizedesvesszőt tartalmazó számok. A maximális számjegyek számát meg lehet határozni a <i>size</i> paraméterben A tizedesvesszőtől jobbra lévő számjegyek maximális számát a <i>d</i> 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 beszűrődni.
set	A SET (halmaz) hasonló az ENUM-hoz. Azonban a SET 64 listázott tételt 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:

```

$ssql = "CREATE TABLE person (
    personID int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY(personID),
    FirstName varchar(15),
    LastName varchar(15),
    Age int
)";
mysql_query($ssql,$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* fájlra. Az *insert.php* fájl egy adatbázishoz kapcsolódik, a *mysql_query()* függvény végrehajtja az INSERT INTO parancsot, és az új adat beszállásra 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 sorrendbe

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_név = új_ertek
WHERE oszlop_név = 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 lefelejténék.

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 elemző

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 módosításhoz.

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 szükség lesz egy XML elemzőre (*parser*).

Két alap típusuk van az XML elemzőknek:

- A fa bázisú elemzők: Ez az elemző átalakítja az XML dokumentumotfa struktúrájává. 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");
$xml = $xmlDoc->documentElement;
foreach ($xml->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

SimpleXML egy újdonság a PHP 5-ben. Egyszerű módja annak, hogyan tudod megszerezni egy elem tulajdonságait és leírását, ha ismered az XML dokumentum szerkezetét.

Összehasonlítva a DOM-mal vagy Expat parser-rel, a SimpleXML csupán néhány sor kódot használ az elem adatainak elolvasásához.

SimpleXML az XML dokumentumot egy feladattá alakítja át. Mint például:

- Elemeket alakít át a SimpleXML Element feladat egyetlen sajátosságaivá. Amikor több mint egy elem van egy szinten, akkor sorba veszi azokat.
- Sajátosságokhoz fér hozzá miközben asszociációs sorokat használ, ahol egy index hasonlít a sajátosságnévhez.
- Elem adat elemekből álló Text adatokat sorokká alakítja át. Ha egy elemnek több mint egy szöveg része van, abban a sorrendben helyezi el azokat, ahogy megtalálja.

A SimpleXML gyors és könnyen használható, amikor alapvető feladatokat hajt végre mint például:

- XML fájlok olvasása
- Adatok kivonása az XML sorokból
- Szövegrészek vagy sajátosságok szerkesztése

Mindazonáltal, mikor a haladó XML-el foglalkozunk, mint a névjegyzék, jobb az Expat parsert vagy a XML DOM-ot használni.

Installálás

Csakúgy mint a PHP 5.0, a *SimpleXML* funkciók a PHP mag részét képezik. Nem szükséges installáció ahhoz, hogy használni lehessen ezeket a funkciókat.

A SimpleXML használata

Ki akarjuk választani az elemeket és adatokat az XML fájlból fentről.

1. Töltsd le az XML fájlt
2. Add meg az első elem nevét
3. Készíts egy olyan pontot, ami el fogja indítani az egyes gyermek kapcsolót, használva a gyermek funkciót.
4. Válaszd ki az elem nevét és adatát az egyes gyermek kapcsolathoz.

Példa:

```
<?php
$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!
```

9. PHP megoldások

Ebben a fejezetben az életből vett példákkal szeretnénk bemutatni a szerver oldali alkalmazásfejlesztést.

9.1. Egy portál váza

A szerző és mások tapasztalata is az, hogy a PHP nyelv alapjainak megtanulása viszonylag egyszerű, de már egy kisebb oldal felépítése is komoly feladat lehet a nyelvvel és a webfejlesztéssel ismerkedők számára. Éppen ezért e jegyzetben a téma kiemelt hangsúlyt kap.

Bevezetésként megjegyzendő még, hogy a bemutatott megoldások a szerző saját, gyakorlatban is használt kódjának részletei. Lehetne szebb, teljesebb megoldást is adni a feladatokra, de itt most a logika demonstrálása a célunk.

include és társai

Ahogy azt már korábban, az *include* függvény bemutatásánál láttuk, az oldal változó és fix részeit *include* segítségével szoktuk összefűzni. Az ehhez kapcsolódó példákat érdemes tehát először áttekinteni.

9.1.1 GAMF honlap

A GAMF honlapjának⁵⁰ a szerző által készített második változata kerül bemutatásra.

Tervezés

Az oldal tervezésekor elsődleges szempont volt, hogy az oldal fix és változó része könnyen szétválasztható legyen.

A tervezés egyik fontos lépése, hogy az oldalak szolgáltatás-típusait, az egyes oldalakat nagy vonalakban megtervezzük. Ennek egyik eszköze az URL tervezése. Ahogy a következő példa is mutatja, az oldal URL-jei nem túl szép képet mutatnak (a közeljövőben készülő harmadik változatban csak barátságos URL-ek fognak szerepelni):

- <http://www.gamf.hu/index.php?modul=keprend>
- <http://www.gamf.hu/index.php?hir=kisokos>
- <http://www.gamf.hu/index.php?kep=sigmaringen.jpg>

A *modulok* tulajdonképpen az egyes oldalakat jelentik. Az oldalak többsége modul alapú, vagyis egyetlen állomány *include*-olásával a HTML oldal fix keretébe behúzható a változó tartalom. A *hir* oldalak a kezdőoldalon rövid hírelőzetesként megjelenő hírek teljes szövegét tartalmazzák. A *kep* oldalak egyetlen kép megjelenítését végzik.

Megjegyzés: Programozástechnikailag érdemes megfigyelni, hogy a jelenlegi megoldás nem használ osztályokat, de még függvényeket sem. Ehelyett erősen épít a PHP *include()* függvényére.

⁵⁰ <http://www.gamf.hu>

Fontosabb változók

<i>\$gyoker</i>	a webservertől fájlrendszerén belül hol találhatóak a honlap állományai (sajnos az egyes aldomainek oldalai nincsenek fizikailag szétválasztva, így a GAMF honlap állományait egy <i>gamf2</i> nevű alkönyvtár tartalmazza)
<i>\$modul</i>	a felhasználó által kért modul oldal azonosítója (a korábbi példánkban <i>keprend</i>)
<i>\$engedelyezett_modulok</i>	az engedélyezett modulok listája, az URL paraméter ellenőrzése során lesz szerepe
<i>\$menu</i>	az oldal bal oldali menüszerkezetének generálásához szükséges adatokat tartalmazza, többszörös tömb szerkezet ezen kívül az engedélyezett modulok listájába is szerepelni fognak a főmenü menüpontjai
<i>\$további_modulok</i>	a főmenüben nem szereplő, de a honlap részét képező aloldalak listája

Néhány sor az utolsó két tömbből:

```
$menu = array(
  array (
    "cím" => "Kari információk",
    "osztály" => "kariinfo",
    "linkek" => array (
      array (
        "modul" => "kezd",
        "szöveg" => "Kezdőoldal"
      ),
      array (
        "modul" => "keprend",
        "szöveg" => "Képzési rendszer"
      ),
      ...
    )
  )
);

$további_modulok = array (
  array("hallgvelemeney",
    "Hallgatói vélemények az intézményről"),
  array("meghirdetett",
    "Meghirdetett szakok, szakirányok"), ...
);
```

URL ellenőrzés

Az URL mindig tartalmazhat hibás vagy rosszindulatú adatot, ezért mindig ellenőrizni kell. Példaként nézzük meg a *modul* tartalomtípus ellenőrzését:


```

$modul = $_GET["modul"];
if (!isset($modul))
    header("Location: " . $gyoker_oldal);

```

Ha nincs modul, akkor vissza a kezdőoldalra.

```

else if (file_exists ("$gyoker/modul/$modul.php") == FALSE)
    header("Location: " . $gyoker_oldal);

```

Ha van modul az URL-ben, akkor megnézzük, hogy van-e a kérésnek megfelelő modul-állomány.

```

$engedelyezett_modulok = array();
foreach ($menu as $doboz) {
    foreach ($doboz["linkek"] as $link) {
        if (isset($link["modul"]))
            $engedelyezett_modulok[$link["modul"]] =
                $link["szöveg"];
    }
}
foreach ($tovabbi_modulok as $m) {
    $engedelyezett_modulok[$m[0]] = $m[1];
}

```

A főmenü menüpontjai és a további modulok tömbje alapján előáll az engedélyezett modulok listája.

```

if (!isset($engedelyezett_modulok[$modul]))
    header("Location: " . $gyoker_oldal);

```

Ha ezen is túljut az ellenőrzés, csak akkor tekintjük korrekt modulnak.

Tartalom és megjelenítés szétválasztása

A webfejlesztésben fontos alapelv, hogy az oldal konkrét megjelenése (pl. a HTML kimenet) és az oldal információtartalma (szövegek, képek) egymástól elkülönülten szerepeljenek. Ezzel a megközelítéssel a későbbi változások követése sokkal egyszerűbben megoldható lesz.

Technikai szinten itt a sablon szemléletet kell megismernünk. Ennek lényege, hogy a kimenet elemei és a változó tartalmat előállító PHP kód jól szétválasztva, lehetőleg vizuálisan is elkülönülten szerepeljenek. Ennek fényében a PHP *echo* vagy *print* függvényei helyett a következő módszer javasolt. (A példa kód az oldal főmenüjét gyártja le):

```

        <div id="menu">
<?  foreach ($menu as $doboz) { ?>
        <h2 class="<?=> $doboz["osztály"] ?> menu_cim">
            <?=> $doboz["cím"] ?></h2>
        <ul class="menu_lista">
<?  foreach ($doboz["linkek"] as $link) { ?>
            <li><a class="<?=> $doboz["osztály"] ?>" href="<?
                if (isset($link["modul"])) {
                    echo $_SERVER["PHP_SELF"] .
                        "?modul=" . $link["modul"];
                } elseif (isset($link["külső"])) {
                    echo $link["külső"];
                }
            ?>><?=> $link["szöveg"] ?></a></li>
<?  } ?>
        </ul>
<?  } ?>
    </div>

```

Hírek megjelenítése

Hírek négyféle módon jelenhetnek meg a honlapon.

- A kezdőoldalon aktuális hírként
- A kezdőoldalon archív hírként
- RSS tartalomként
- Teljes szövegű hírként

Jól látszik, hogy nem lenne célravezető a híreket „kézzel” szerkeszteni. Ehelyett a hírek megjelenítéséhez kapcsolódó oldalak két forrásból állnak össze:

- A hírek adatai
- A hírek megjelenítése négyféle módon

A hírek adatai lehetnek adatbázisban, vagy – ahogy jelen esetben is – egy PHP változóban eltárolva. Nézzük először egy hír adatait:

```

<?
$hir = array (
    "cim" => "Karrier Nap a Kecskeméti Főiskolán",
    "csakcim" => "Karrier Nap a Kecskeméti Főiskolán",
    "kezddatum" => "2006.11.24.",
    "vegdatum" => "2006.11.29.",
    "archivalhato" => true,
    "eloszoveg" => "",
    "szoveg" =>
        '<p><a class="kiskep"
            href="gamf2/fenykep/karriernap2006.jpg"
            rel="lightbox"></a></p>',
);
?>

```

Megjegyzés: A tömb adatai egy adatbázis rekordban is lehetnének. Ha *\$hir* változót az adatbázisból a *mysql_fetch_array* függvényt nyertük volna ki, akkor is ugyanezt a tömböt használhatnánk.

A kulcsok közül néhány magyarázatra szorul:

- A *cim* több helyen kerül felhasználásra, egyedül archív hírként nem jelenik meg. Ekkor ugyanis a *csakcim* jelenik meg.
- A hír *kezddatum* és *vegdatum* között tekinthető aktívnek. (Érdemes megfigyelni, hogy a hírek időzítése automatizálható, ezt sem kell „kézzel” vezérelni.)
- A lejárat után egyes hírek még járnak bizonyos információ-értékkel. Az archív hírek között csak az *archivalhato* hírek jelennek meg.
- Az *eloszoveg* csak az első esetben jelenik meg, a *szoveg* helyett.

Most nézzük meg, hogyan lesz az adatokból megjeleníthető tartalom.

Kezdőoldal

Az *\$aktiv_hirek* és az *\$archiv_hirek* tömbök már tartalmazzák a megjelenítendő híreket a megjelenítés sorrendjében. Ekkor a megjelenítendő tartalom így áll elő:

```
<h1>A Kar legfrissebb hírei</h1>
<? foreach ($aktiv_hirek as $hir) {?>
<p><strong><a href="<?=$_SERVER["PHP_SELF"] ?>?hir=
    <?=$hir["fajlnev"] ?>"><?=$hir["csakcim"] ?></a>
    <span class="hirdatum"><?=$hir["kezddatum"] ?></span>
</strong></p>
<?
    if (isset($hir["eloszoveg"]))
        echo $hir["eloszoveg"];
    }
} ?>
<h2>Archív hírek</h2>
<ul class="hirek">
<? foreach ($archiv_hirek as $hir) {?>
<li><a href="<?=$_SERVER["PHP_SELF"] ?>?hir=
    <?=$hir["fajlnev"] ?>"><?=$hir["csakcim"] ?></a>
    <span class="hirdatum"><?=$hir["kezddatum"] ?></span>
</li>
<? } ?>
</ul>
<?} ?>
```

A hírek RSS-ként való megjelenítése ehhez nagyon hasonló. Itt is egy – az előzőtől eltérő – *\$aktiv_hirek* tömb tartalmazza a tíz legfrissebb hírt. A kimenetet előállító kód:

```
<? header("Content-type: text/xml"); ?
    echo '<' . '?xml version="1.0" encoding="iso-8859-2"?>';
?>
<rss version="2.0">
    <channel>
        <title>Kecskeméti Főiskola GAMF Kar</title>
        <link>http://www.gamf.hu</link>
        <description>A GAMF Kar aktuális hírei</description>
        <language>hu</language>
    <? foreach ($aktiv_hirek as $k => $e) { ?>
```

```

    <item>
      <title><?= $e['csakcim'] ?></title>
      <link>http://www.gamf.hu/index.php?hir=<?= $e['fajlnev']
        ?></link>
      <description/>
    </item>
  <? } ?>
</channel>
</rss>

```

A hír teljes szövegének megjelenítése ezeknél jóval egyszerűbb:

```

<? include ($hir_nev); ?>
<h1><?= $hir["cim"] ?></h1>
<p class="hirdatum"><?= $hir["kezddatum"] ?></p>
<?= $hir["eloszoveg"] ?>
<?= $hir["szoveg"] ?>

```

Jól látszik, hogy kezdetleges formában ugyan, de sikeresen szétválasztásra került az oldal fix megjelenése és a változó tartalom.

9.1.2 Egy másik megközelítés

Kollégám honlapja⁵¹ néhány dologban jelentősen eltér a GAMF honlapon alkalmazott megoldástól. Röviden:

- Az oldal csak barátságos URL-t alkalmaz. Mind az emberi fogyasztásra, mind a keresőrobotok (Google) számára hasznosabb, emészthetőbb. Bár ezt az Apache konfigurálásával is elérhetnénk, de itt egy egyedi ötletre épülő megoldás kerül bemutatásra. (A szerzőnek legalábbis nincs tudomása arról, hogy ezt a megoldást más publikálta volna.)
- Az oldal URL és könyvtárszerkezete megegyezik. Minden könyvtárban található egy *index.php* állomány, amely az adott oldal vezérlését látja el, valamint egy *content.tpl.php* állomány, amely az adott URL-en elérhető speciális tartalmat tartalmazza.

Az *index.php* három változót beállít (milyen nyelvű az aktuális oldal, a honlap gyökerének helye (ahol a kezdőoldal található) és az al-oldal URL-azonosítója), majd a konfiguráció és a tartalom *include*-olása következik.

```

<?
  $akt = 'en';
  $gyoker = '../';
  $oldal = 'introduction';

  include "{$gyoker}config.php";
  include "{$gyoker}index.tpl.php";
?>

```

A *config.inc.php* tartalmazza a két nyelvű főmenüt:

⁵¹ <http://foighiv.gamf.hu/toth.akos/>

```

$ny = array (
  'hu' => array (
    'kezd' => '.',
    'szoveg' => 'Magyar változat',
    'kar' => 'iso-8859-2',
    'cim' => 'Tóth Ákos honlapja',
    'menu' => array (
      '.' => 'Bemutatókozás',
      'szakmai_oneletraajz' => 'Szakmai önéletrajz',
      'oktatás' => 'Oktatás',
      'zenei_oneletraajz' => 'Zenei önéletrajz',
    ),
  ),
  'en' => array (
    'kezd' => 'introduction',
    'szoveg' => 'English version',
    'kar' => 'iso-8859-2',
    'cim' => "Ákos Tóth's homepage",
    'menu' => array (
      'introduction' => 'Introduction',
      'cv' => 'Curriculum Vitae',
      'music' => 'Music',
    ),
  ),
);
?>

```

Az *index.tpl.php* tulajdonképpen egy sablon fájl, a lényeges (főmenü és tartalmat generáló része:

```

<div id="header">
  <div class="headtitle"><?= $ny[$sakt]['cim'] ?></div>
  <div class="lang">
    <? foreach ($ny as $k => $e ) {
      if ($k != $sakt) { ?>
        <a href="<?= $e['kezd'] ?>"><?= $e['szoveg'] ?></a>
      }
    } ?>
  </div>
</div>
<div id="menu">
  <ul>
    <? foreach ($ny[$sakt]['menu'] as $k =>$e) {
      if ($k == $oldal) { ?>
        <li>&nbsp;<?= $e ?>&nbsp;</li>
      } else { ?>
        <li><a href="<?= $k ?>"><?= $e ?></a></li>
      }
    } ?>
  </ul>
</div>

```

9.1.3 MVC megközelítés⁵²

Az oldal kialakításakor a Modell – View – Controller tervezési minta került figyelembevételre.

A tervezéskor kiderült, hogy a fix főmenü egyes menüpontjai (pl. *Elérhetőségek*) statikus jellegű tartalmat, vagyis egyetlen fix tartalmú oldalt jelentenek, míg más menüpontok az egy témába bejegyzett blog vagy hír-szerű bejegyzések tartalomjegyzékei.

Az utóbbi esetben figyelembe kell venni, hogy a témához tartozó hírek részleteiben is megtekinthetők. Végül már csak kisebb szolgáltatások, hogy a kezdőoldalra és az RSS oldalra a friss bejegyzéseket témakörtől függetlenül is ki kell gyűjteni.

Adatbázis terv

Az oldal bejegyzései (nagyreszt) egy táblában kerülnek tárolásra. További táblák bevezetése is indokoltnak tűnhet, de a jelenlegi verzió más táblára nem épít. (Pl. a blog-kategóriák nem táblában, hanem egyszerű PHP tömbben szerepelnek.)

Megjegyzés: A szerző saját maga által karbantartott oldalak esetén többször alkalmazza ezt a megközelítést. Előnye, hogy nem kell minden egyes laplekérés esetén ugyanazt a lekérdezést ugyanazért az eredményért végrehajtani. Hátránya viszont, hogy egy később másnak átadott adminisztráció esetén újabb kategóriák nem vehetők fel, csak programozóként.

```
CREATE TABLE `hir` (
  `kod` int(11) NOT NULL auto_increment,
  `blognev` varchar(32) NOT NULL default '',
  `url` varchar(32) NOT NULL default '',
  `datum` date NOT NULL default '0000-00-00',
  `tartalom` tinyint(1) NOT NULL default '0',
  `publikus` tinyint(1) NOT NULL default '1',
  `lehetozza` tinyint(1) NOT NULL default '0',
  PRIMARY KEY (`kod`)
) ENGINE=MyISAM DEFAULT CHARSET=latin2;
```

A menü generálásához használt *\$oldalak* tömb:

```
$oldalak = array (
  'Személyes' => array(
    'elerhet' => array(
      'cim' => 'Elérhetőségek',
      'tipus' => 'statikus'
    ), //...
    'csaladomrol' => array(
      'cim' => 'Családomról',
      'tipus' => 'blog',
      'felirat' => 'Sok-sok fénykép'
    ),
  ), // ...
```

A *\$portal_adatok* tömb az egész honlapra jellemző globális adatokat tartalmazza:

⁵² Az itt bemutatásra kerülő verzió 2007 januárjáig működött éles környezetben a <http://nagyusztav.hu> címen

```
$portal_adatok = array (
    "cimsor" => "Nagy Gusztáv oldala",
    "leiras" => "Nagy Gusztáv oldala",
    "kulcsok" => "Nagy Gusztáv, oktatás, Jézus, Krisztus",
    "szerzo" => "Nagy Gusztáv",
    "gyoker" => "http://nagygusztav.hu/",
    "index" => "index.php",
);
```

index.php

A kezdőoldal nagyon egyszerű feladatot lát el:

```
include_once('adat/config.inc.php');
include_once('lib/Sablon.php');
include_once('lib/Oldal.php');
include_once('lib/Vezerlo.php');

$vezerlo = new Vezerlo($_SERVER["QUERY_STRING"]);
echo $vezerlo->general();
```

Az include-olásokon kívül csak két dolog történik: A vezérlő osztály példánya létrejön, paraméterként megkapva a kérést. Végül a *general* függvény a kimenetet előállítja, amit el is küldünk a kliensnek.

Vezerlo osztály

A *Vezerlo* osztály feladata a kérés (URL, stb.) feldolgozása, a szükséges oldal kiválasztása és betöltése.

Oldal osztályok

Az egyes oldaltípusok (statikus, blog, kezdő és rss) kiszolgálására az *Oldal* ős leszármazottai szolgálnak.

Sablon osztály

A tartalom és a megjelenítés szétválasztására szolgáló osztály. Nem alkalmaz önálló sablonnyelvet, hanem az erre kitűnően megfelelő PHP-t alkalmazza.

Adatbázis osztály

Minden adatbázis-kommunikáció ezen az osztályon keresztül történik. Ez az osztály a MySQL szerverrel való kapcsolatra van felkészítve. Egy esetleges szerver-váltás, vagy adatbázis-újratervezés esetén csak ezt az osztályt kell módosítani, a többi kód változatlan maradhat.

Úgy is fogalmazhatunk, hogy ez az adatbázis-kapcsolatot beburkoló (mások felé elfedő) osztály.

Nézzük akkor részletesebben az egyes osztályok kódjait.

Vezerlo osztály

```
class Vezerlo {
    var $oldal = null;
    var $keres = null;
    var $joOldalak = array();
}
```

Három adattag, amelyeket a konstruktor fog feltölteni. Az *oldal* az *Oldal* osztály valamelyik leszármazottjának példányát fogja tartalmazni. A *keres* az URL-végét fogja tartalmazni, ami alapján a konkrét kérés beazonosítható. Végül a *joOldalak* tömb annak eldöntéséhez szükséges, hogy az aktuális kérés egyáltalán korrekt, kiszolgálható oldalt jelent-e.

```
function Vezerlo($keres = null) {
    $this -> keres = $keres;

    $this -> joOldalak[$GLOBALS['kezdooldal']] = array (
        'cim' => 'Kezdő oldal',
        'tipus' => 'kezdo'
    );
    $this -> joOldalak['rss'] = array (
        'cim' => 'RSS csatorna',
        'tipus' => 'rss'
    );
}
```

A kezdő és az RSS oldal nem az adatok tömbből származnak, mert azok a főmenübe nem fognak bekerülni.

Megjegyzés: Ezen a ponton is tetten érhető az inkrementális fejlesztés egyik hátránya: ez a két speciális oldal is bekerülhetett volna az adatok tömbbe, de akkor a főmenü generálását kellene módosítani. Nehéz eldönteni, hogy melyik a jobb megoldás.

```
foreach ($GLOBALS['oldalok'] as $cim => $menublokk) {
    foreach ($menublokk as $azon => $menupont) {
        $this -> joOldalak[$azon] = $menupont;
    }
}
```

Itt egészült ki a *joOldalak* tömb, hogy a kérés érvényességét minél előbb el tudjuk dönteni.

```
if (strlen($keres) == 0) {
    require_once ("lib/kezdoOldal.php");
    $this -> oldal = new kezdoOldal($this -> keres,
        $this -> joOldalak);
}
```

Ha a kérés üres, akkor a kezdőoldalt kell kiszolgálni. Érdekes már itt megfigyelni, hogy az oldal típusa dönti el, hogy a kimenet hogyan is generálódik majd.

```
else {
    $oldalKeres = strtok($keres, '/');
```

Kinyerjük a kérést első elemét a / karakterig.

```
if (isset($this->joOldalak[$oldalKeres])) {
    $osztalyNev = $this->joOldalak[$oldalKeres]['tipus']
        . 'Oldal';
    include_once ("lib/{$osztalyNev}.php");
    $this -> oldal = new $osztalyNev($keres);
}
```


Ha kérés első eleme (*\$oldalKeres*) szerepel a *joOldalak* között, akkor gondoskodunk a megfelelő nevű leszármazott példányosításáról. Különben a kezdőoldalra küldjük a klijenst.

Megjegyzés: Itt ismét célszerű lenne bővíteni egy 404 hibaoldal publikálása felé.

```

        else {
            $this -> kezdoOldalra();
        }
    }
}

```

A *general* függvény is csak egyszer kerül meghívásra, a példányosítás után. Feladata a válasz szöveg előállítás.

```

function general() {
    header("Content-type: "
        + $this -> oldal -> tartalomTipus());
}

```

A *Content-type* egyedi megadására van lehetőségünk.

```

return $this -> oldal -> general();
}

```

Végül a kezdőoldalra való visszajutás is meghívható:

```

function kezdoOldalra() {
    header('Location: ' . $_SERVER["PHP_SELF"]);
}
}

```

Oldal absztrakt osztály

Megjegyzés: Az *oldal* a PHP 4-es verziójához volt kötve a szolgáltató miatt, így az *abstract* kulcsszó csak jelzésértékű.

```

/* abstract */
class Oldal {
    var $keresek = array();
    var $keres;
    var $oldalSablon;
    var $adatbazis = null;
}

```

Kezdjük itt is az adattagokkal. A *\$keres* hasonló szerepet játszik, mint a *Vezerlo* osztálynál. A *\$keresek* ugyanezt az adatot már tömbként tartalmazza. Nézzük a kérés eltárolását:

```

function Oldal($k = null) {
    global $kezdooldal;

    if ($k == null) {
        $this -> keres = $kezdooldal;
        $this -> keresek[] = $kezdooldal;
    } else {
        $this -> keres = $k;
        $this -> keresek = explode('/', $k);
    }
}

```

Az *oldal* felépítése a *Sablon* osztály segítségével történik. Ezért itt egy példányt hozunk létre, ami be is tölti az *index* sablont, majd a sablonba különböző változókat pakolunk. (A sablonokról később lesz szó.)

```

$this -> oldalSablon = new Sablon('index');
$this -> oldalSablon -> berakTomb(
    $GLOBALS['portal_adatok']);
$this -> oldalSablon -> berak('fomenu',
    $GLOBALS['oldalak']);
if (! $this -> keresek[1]) {
    $this -> oldalSablon -> berak('aktualisFomenu',
        $this -> keresek[0]);
}
}

```

A *tartalom* függvény csak a leszármazottakban lát el hasznos feladatot.

```

/* abstract */
function tartalom() {
    return null;
}

```

A *general* függvény a sablonba teszi be a fő tartalmat, majd a sablon objektum segítségével elvégzi a sablonváltozók cseréjét.

```

function general() {
    $this -> oldalSablon -> berak('tartalom',
        $this -> tartalom());
    return $this -> oldalSablon -> cserel();
}

```

A *statikusOldal* osztály

Az osztály szerepe az egyetlen statikus oldalból álló tartalomrészek (pl. elérhetőségek) kiszolgálása:

```

class statikusOldal extends Oldal {
    function statikusOldal($k = null) {
        parent::Oldal($k);
    }
}

```

A PHP 4-ben figyelni kell az ős konstruktorának direkt meghívására. A *tartalom* függvény betölti a kérésnek megfelelő sablon állományt, amibe a *\$portal_adatok* tömb kerül még be.

```

function tartalom() {
    global $portal_adatok;

    $sablon = new Sablon($this -> keres);
    $sablon -> berakTomb($GLOBALS['portal_adatok']);

    $szoveg = $sablon -> cserel();

    return $szoveg;
}
}

```

A *blogOldal* osztály

A *blogOldal* osztály szerepe az egyes témákhoz (pl. *Web programozás*) kapcsolódó bejegyzések kiszolgálása. A *tartalom* függvény kódjából csak néhány fontosabb rész következik.

Itt most azt fogjuk látni, hogy a függvény sokat kommunikál az adatbázis-szerverrel, de ennek fizikai megvalósítása teljes egészében az *Adatbázis* osztályba van zárva. A *tartalom* függvény számára csak a függvényhívások érdekesek.

```
| $hirekSzoveg = '';
```

Ez a változó fogja tartalmazni a kimenetet. Megpróbálunk csatlakozni az adatbázishoz:

```
| $adatbázis = new Adatbázis("localhost", "nagygyusztav",
| "nagygyusztav", "***");
```

Csak működő kapcsolat esetén tudunk dolgozni:

```
| if ($adatbázis -> mukodik()) {
```

Ha a *keresek* nem egyelemű, vagyis egy konkrét bejegyzést is tartalmaz az URL, akkor ennek az egy bejegyzésnek (hírnek) és hozzászólásainak megjelenítése szükséges:

```
| if ($this -> keresek[1]) { // van konkrét kérés
|     $egyHir = $adatbázis -> egyHir($this -> keresek[0],
|     $this -> keresek[1]);
```

A lekérés után pedig sablonnal előállítjuk a végleges formát:

```
| $hirSablon = new Sablon('blogBlokKozza');
| $hirSablon -> berak('hirCim', $egyHir['cim']);
| $hirSablon -> berakTomb($GLOBALS['portal_adatok']);
| $hirSablon -> berak('datum', $datum);
| $hirSablon -> berak('url', $egyHir['url']);
| $hirSablon -> berak('blognev', $egyHir['blognev']);
| $hirSablon -> berak('lehetozza',
|     $egyHir['lehetozza']);
| $hirSablon -> berak('hirSzoveg', $egyHir['rovidtart']);
```

Ha a bejegyzéshez engedélyezettek a hozzászólások, akkor lekérdezzük és meg is jelenítjük azokat:

```
| if ($egyHir['lehetozza']) {
|     $hozzak = $adatbázis -> hozzaszolasok(
|         $this -> keresek[0], $this -> keresek[1]);
|     if ($hozzak) {
|         $hirSablon -> berak('hozzaszolasok', $hozzak);
|     }
| }
| $hirekSzoveg .= $hirSablon -> cserel();
```

A csak listázás esetén (*else* ág) hasonló technikával épül fel az oldal.

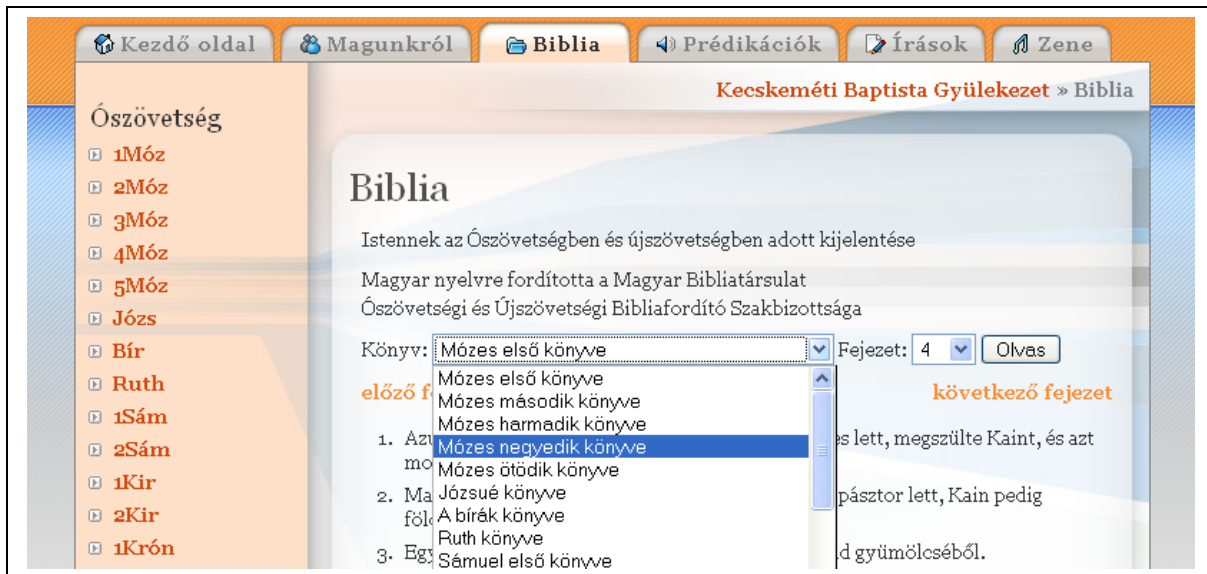
9.2. Biblia

A Kecskeméti Baptista Gyülekezet honlapjának⁵³ készítésekor felmerült az igény egy olyan szolgáltatásra, amely a Biblia szövegét az oldalba integrálja különböző szolgáltatások céljából. A következőkben két, már megvalósult szolgáltatás vázlata fog következni:

- Lehesse a Biblia szövegében gyorsan navigálni, akár könyvenként, akár fejezetenként lépkedni.
- Lehesse a honlap más részein linkelési lehetőséget biztosítani.

Nézzük meg először a működő megoldást, hogy így megérthessük a tervezési döntéseket.

⁵³ <http://kbg.hu>



Az ábrán látszik, hogy az aktuális könyv kiválasztható a bal oldali közvetlen linkekkel, vagy a lenyíló listából is. Az előbbi esetén az adott könyv első fejezetére lépünk, a második esetben pedig még van lehetőség a fejezet pontos kiválasztására is az Olvas gomb kattintása előtt.

Megjegyzés: Itt egy kis feladatot kap a JavaScript is: a könyv választása után a választható fejezetszámok listájának változnia kell, hiszen van olyan könyv, amelyik 150 fejezetes, és van, amelyik egyetlen fejezetből áll.

A következő két ábra a linkelés indulási és érkezési oldalára mutat példát:



A folyó szövegben látható első link az Efézusi levél 4. fejezetének 11-12. versére utal. A felhasználó számára kényelmi szolgáltatás, ha a linkre kattintva a szöveggörnyezetében meg tudja nézni a hivatkozott verseket.

Megjegyzés: Az oldal más linkektől eltérő szaggatott aláhúzással jelzi, hogy a link egy bibliai hivatkozás.

Megjegyzés: Egy másik megoldás lehetett volna, hogy a hivatkozás itt helyben, a *title* tulajdonsággal jelenjen meg. Ennek azonban két hátránya is van. Egyrészt hosszabb hivatkozás esetén zavaró lehetne a túl hosszú szöveg (akár egész fejezetre is szokás hivatkozni). Másrészt így nem lehetne szöveggörnyezetében tanulmányozni az adott igeverseket, csak kiragadva.

Még egy olyan megoldás is elképzelhető lenne, hogy egyetlen igevers esetén jelenjen meg a szöveg *tooltip*-ként, és a link is működjön minden esetben. Erre valószínűleg a legjobb lenne egy JavaScriptes megoldást alkalmazni: az oldal betöltődése után a JavaScript végignézi az adott oldal *a* tagjait, *class* alapján kiválogatja az igelinkeket, betölti a hozzá tartozó igeverset, és elhelyezi a *title* értékeként.

A következő ábra mutatja a kattintás következményét. Jól látható, hogy a 11-12-es vers kiemelten jelenik meg:

▢ Péld	8. Ezért mondja az Írás: "Felment a magasságba, foglyokat vitt magával, ajándékot adott az embereknek."
▢ Préd	9. Az pedig, hogy "felment", mi más jelent, mint azt, hogy le is szállt erre a földre.
▢ Énekek	10. Aki leszállt, az "fel is ment", feljebb minden égnél, hogy betöltse a mindenséget.
▢ Ézs	11. És ő "adott" némelyeket apostolokul, másokat prófétákul, ismét másokat evangélistákul, vagy pásztorokul és tanítókul,
▢ Jer	12. hogy felkészítse a szenteket a szolgálat végzésére, a Krisztus testének építésére,
▢ JSir	13. míg eljutunk mindnyájan a hitnek és az Isten Fia megismerésének egységére, a felnőttkorra, a Krisztus teljességét elérő nagykorúságra,
▢ Ez	14. hogy többé ne legyünk kiskorúak, akik mindenféle tanítás szelében ide-oda hányódnak és sodródnak az emberek csalásától, tévútra csábító ravaszágától;
▢ Dán	
▢ Hós	
▢ Jóel	

Tervezés

A tényleges kódolás előtt két dolgot kell még tisztázni:

- Hol és hogyan tároljuk a Biblia szövegét?
(A terjedelem 66 könyv, 1189 fejezet, 3.676.751 karakter)
- Hogyan tervezzük meg az oldalhoz tartozó URL-t?

Az adattároláshoz lehetne adatbázist alkalmazni, sőt a kereshetőséghez szükséges is lenne. (A kereshetőség még nincs megoldva a honlapon.) Az oldal fejlesztésekor azonban egyéb okok miatt inkább a szöveges tárolás valósult meg. Gyakorlatilag tehát minden könyv egy neki megfelelő nevű alkönyvtárban található, azon belül pedig a fejezetek egy-egy szöveges állományban, soronként 1-1 verset tartalmazva. Például a János 3,16 a következő helyen van: jn/3.txt 16. sora.

Az URL tervezésekor sem kell túl bonyolult dolgokat kitalálni, hiszen a szokásos ígehely-hivatkozások kódolása kicsit megváltoztatva tökéletesen alkalmas lesz. Így az előző ábrákon szereplő bibliaoldalak URL-jei a következők:

- <http://kbggy.hu/index.php?biblia/1moz/4>
- <http://kbggy.hu/index.php?biblia/ef/4/11-12>

Jól láthatók, hogy az URL-ben használatos hierarchikus hivatkozás itt is jó megoldás. A könyv és fejezet megadása szükséges, azon belül a versek megadása csak a kiemeléshez kell.

Adatok

A Biblia könyveinek néhány adatát tömbökben tároljuk. A tömbök első sorai következnek. Először a gépi tároláshoz (könyvtárnév, URL-tag) alkalmazott rövidítések:

```
$biblia_gepi_nevek = array (
    "1moz",
    "2moz",
    "3moz" ...
```

A bibliai könyvek rövidített nevei:

```
$biblia_nevek = array (
    "1moz" => "1Móz",
    "2moz" => "2Móz",
    "3moz" => "3Móz" ...
```

A könyvek teljes címei:

```
$biblia_cimek = array (
    "Mózes első könyve",
    "Mózes második könyve",
    "Mózes harmadik könyve" ...
```

Végül pedig a könyvek fejezetszámai:

```
$biblia_fejezetszamok = array (
    "1moz" => 50,
    "2moz" => 40,
    "3moz" => 27,
```

Konstruktor

Következő lépésként az URL megfelelő részét fel kell dolgozni, figyelembe véve, hogy a megadott értékek lehetnek hibásak is. A következő kódrészletek egy *BibliaOldal* nevű osztályból származnak. Ez az osztály felelős az URL feldolgozásáért és a kimenet előállításáért.

A konstruktorban történik az URL feldolgozása. A *\$this -> keresek* tömb az URL kérdőjel utáni részét „robbantja szét”:

```
| $this -> keresek = explode ('/', $k);
```

Következnek az ellenőrzések.

Ha nincs könyv megadva, az első könyv első fejezetére küldi az oldalt:

```
| if (!isset($this -> keresek[1])) {
|     $this -> kezdOldalra($this -> keresek[0] . '/'
|     . $biblia_gepi_nevek[0] . '/1');
| }
```

Ha van név megadva, de az nem érvényes, szintén visszaküld:

```
| elseif (!in_array($this -> keresek[1], $biblia_gepi_nevek)) {
|     $this -> kezdOldalra($this -> keresek[0] . '/'
|     . $biblia_gepi_nevek[0] . '/1');
| }
```

Ha nincs fejezetszám, akkor az adott könyv első fejezetére küld:

```
| elseif (!isset($this -> keresek[2])) {
|     $this -> kezdOldalra($this -> keresek[0] . '/'
|     . $this -> keresek[1] . '/1');
| }
```

Ha van fejezetszám, de az túl nagy, akkor is visszaküld:

```
| elseif ($biblia_fejezetszamok[$this -> keresek[1]]
|     <$this -> keresek[2]) {
|     $this -> kezdOldalra($this -> keresek[0] . '/'
|     . $this -> keresek[1] . '/'
|     . $biblia_fejezetszamok[$this -> keresek[1]]);
| }
```

Végül a 0 és negatív fejezetszámok is kiszűrésre kerülnek:

```
| elseif ($this -> keresek[2]<1) {
|     $this -> kezdOldalra($this -> keresek[0] . '/'
|     . $this -> keresek[1] . '/1');
| }
```

Bal oldali linkek gyártása

A tényleges linkek legyártása sablon megoldással történik, itt csak a végeredmény következik:

```
<ul class="menublokk">
  <li><a href="index.php?biblia/1moz/1">1Móz</a></li>
  <li><a href="index.php?biblia/2moz/1">2Móz</a></li>
  <li><a href="index.php?biblia/3moz/1">3Móz</a></li>
  ...
</ul>
```

Aktuális oldal gyártása

Az ellenőrzésen már túl van a kódunk, tehát az adott fejezet létezik. A következő kód egy tömbbe betölti a teljes fejezetet:

```
$fejezet_file_nev = 'adat/biblia/' . $this -> keresek[1]
  . '/' . $this -> keresek[2] . '.txt';
$fejezet_sorok = array();
$f = fopen($fejezet_file_nev, "r");
while (!feof($f)) {
  $sor = fgets($f);
  if (strlen($sor) == 0) {
    break;
  }
  $fejezet_sorok[] = $sor;
}
fclose($f);
```

A következő kód a fejezetszámon belül kiemelendő versekkel foglalkozik. Ha itt hiba van, azt lényegében figyelmen kívül hagyjuk. Itt a szintaxis egy pontokkal elválasztott lista, ahol egyedi számok mellett intervallumok is megadhatók.

Pontok mentén robbantjuk az URL végét:

```
if (isset($this -> keresek[3])) {
  $t = explode('.', $this -> keresek[3]);
```

A *jelolt* tömbben gyűjtjük a kiemelésre jelölt versek számait.

```
$jelolt = array();
foreach ($t as $resz) {
```

Nézzük meg, van-e benne kötőjel. Ha nincs, akkor egyedi számként próbáljuk kezelni:

```
  $pos = strpos($resz, '-');
  if ($pos === false) {
    settype($resz, "integer");
    if (($resz > 0) && ($resz <= count($fejezet_sorok))) {
      $jelolt[$resz-1] = 1;
    }
  }
}
```

Ha van kötőjel, akkor az intervallumot is figyelembe vesszük:

```

else {
    $seleje = (integer) substr($resz, 0, $pos);
    $vege = (integer) substr($resz, $pos+1);
    if ( ($seleje <= $vege) &&
        ($seleje > 0) &&
        ($vege <= count($fejezet_sorok)) ) {
        for (; $seleje <= $vege; $seleje++) {
            $jelolt[$seleje-1] = 1;
        }
    }
}
}

```

Végül a jelöltként összegyűjtött versszámokat *strong* tagba foglaljuk.

```

foreach ($jelolt as $k => $e) {
    $fejezet_sorok[$k] = '<strong>' . $fejezet_sorok[$k]
        . '</strong>';
}
}

```

Innentől a feladat a *\$fejezet_sorok* tömb kiírása, pl. egy számozott listaként.

9.3. 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 cikk⁵⁴ 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*⁵⁵ címmel *jpmaster77* tollából.

Adatbázis

A felhasználók adatainak tárolásához (legegyszerűbb esetben) egyetlen táblára van szükségünk. Ennek szerkezete a következő SQL paranccsal kialakítható:

```

CREATE TABLE users(
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nick VARCHAR(30),
    jelszo VARCHAR(32),
    email VARCHAR(60)
);

```

⁵⁴ http://www.segedletek.hu/segedletek/php/php_p_mysql_alapu_belepteto_rendszer.html

⁵⁵ <http://www.evolt.org/PHP-Login-System-with-Admin-Features>

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_check.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) {
        $hibak['nick'] = 'A nick legalább 4 betűs legyen';
    } else if (!ereg('^[a-zA-Z0-9\-\_áéíóöüóúÁÉÍÓÖŰŰ]+$',
        $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'] == (!ereg('^[_\.\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");
?>

```

9.4. 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 phpPatterns⁵⁶ oldala. Ezen kívül a Zend *PHPPatterns: Instructions*⁵⁷ 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 újrahaznosítható alkalmazásokat készíthetünk.

E fejezetben néhány alapvetőbb tervezési mintát fogunk megvizsgálni.

9.4.1 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.)

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;
} ?>
```

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. A következő dinamikus példa a PHP-Nuke megoldása:

```
if (!isset($file)) { $file="index"; }
...
$modpath .= "modules/$name/$file.php";
if (file_exists($modpath)) {
    include($modpath);
} else {
    die ("Sorry, such file doesn't exist...");
}
```

⁵⁶ <http://www.phppatterns.com/>

⁵⁷ <http://devzone.zend.com/node/view/id/3>

9.4.2 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.

A fejezet példája⁵⁸ segítségével az űrlap által küldött adatok érvényességének ellenőrzését (validálását) fogjuk elvégezni.

Nézzünk egy hagyományos, nem túl praktikus megközelítést:

```
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 (logikailag⁵⁹) 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:

⁵⁸ A példa a http://www.phppatterns.com/docs/design/strategy_pattern oldalról származik

⁵⁹ A PHP 4-es verziója még nem ismeri az absztrakt osztály fogalmát

```
class ValidateUser extends Validator {
    var $user;

    function ValidateUser ($user) {
        $this->user=$user;
        Validator::Validator();
    }

    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');
        }
    }
}
```

Érdeemes megfigyelni, hogy ez a kód épít a PHP 4-es verziójának azon specialitására, ami szerint az őosztály konstruktorát nem hívja meg automatikusan a leszármazott konstruktor. (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() {
    $pattern=    "/^([a-zA-Z0-9]+([\.\a-zA-Z0-9_-])*@[a-zA-Z-
Z0-9_-])+(\.[a-zA-Z0-9_-]+)+/";
    if(!preg_match($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 } ?>

```

9.4.3 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 megoldásának csupán egy részével fogunk foglalkozni. A teljes példa megtalálható a phpPatterns oldalán⁶⁰.

Nézzük először az adatbázis-kapcsolat felépítéséért és az SQL parancsok futtatásáért felelős *DataAccess* osztályt:

⁶⁰ http://www.phppatterns.com/docs/design/data_access_object_pattern_more_widgets

```

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:

```

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.

9.4.4 MVC⁶¹

Az MVC (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→	adatfeldolgozás→	eredmény megjelenítése
Controller (Vezérlő) →	Model (Modell) →	View (Nézet)

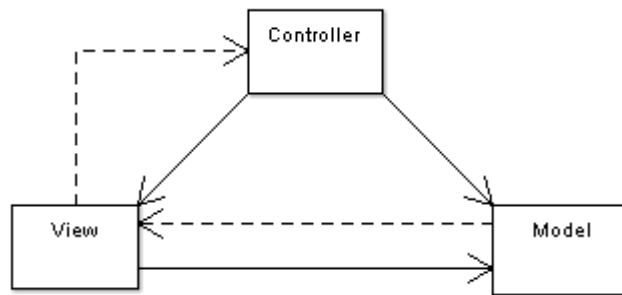
A **vezérlő** dolgozza fel a felhasználói adatbevitelt. Parancsokká és függvényhívásokká képezi le azokat. Ezek a parancsok fogják előidézni az adatok módosítását, törlését, vagy a nézetek megváltozását. Például ha a felhasználó kiválasztja a menü egyik elemét, akkor egy controller fogja meghatározni, 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.

A Model, View és Controller kapcsolatai. A folytonos vonal a közvetlen, a szaggatott pedig az indirekt kapcsolatokat mutatja.

⁶¹ Barkóczy Roland szakdolgozata alapján



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.

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 modelltől 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.

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 dizájnerek akár párhuzamosan, egymás zavarása nélkül is végezheti a dolgát. Ráadásul egymás munkáját is elég alacsony szinten ismerni az együttműködéshez.

10.1. Smarty

A Smarty⁶² 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 dizájnerek önállóan is menedzselni tudja, nincs szükség a programozó beavatkozására.

Biztonságos

A sablonok nem tartalmazzák PHP kódot. Ezzel csökkentjük ugyan a dizájnerek 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 dizájnereknek nem kell a PHP bonyolult szintaxisával megismerkedni, helyette egy (a HTML-hez sok tekintetben közelebb álló) sablon-nyelvet kell alkalmazni.

⁶² <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 dizájnér 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 dizájnér és a programozó sablonrendszer használatával nagyjából a következő módon dolgozhatnak a követelményspecifikáció elkészítése után:

- A dizájnér 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 dizájnér 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 cikket publikáló honlapot. Az egyes cikket 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 dizájnér 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 oldalról⁶³, 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 érniük. Éppen ezért vagy egy globálisan elérhető helyre, vagy pl. a portál gyökérmappájába célszerű helyezniük. 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 dizájnner hozza létre: sablonok és konfigurációs állományok kerülnek beléjük.

A webservert 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*):

⁶³ <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 kiírás 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=$smarty.env key=key item=value }
  <tr><td>{$key}</td><td>{$value}</td></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 fejléctet és lábléctet 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 következzenek néhány függvény, amelyek úr-
latok tömbből való generálását könnyítik meg:

Html_checkboxes

Jelölőnégyzeteket 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 el-tárolt 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:

```
$smarty = new Smarty;
if(!is_cached('sajatoldal.tpl', md5($_COOKIE['name']))) {
    /* beállítás */
    $smarty->assign('name', $_COOKIE['name']);
}$
$smarty->display('sajatoldal.tpl', md5($_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 szemétygyűjtéssel, így minden tárolt oldal egy fájl 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 dizájnner 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.

Érdeemes 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 massassi⁶⁴ megoldása alapján egy kissé testreszabott megoldás, a nagygyusztav.hu oldal korábban bemutatott kódjához illeszkedik.

⁶⁴ http://www.massassi.com/php/articles/template_engines/

```
<?
class Sablon {
    var $dir = 'sablon/';
```

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. Érdeemes 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 nagygyusztav.hu oldal kódjából. A következő sablon állomány (*ujBejegyzesekBlokkt.tpl.php*) egy blogbejegyzés előzetest fog készíteni a kezdőoldalra.

```
<? /* Csak az előzetes megjelenítésére, hozzászólások számával
*/ ?>
<div class="hir">
<? if ($lehetőzza || $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>
<? } ?>
<?= $hirSzoveg ?>
<? if ($lehetőzza || $tartalom) { ?>
<p class="hozzaszolasszam">
<? if ($tartalom) { ?>
  <a href="<?= $index ?><?= $blognev ?>/<?= $url ?>">
    Teljes bejegyzés&raquo;</a>
<? }?>
<? if ($lehetőzza) { ?>
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'];
}
$ssablon = new Sablon('ujBejegyzesekBlokkt');
$ssablon -> berakTomb($GLOBALS['portal_adatok']);
$ssablon -> berak('ujBejegyzesek', $ujBejegyzesek);
$sszoveg .= $ssablon -> cserel();
```

Források

- George S. Schlossnagle: PHP fejlesztés felsőfokon
(<http://www.kiskapu.hu/>)
- deadline: PHP, mint sablonnyelv
(<http://deadline.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/)

11. Keretrendszerek

A webfejlesztők szóhasználatában a keretrendszerek (*frameworks*) olyan kódgyűjteményt jelentenek, amelyek a fejlesztő munkáját nagyban egyszerűsíteni tudják, mivel gyakran előforduló problémákra kész megoldásokat nyújtanak.

A keretrendszerek között találunk olyanokat, amelyek esetén a nyújtott szolgáltatások csak lazán kapcsolódnak egymáshoz (esetleg teljesen függetlenek egymástól), míg más esetekben egy egységes rendszert alkotva, a saját kódunkat csak mint „dekorációt” ráaggatva kell elhelyeznünk.

A keretrendszerek tipikus szolgáltatásai közé tartozik az adatbázis-kezelés absztrakciója, sablonkezelés, gyorstárazás, munkamenet-kezelés, felhasználó-kezelés, stb. Vannak rendszerek, amelyek kimondottan MVC alapokra épülnek, és vannak, amelyek a kliens és szerver oldal közötti AJAX alapú kommunikációt helyezik a középpontba.

Manapság nem az a fő kérdés, hogy használjunk-e keretrendszert, hanem inkább az, hogy melyiket alkalmazzuk a sok elérhető közül. A választás nem könnyű, és nem is „veszélytelen”: egy keretrendszer felépítésének és API-jának megismerése felérhet egy új nyelv megtanulásának költségeivel. Ha egy rendszer nem váltja be a hozzá fűzött reményeket, sok munka mehet veszendőbe.

Az összes keretrendszer bemutatása helyett csak kettőre, a Code Igniter-re és a CakePHP-re fogunk koncentrálni. Érdeemes megfigyelni, hogy a keretrendszerek sokszor másolják a nagy elődök bevált ötleteit.

Megjegyzés: Erről egy kis képet kaphatunk a WACT oldalán⁶⁵ található listából: a közel 50 keretrendszer közül 20 valamelyik másik rendszer logikájára épül. A „másolás” többnyire más platformbeli rendszerről történik. A legtöbben a Ruby on Rails⁶⁶ alapötletét másolják, így a Code Igniter és a CakePHP is.

11.1. Code Igniter

A Code Igniter⁶⁷ fejlesztői egy olyan webalkalmazás fejlesztő keretrendszert hoztak létre, amely a PHP alapú webfejlesztést komoly eszközökkel támogatja. A fontosabb jellemzői:

- ingyenes
- nyílt forrású
- PHP 4 vagy 5 alatt is működik
- könnyűsúlyú
- gyors
- MVC alapú
- tiszta URL-eket alkalmaz
- kiegészíthető
- nem szükséges hozzá külön sablonrendszer
- jól dokumentált.

⁶⁵ http://www.phpwact.org/php/mvc_frameworks

⁶⁶ <http://www.rubyonrails.hu/>

⁶⁷ <http://codeigniter.com/>

11.1.1 Telepítés

A Code Igniter alkalmazásához PHP és adatbázis-szerver (MySQL, MySQLi, MS SQL, Postgre, Oracle, SQLite, vagy ODBC) szükséges.

A rendszert egyetlen tömörített fájlként letölthetjük. A fájl tartalmát kell az oldal gyökérkönyvtárába másolnunk.

Megjegyzés: az *index.php* mellett csak a *system* alkönyvtár nélkülözhetetlen, a *user_guide* a használatához szükséges sűgöt tartalmazza.

11.1.2 Az URL-ek felépítése

Az URL-ek felépítése lehetővé teszi, hogy mind az emberi látogatók, mind a keresőrobotok számára beszédes címeink legyenek. Az URL nagyon jól olvasható módon mutatja az oldal tartalmi hierarchián belüli pozícióját:

```
| www.your-site.com/news/article/my_article
```

Az URL elemei a következő logikára épülnek:

```
| www.your-site.com/class/function/ID
```

Ez a szerkezet tehát mutatja, hogy a kérés a *class* osztály *function* nevű függvényének szól, amely paraméterként megkapja az ID értéket. A megfelelő osztály betöltése, példányosítása és a függvény paraméteres meghívása mind a Code Igniter feladata, a fejlesztőnek ezzel már nem kell törődnie.

Igazság szerint az URL tartalmaz egy *index.php* elemet is, de ezt a felhasználók elől szeretnénk elrejtetni.

```
| www.your-site.com/index.php/news/article/my_article
```

11.2. CakePHP

A CakePHP⁶⁸ egy ingyenes, nyílt forráskódú, gyors fejlesztői keretrendszer PHP-hez. Könyvtárak, osztályok struktúrája és run-time infrastruktúra azon webes alkalmazásokat készítő programozók számára, akiket megihletett a Ruby on Rails keretrendszer. Elsődleges célunk hogy lehetővé tegyük a rendezett és gyors munkát – a rugalmasság elvesztése nélkül.

11.2.1 Alapkonceptió

Ebben a fejezetben röviden, tömören szeretnénk bemutatni a Cake-ben is megvalósított MVC koncepciót. Az MVC általános bemutatása után megnézzük, hogyan is lett megvalósítva az MVC a CakePHP-ban, majd néhány egyszerű CakePHP példán keresztül megnézzük, hogyan lehet alkalmazni.

Az MVC tervezési minta

A Model-View-Controller (Modell Nézet Vezérlő) tervezési minta segítségével logikailag csoportosíthatjuk a kódot, elősegítvén az újrahasznosítást, a karbantarthatóságot, és általában véve jobb kód készítését.

⁶⁸ <http://cakephp.hu/>

(A fejezet tartalma a CakePHP dokumentációja alapján készült.)

Az MVC-t eredetileg arra tervezték, hogy a hagyományos input, feldolgozás és output feladatok leképezzék a grafikus interfésszel rendelkező alkalmazásokban.

A Cake terminológiájában a **Model** jelképezi az egyes adatbázis táblákat/rekordokat, és azok kapcsolatait más táblákhoz/rekordokhoz, valamint ez az egység tartalmazza az adat-érvényességi szabályokat, melyeket az adatok bevitelkor vagy frissítésekor kell figyelembe venni.

A **View** a Cake nézet fájljait jelenti, amelyek PHP kóddal beágyazott szabvány HTML fájlok.

A Cake **Controller** kezeli a szerver felőli kéréseket. Ez a modul fogadja a felhasználói adatbevittelt (GET vagy POST adatokon keresztül), megvalósítja az üzleti logikát, a Model-t felhasználva írja/olvassa az adatbázis és más adatforrások adatait, és végül megjelenítendő adatokat küld a megfelelő View fájloknak.

Azért, hogy alkalmazásainkat a lehető legegyszerűbben építhessük fel, a Cake az alkalmazáson belüli objektumok interakciójának kezelésén felül a fájlok tárolásának megszervezésére is használja az MVC mintát.

A Cake fájlszerkezetének áttekintése

Amikor kicsomagoljuk a Cake-t, a szerveren a következő három könyvtárat találjuk:

```

| app
| cake
| vendors

```

A *cake* mappában található a rendszer magjához tartozó könyvtárak. Ezt a könyvtárat sosem kell módosítani.

Az *app* könyvtárban fognak elhelyezkedni az alkalmazásunk fájljai. A *cake* és *app* mappa különválasztásával lehetővé válik, hogy több *app* könyvtár osztozzon egyetlen *cake* könyvtárrendszeren. Emiatt a CakePHP frissítése is egyszerűvé válik: csak le kell tölteni a Cake legfrissebb változatát, és máris másolhatjuk a *cake* könyvtárba, felülírva a régi fájlokat. Nem kell aggódnunk amiatt, hogy a saját alkalmazásunk fájljait cseréljük ki.

A *vendor* könyvtárban a mások által fejlesztett alkalmazásokat tároljuk. A későbbiekben részletesen ismertetjük ezeket, előljáróban csak annyit, hogy az alapötlet az, hogy a Cake *vendor()* függvényének segítségével érhetjük el az itt elhelyezett osztályokat.

Lássuk egyben a fájlszerkezet fontosabb elemeit:

```

| /app
|   /config          config fájlokat tartalmaz az adatbázishoz,
|                       ACL-hez, stb
|   /controllers    vezérlők helye
|   /components     komponensek helye
|   /index.php      lehetővé teszi, hogy a cake a /app könyvtárat
|                       DocumentRoot-ként kezelje
|   /models         modellek
|   /plugins        Pluginek
|   /tmp            cache és log fájlok helye
|   /vendors        külső alkalmazások helye

```

/views	nézetek
/elements	elemek, kis nézet-darabok
/errors	testreszabott hibaüzenet oldalak
/helpers	helperek helye
/layouts	az alkalmazás layout fájljai
/pages	statikus nézetlapok
/webroot	az alkalmazás documentRoot-ja
/css	
/files	
/img	
/js	
/cake	cake rendszerkönyvtár
index.php	
/vendors	szerveroldali külső alkalmazásokhoz használt könyvtár.
VERSION.txt	a telepített Cake rendszer verziószáma.

11.2.2 A CakePHP telepítése

A következő módszer, amivel a CakePHP-t beállítjuk, csak fejlesztési környezetre ajánlott, a kevésbé biztonságos volta miatt.

Megjegyzés: az /app/tmp elérési útnak írhatóknak kell annak a felhasználónak, amelyen fut a webszerver. (Linux szerver esetén az apache-nek kell tudnia írni ezt a könyvtárat.)

Fejlesztői telepítés

Fejlesztéskor a teljes Cake telepítési könyvtárstruktúráját közvetlenül a webelérési útra be lehet másolni, mint ahogyan látható:

```
| /wwwroot
|   /cake
|     /app
|     /cake
|     /vendors
|     .htaccess
|     index.php
```

A fejlesztői telepítéskor a wwwroot könyvtár mint a web gyökérkönyvtára funkcionál, ezért az URL-jeink a következőképpen néznek ki (mod_rewrite használata esetén):

```
| www.example.com/cake/controllerName/actionName/param1/param2
```

Végleges telepítés

A végleges telepítés testreszabásához szükségesek a DocumentRoot-hoz való jogok a szerveren. Ebben az esetben a teljes domain, mint egy teljes CakePHP alkalmazás működik.

A végleges telepítés a következő felépítéssel bír:

```

/útvonal_a_cake_fájlokhoz
/app
  /config
  /controllers
  /models
  /plugins
  /tmp
  /vendors
  /views
  /webroot <-- Ennek az elérési útnak kell lennie
                a DocumentRoot-nak

.htaccess
index.php
/cake
/vendors
.htaccess
index.php

```

httpd.conf ajánlott beállításai

```
| DocumentRoot /útvonal_a_cakehez/app/webroot
```

Ilyen beállítások esetén a webroot könyvtár, mint web gyökérkönyvtára érhető el és az URL-ek a következőképpen néznek ki (mod_rewrite használata esetén):

```
| http://www.example.com/controllerName/actionName/param1/param2
```

Bizonyosodjunk meg a helyes működésről

Rendben, nézzük mit csináltunk. Attól függően melyik telepítést választottuk, a böngésződ <http://www.example.com> vagy <http://www.example.com/cake> elérési útra kell beállítani. Ennél a pontnál a CakePHP alapoldala fog megjelenni, és egy üzenetet kapunk, amely az adatbázis-kapcsolat státuszát mutatja.

11.2.3 Konfiguráció

Adatbázis konfiguráció

Az adatbázis konfigurációd az `app/config/database.php` fájlban található. Az új telepítés nem tartalmazza a `database.php` fájlt, így létre kell hoznod egy másolatot a `database.php.default` fájlról. Amint elkészítetted a másolatot és átnevezted, az alábbiakat fogod látni (`app/config/database.php`)

```

var $default = array('driver' => 'mysql',
                    'connect' => 'mysql_connect',
                    'host' => 'localhost',
                    'login' => 'user',
                    'password' => 'password',
                    'database' => 'project_name',
                    'prefix' => );

```

Cseréld ki az alpból megadott információkat az adatbázisodhoz való kapcsolódáshoz szükséges adatokra.

Megjegyzés az előtaggal (prefix) kapcsolatban: a karaktorsor amit megadsz, minden SQL híváshoz meg lesz adva, amit a Cake készít az adatbázisodhoz, amikor a táblákkal dolgozol. Itt határozod meg egyszer, ezáltal nem szükséges máshol megadnod. Ugyancsak megengedi, hogy kövesd a Cake tábla elnevezési konvencióját, ha olyan hozton vagy, ami csak egy adatbázisnevet enged meg.

Megjegyzés: HABTM kapcsolt táblákhoz csak egyszer add meg a prefixet: prefix_apples_bananas, nem pedig prefix_apples_prefix_bananas.

A CakePHP az alábbi adatbázis meghajtókat támogatja:

- mysql
- postgres
- sqlite
- pear-drivertype (így írhatod pl. pear-mysql-t)
- adodb-meghajtónév

A *connect* kulcs az alap (*\$default*) kapcsolatnál lehetővé teszi, hogy az adatbázis kapcsolatot állandóra állítsd vagy sem. Olvasd el a megjegyzéseket a *database.php.default* fájlban az adatbázishoz kapcsolódási módok megadásában való segítséghez.

Az adatbázis tábláinak az alábbi konvenciókat kell követniük:

- A Cake által használt táblaneveknek az angol többes számot kell használniuk pl. "users", "authors" or "articles". Vedd figyelembe, hogy a megfelelő modelleknek egyes számú nevük van.
- A tábláknak tartalmazniuk kell egy elsődleges kulcsot: 'id'.
- Amennyiben össze szeretnéd kapcsolni a táblákat, a kulcsot a táblanévé mögé írd, így: 'article_id'. A táblanévé egyes szám, amit alsó aláhúzás, majd az 'id' követ.
- Amennyiben belefoglalod a létrejött és/vagy módosított oszlopokat a táblába, a Cake automatikusan kitölti a mezőt, ha helyes.

Láthatod, hogy, hogy a *database.php* fájl magában foglalja a *\$test* kapcsolat beállítását. Töltsd ki ezt a konfigurációt (vagy adj hozzá egy másik, hasonlóan formázott konfigurációt) és ezt használd az alkalmazásodban, a következő sort elhelyezve: *var \$useDbConfig = 'test';*

Az egyik modelleden belül. Ezen a módon bármennyi további kapcsolati beállítást is adhatsz hozzá.

Globális konfiguráció

A CakePHP globális konfigurációja a *app/config/core.php* állományban található. Mivel igazán nem kedveljük a konfigurációs fileokat, essünk túl rajta. Van itt néhány dolog, amit meg tudsz változtatni itt, és ezeknek a beállításoknak a leírását a kommenteken belül megtalálod a *core.php* fileban.

DEBUG: Ezt különböző értékekre állítva, segít téged az alkalmazásod fejlesztése közben a hibakeresésben. Nullától különböző értékekre állítva, kényszeríteni fogja a Cake-et, hogy jelenítse meg a *pr()* és a *debug()* függvényhívások eredményét, és megakadályozza a felvillanó üzenetek az automatikus átirányításról. 2-es vagy magasabb értékekre állítva, az SQL utasítások megjelenését eredményezi a lap alján.

Szintén, amikor debug módban vagyunk (amikor a *DEBUG* változó 1 vagy magasabb értékre van állítva), a Cake meg fog jeleníteni bizonyos generált hibaoldalakat, úgymint "Missing Controller" (Hiányzó Vezérlő), "Missing Action" (Hiányzó akció), stb. Éles rendszer esetén, (amikor a *DEBUG* 0-ra van állítva), Cake a "Not Found" oldalt jeleníti meg, ami az *app/views/errors/error404.html* állományban felülbíráható.

CAKE_SESSION_COOKIE: Változtasd meg ennek az értékét annak a cookie-nak a névre, amit használni szeretnél a felhasználói session-ök kezelésére a Cake alkalmazásodban.

CAKE_SECURITY: Változtasd meg ennek az értékét az általad kedvelt session-ellenőrzés jelölésére. A Cake gondoskodni fog az időtúllépésről, új session id-k generálásáról, és a régi session file-ok törléséről az általad itt megadott beállításoknak megfelelően. A lehetséges értékeke a következők:

- high (magas): a session 10 perc inaktivitás után lejár, és a session id-k minden kérés után újragenerálódnak
- medium (közepes): a session 20 perc inaktivitás után lejár
- low (alacsony): a session 30 perc inaktivitás után lejár

CAKE_SESSION_SAVE: Határozd meg, hogy hogyan szeretnéd a session adatokat elmenteni. Lehetséges értékeke:

1. cake: A session adat a tmp/-be kerül elmentésre a Cake telepítésedben
2. php: A session adat a php.ini-ben meghatározott módon kerül elmentésre
3. database: A session adat a 'default' kulccsal megadott adatbázis-kapcsolatnak megfelelően lesz elmentve.

Útvonal konfiguráció

A "Routing" egy leegyszerűsített, tisztán PHP alapú, mod_rewrite-szerű valami, ami képes leképezni az URL-eket a controller/action/params között oda-vissza. Ezt adták hozzá a Cake-hez, hogy a szép URL-ek még konfigurálhatóbbak legyenek, és hogy elválasszák tőlünk a mod_rewrite szükségességét. Habár a mod_rewrite a címsort sokkal rendezettebben tartja.

Az útvonalak egyedi szabályok, amik leképezik az illeszkedő URL-eket a meghatározott vezérlőkre és akciókra. Az útvonalak az app/config/routes.php file-ban konfigurálhatóak. Ezeket így lehet beállítani:

Útvonal Minta

```
<?php
$Route->connect (
    'URL',
    array('controller'=>'controllername',
          'action'=>'actionname', 'firstparam')
);
?>
```

Ahol:

- URL a szabályos kifejezéssel a Cake URL, amit szeretnél leképezni,
- a controllername a vezérlő neve, amit szeretnél hívni,
- actionname a vezérlő akciójának/függvényének a neve, amit hívni szeretnél,
- és a firstparam az akció/függvény első paramétere, amit megadtál.

Minden paraméter, ami a firstparam-ot követi, szintén elküldésre kerülnek a vezérlő akciójához/függvényéhez.

A következő példa illeszti az összes URL-t a /blog-ban a BlogController-hez. Az alapértelmezett akció a BlogController::index() lesz.

Útvonal Példa

```
<?php
$route->connect ('/blog/:action/*',
                array('controller'=>'Blog', 'action'=>'index'));
?>
```

Egy /blog/history/05/june -hoz hasonló URL eztuán a következőképpen kezelhető:

Útvonal kezelése egy Vezérlőben

```
<?php
class BlogController extends ApplicationController
{
    function history ($year, $month=null)
    {
        // .. A megjeleníti megfelelő tartalmat
    }
}
?>
```

A 'history' az URL-ből a :action-nel volt illesztve a Blog útvonalából. A *-gal illesztett URL elemek az aktív vezérlő kezelő függvényének vannak továbbadva paraméterként, itt a \$year és \$month. A /blog/history/05 URL-lel meghívva, a history()-nak csak egy paramétert kell átadnia: 05.

A következő példa egy alapértelmezett CakePHP útvonal, ami arra van használva, hogy beállítson egy útvonalat a PagesController::display('home')-ra. A Home egy nézet, amit a /app/views/pages/home.thtml file létrehozásával felülbíráhatunk.

Az alapértelmezett útvonal megadása

```
<?php
$route->connect ('/', array('controller'=>'Pages',
                           'action'=>'display', 'home'));
?>
```

11.2.4 GyorsVáz azaz Scaffolding képességek

A Cake GyorsVáz (Scaffolding) képességei nagyon jók

Annyira jó, hogy éles alkalmazásokban is szeretnénk használni. Nos, mi is azt gondoljuk, hogy jó, de a GyorsVáz nem más, mint GyorsVáz. Egy projekt kezdeti stádiumában egy jó adag tudást kell összeszednünk. Ennek elősegítésére szolgál, nem pedig, hogy teljesen rugalmas legyen. Amennyiben az alkalmazásunk logikáját és nézeteit testre szeretnénk szabni, eljön az ideje elvetni a GyorsVáz képességeket, és saját kódot kell készíteni.

Egy webes alkalmazás első felében nagy segítséget nyújt a GyorsVáz. A kezdeti adatbázis-sémák nagyon képlékenyek, és állandóan változnak, ami teljesen természetes egy tervezési folyamat elején. Ennek van egy hátránya: a webfejlesztő nem szívesen készít űrlapokat, amelyeknek nincs tényleges haszna. A fejlesztő erőlködésének könnyítése végett lett a scaffolding képesség a Cake-be beépítve.

A GyorsVáz folyamat megvizsgálja, analizálja az adatbázis tábláit, és egy listát készít szerkesztés céljából, hozzáad, töröl, és módosít gombokkal, illetve a űrlapokat, valamint

nézeteket, melyekkel az egyes adatbáziselemeket lehet vizsgálni. Ha GyorsVáz képességet szeretnénk alkalmazásunkhoz adni, a `$scaffold` változót a vezérlőhöz kell adnunk:

```
<?php
class CategoriesController extends ApplicationController
{
    var $scaffold;
}
?>
```

Egy fontos dolog, amit a GyorsVázzal kapcsolatban érdemes megjegyezni: a tábla minden olyan mezője, amely `_id`-vel végződik, az külső kulcs egy táblához, amelynek neve megegyezik az aláhúzás jelet megelőző karaktersorral.

Például, ha van egy beágyazott kategória, akkor feltehetően van egy mező `parent_id` elnevezéssel. Ebben az esetben, a legjobb megoldás az lenne, ha átneveznénk `parentid`-nek. Akkor is, amikor egy tábla egy külső kulccsal rendelkezik (például a címek tábla egy `category_id` mezőt tartalmaz), és a modelljeink megfelelően vannak összekapcsolva, egy lenyíló menü automatikusan fel lesz töltve adatokkal, amelyeket a kapcsolt táblából (`category`) vesz a `show/edit/new` nézetekben. A kapcsolt táblából választandó mező beállításához a `$displayField` változót kell a kapcsolt modellben beállítani. A kategória (`category`) példánk folytatásaként beállítjuk a címek modellt:

```
<?php
class Title extends AppModel {
    var $name = 'Title';
    var $displayField = 'title';
}
?>
```

Nem ajánljuk ezt a technikát éles alkalmazások esetén, de egy nézet ilyen jellegű testreszabása nagyon hasznos lehet egyszerűbb esetekben.

Amennyiben meg szeretnénk változtatni a GyorsVáz nézeteit, a saját nézeteinket kell feltölteni:

Tetszőleges Scaffolding nézet egy egyszerű vezérlőhöz

A `PostsController` szokványos scaffolding nézetét ehhez hasonlóan kell elhelyezni:

```
/app/views/posts/scaffold/index.scaffold.thtml
/app/views/posts/scaffold/show.scaffold.thtml
/app/views/posts/scaffold/edit.scaffold.thtml
/app/views/posts/scaffold/new.scaffold.thtml
```

Szokásos Scaffolding nézetek az egész alkalmazás számára

Szokásos scaffolding nézetet így kell megadni, minden vezérlő számára:

```
/app/views/scaffold/index.scaffold.thtml
/app/views/scaffold/show.scaffold.thtml
/app/views/scaffold/edit.scaffold.thtml
/app/views/scaffold/new.scaffold.thtml
```

Ha ennél a pontnál a vezérlőlogikát is meg szeretnénk változtatni, itt az ideje a GyorsVáz elvetésének, és elkezdni magunk felépíteni. A Cake kódgenerátorának nagyon hasznos képessége a Bake. A Bake megcsinálja nekünk a GyorsVáz kódokat, amit magunk is testre szabhatunk az alkalmazásunkhoz illesztve.

11.2.5 Nézetek

A Nézet egy oldal sablonja, általában az esemény (action) alapján elnevezve. Például a `PostsController::add()` nézete itt lenne megtalálható:

```
/app/views/posts/add.html.
```

A Cake nézetek sima PHP fájlok, szóval bármilyen PHP kódot használhatunk bennük. Bár a legtöbb nézet fájl leginkább HTML kódot fog tartalmazni, azért ne feledjük, hogy a nézet bármilyen módon ábrázolhat egy bizonyos adathalmazt, akár XML, képek, stb. formában is.

A nézet sablonban hozzáférhetünk a hozzátartozó modell adataihoz. A modell adatai egy `$data` nevű tömbben érhetőek el. Ezen felül bármilyen extra adat, amit a nézetnek a `set()` függvénnyel adtunk át a vezérlőben, szintén elérhető.

Egy HTML segítő (helper) minden nézetből automatikusan elérhető, hisz a nézetekben ez a leginkább használt segítő. Ez nagyon hasznos űrlapok készítésénél, beleértve a szkripteket és egyéb médiákat, valamint link készítésénél és az űrlap ellenőrzésénél. A további részletek megtalálhatók a Segítők fejezetben.

A legtöbb nézetekben használható függvényt ilyen segítők szolgáltatják. A Cake rengeteg segítővel érkezik, és mi is megírhatjuk a sajátunkat. Mivel a nézeteknek nem szabadna semmiféle alkalmazáslogikát tartalmazniuk, nincs túl sok közvetlenül használható (publikus) függvény a nézet osztályban. Egy hasznos azonban akad, a `renderElement()`, amit a következő részben tárgyalunk részletesebben.

Megjegyzés: nem szabad összekeverni az alkalmazáslogikát és a megjelenítési logikát. A megjelenítéslogikára példa az, hogy egy oktató adatait megkapva nem biztos, hogy személyes honlapra mutató linket is kap a nézet. Ekkor nem csak a link elemnek, hanem a „Honlap” feliratnak, sőt akár egy bejegyzésnek is hiányozni kell.

Oldaltervek (Layouts)

Az Oldalterv tartalmazza az összes megjelenítéssel kapcsolatos kódot, ami egy nézet köré épül. Bármilyen, amit minden egyes nézetben látni szeretnénk, az Oldaltervbe kell legyen pakolva.

Az Oldalterv fájlok itt találhatóak: `/app/views/layouts`. A Cake alapértelmezett Oldaltervét felül lehet bírálni a `/app/views/layouts/default.html` módosításával. Amint az alapértelmezett Oldalterv elérhető, a vezérlő nézetének kódja ebbe kerül, amikor az oldal generálásra megtörténik.

Amikor egy Oldaltervet készítünk, a Cake-nek tudnia kell, hogy hova tegye ezen belül a vezérlő a nézet kódját. Szóval bizonyosodjunk meg róla, hogy az Oldalterv tartalmazza a `$content_for_layout` (és esetleg a `$title_for_layout`) változókat. Íme egy példa:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title><?php echo $title_for_layout?></title>
  <link rel="shortcut icon" href="favicon.ico"
        type="image/x-icon">
</head>
```

```

<body>
  <div id="header">
    <div id="menu">...</div>
  </div>

  <?php echo $content_for_layout ?>

  <div id="footer">...</div>
</body>
</html>

```

Az Oldalterv címét a legegyszerűbb a vezérlőben beállítani, a `$pageTitle` változót használva.

```

class UsersController extends AppController {
    function viewActive() {
        $this->pageTitle = 'View Active Users';
    }
}

```

Annyi Oldaltervet készíthetünk a Cake alkalmazásunkhoz, amennyit csak szeretnénk. Mindegyiket a `app/views/layouts` könyvtárba kell tenni. Ezek után pedig a vezérlőben kapcsolgathatunk köztük a vezérlő `$layout` változójával, vagy a `setLayout()` függvényt használva.

Alapelemek

Sok alkalmazás apróbb prezentációs kódot tartalmaz, amit oldalról-oldalra meg kell ismételni, néha az Oldalterv más és más részén. A Cake segítségével ezeket az ismétlődő kódrészleteket könnyedén lehet kezelni. Az ismétlődő, többször használatos részeket alapelemeknek nevezzük. Reklámok, segítő szövegek, navigációs elemek, extra menük gyakran vannak Cake alapelemekként megvalósítva. Egy alapelem tulajdonképpen egy mini-nézet, amit a többi nézetben fel lehet használni.

Az alapelemek a `/app/views/elements/` könyvtárban találhatóak, és `.html` a kiterjesztésük.

Egy alapelemnek kiinduláskor nincs hozzáférése semmilyen adathoz. A szükséges adatokat egy hash-tömbként adhatjuk át neki.

Alapelem hívása paraméterek nélkül:

```

<?php echo $this->renderElement('helpbox'); ?>

```

Alapelem hívása adatokkal:

```

<?php echo
    $this->renderElement('helpbox',
        array("helptext" => "Oh, this text is very helpful."));
?>

```

Az alapelem-ben az átadott változókat a hash kulcsainak segítségével érhetjük el (kb. úgy, mint ahogy a vezérlő `set()` függvénye működik a nézetekkel). A fenti példában a `/app/views/elements/helpbox.html` használhatja a `$helptext` változót.

Az alapelemek összességében a nézetet sokkal átláthatóbbá tehetik, az ismétlődő részeket egy külön fájlba áthelyezve. Bizonyos sűrűn használt tartalmi részeket ide helyezve is segíthetnek még.

11.2.6 Komponensek

A komponenseket a vezérlők bizonyos körülmények közötti segítésére használjuk. A Cake belső könyvtárainak bővítése helyett speciális funkciókat komponensekké alakíthatunk.

A komponensek kis megosztható "vezérlőcskék". A fő cél az újrafelhasználhatóság. Egy kis analógiával a komponens az a vezérlőnek, mint a segítő a nézetnek. A lényegi különbség az, hogy a komponensek **üzleti logikát**, míg a segítők **prezentációs logikát** tartalmaznak.

Az újdonsült fejlesztők (Baker-ek) számára, gyakori kérdés, amikor az újrafelhasználhatóságot próbálják elérni: X dolgot próbálok megvalósítani, most ez komponens vagy segítő legyen? A válasz egész egyszerű: Mit csinál? Üzleti, prezentációs, vagy mindkét logikát használja-e X? Ha üzleti logikát, akkor komponens. Ha prezentációs logikát, akkor segítő. Ha mindkettőt, akkor komponens is és segítő is egyben.

Egy példa lenne az utóbbira egy hitelesítő (autentikációs) rendszer. Szeretnénk, ha be/ki lehetne lépni, korlátozni az erőforrásokhoz való hozzáférést (egy esemény: szerkesztés, hozzáadás, törlés) - mindez üzleti logika - szóval ez egy komponens kell legyen. Ugyanakkor szeretnénk néhány bejegyzést hozzáadni a főmenühez, amikor egy felhasználó be van jelentkezve - ez pedig prezentációs logika.

Saját komponens készítése

Komponens készítéséhez készítsünk egy új fájlt az `app/controllers/components/` könyvtárban.

Tételezzük fel, hogy készítettünk egy `foo.php` nevű fájlt. Ebben a fájlban készítenünk kell egy osztályt, ami megfelel a fájl nevének (hozzáfűzve a `'Component'` végződést). Ebben az esetben ezt kell írunk:

```
class FooComponent extends Object {
    var $someVar = null;
    var $controller = true;

    function startup(&$controller)
    {
        //...
    }

    function doFoo()
    {
        $this->someVar = 'foo';
    }
}
```

Hogy ezt használni is tudjuk, a vezérlőhöz hozzá kell adni az alábbi kódot:

```
| var $components = array('Foo');
```

A komponens most már használhatjuk is így:

```
| $this->Foo->doFoo();
```

Egy komponens hozzáférhet az őt betöltő vezérlőhöz a `startup()` metódus segítségével, ahogy fentebb láthattuk. Ez rögtön a `Controller::beforeFilter()` lefutása után hívódik meg. Így a `beforeFilter()`-ben beállíthatunk különféle komponens tulajdonságokat, amik meghatározzák, mi történjen majd a `startup()` metódusban.

Ha a modelleinket szeretnénk a komponenseinkben használni, létrehozhatunk egy új példányt így:

```
| $foo =& new Foo();
```

Használhatunk más komponenseket egy komponensen belül is. Pusztán deklarálnunk kell, milyen más komponenseket szeretnénk elérni:

```
| var $components = array('Session');
```

Komponenseid publikálása

Ha úgy érezzük, hogy komponensünk hasznos lehet mások számára is, hozzáadhatjuk a CakeForge-hoz. Egy komponens, ami egyre több ember számára hasznos a közösségben, egy nap a fő Cake disztribúcióban is benne lehet. Ellenőrizzük a snippet (kódmorzsák) archívumot is a felhasználók által beküldött komponensekért.

Források

A keretrendszerek közötti választáshoz néhány forrás:

- http://en.wikipedia.org/wiki/List_of_web_application_frameworks#Pseudo-frameworks
- http://www.phpwact.org/php/mvc_frameworks
- http://www.mustap.com/phpzone_post_73_top-10-php-mvc-frameworks
- http://ajaxpatterns.org/PHP_Ajax_Frameworks

12. További szerver platformok

Bár a webfejlesztéssel ismerkedők számára ma legnépszerűbb a PHP nyelv és platformmal való ismerkedés, természetesen több más platformnak is jelentős szerepe van az iparban. Így tehát szükséges, hogy a webfejlesztő többféle platform esetén is megállja a helyét.

Ebben a fejezetben mindössze rövid bevezetést fogunk adni a ma jelentősebb platformokról. Nem fogunk szót ejteni a szemlátomást hanyatlóban lévő vagy nem túl sok jövővel kecsegtető alternatívákról. (Hogy 5-10-20 év múlva milyen platformon fogunk fejleszteni, ma lényegében megjósolhatatlan.)

12.1. J2EE

A Java kliens oldali (asztali alkalmazásokhoz szükséges) alapszolgáltatásokat nyújtó J2SE csomagja mellett a szerver oldali J2EE csomag is rendkívül fontossá vált az ipari alkalmazások tekintetében. Elsősorban nem kis méretű, hanem kimondottan nagy méretű alkalmazások esetén a legkomolyabb piaci szereplők között szerepel.

Források

- Bittologatók: Java tanfolyam
<http://www.cs.ubbcluj.ro/~bittologatok/>
- JavaServer Pages programozóknak
<http://javasite.bme.hu/dokument/jsp/JsvsServerPages.pdf>
- Bilicki Vilmos: Fejlett Programozási Technológiák II.
http://www.inf.u-szeged.hu/~bilickiv/fpt_2006_1/index.html

12.2. Microsoft .NET

A .NET a Microsoft web-stratégiája. Egy olyan új infrastruktúrát és keretrendszert jelent, amellyel hagyományos webalkalmazások fejlesztése mellett webszolgáltatások kiszolgálását is célul tűzik ki.

A .NET a következő Internetes szabványokra épül:

- HTTP, kommunikációs protokoll az internetes alkalmazások között
- XML, a cserélődő adatok formátuma az internetes alkalmazások között
- SOAP, szabványos formátum a web szolgáltatások kiszolgálására
- UDDI, szabvány a Web szolgáltatások keresésére és feltárására

12.2.1 A .NET Keretrendszer

A .NET keretrendszer egy infrastruktúra az új Microsoft .NET felület számára. A .NET keretrendszer egy közös környezet a web szolgáltatások és web alkalmazások megalkotására, telepítésére és futtatására.

A .NET keretrendszer tartalmaz olyan közös osztálykönyvtárakat- úgy mint az ADO.-NET, ASP.NET és a Windows Forms-t - amelyel támogatja azokat az alapvető szolgáltatásokat amik különböző számítógépes rendszerekbe épülhetnek be.

A .NET keretrendszer egy semleges nyelvezet. Jelenleg a következő nyelveket támogatja: C++, C#, Visual Basic, JScript (a Microsoft verziójú JavaScript), Cobol. "Third-party" programozási nyelvek, mint az Eiffel, Perl, Python, Smalltalk stb.- szintén rendelkezésre állnak majd a jövőbeni .NET keretrendszerű alkalmazásokhoz.

Az új Visual Studio.NET egy közös fejlesztői környezet az új .NET keretrendszer számára.

Végrehajtási környezetet biztosít a tulajdonságokban gazdag alkalmazásoknak és az egyszerűsített fejlesztéseknek. Továbbá könnyű egységbe rendezést biztosít számos különböző fejlesztési nyelv között.

12.2.2 A .NET építőkövei

A .NET építőkövei internet szolgáltatások halmaza

Web Szolgáltatások

Web Szolgáltatások adatokat és szolgáltatásokat biztosítanak más alkalmazások számára.

A jövőbeli alkalmazások a webes szolgáltatásokat szabványos web formátumokon (HTML,XML és SOAP) keresztül érik majd el, anélkül, hogy tudniuk kéne, hogyan is van maga a web szolgáltatás megvalósítva.

Webszolgáltatások a fő építőelemei a Microsoft .NET programozási modellnek.

Szabványos Kommunikáció

Hivatalos web szabványokat (XML, UDDI, SOAP) fognak használni annak leírására, hogy mi az internetről jövő adat, és hogy milyen webszolgáltatás tudja megvalósítani.

A jövőbeli webes alkalmazások olyan rugalmas szolgáltatásokra épülnek majd, amik az adatok sértetlensége nélkül tudnak majd információt cserélni.

Internet tárolás

.NET biztonságos és elérhető helyeket kínál az adatok és alkalmazások weben való tárolására.

Ezek a webszolgáltatások a Microsoft létező NTFS, SQL Server és Exchange technológiájára épülnek.

Internet dinamikus szállítás

Megbízható automatikus igény szerinti frissítés és telepítés az alkalmazástól függetlenül.

A .NET támogatni fogja a gyors alkalmazásfejlesztést, amit dinamikusan újra konfigurálhatunk.

Internetes azonosítás

A .NET több különböző szintű hitelesítési szolgáltatást támogat, mint például a jelszó, a "wallet" vagy a mágneskártya.

Ezek a szolgáltatások a már létező Microsoft Passport és Windows Hitelesítési technológiákra épülnek.

Internetes üzenettovábbítás

A .NET támogatja az üzenetek egyesülését. Az e-mail, voice-mail és fax egy egységes internet szolgáltatásban egyesül megcélózva ezzel a pc-ket és minden internet elérésre alkalmas "smart" eszközt.

Ezek a szolgáltatások a már létező Hotmail, Exchange és Instant Messenger technológiákra épülnek.

Internetes Naptár

A .NET támogatja a munkahelyi, a társas és magán naptárak internetes egyesítését, valamint engedélyezi ezeknek az elérését minden internet elérésre alkalmas eszköz által (pc-k, palmtopok, telefonok).

Ezek szolgáltatások a már létező Outlook és Hotmail technológiákra épülnek.

Internetes könyvtár szolgáltatások

A .NET támogatja az új fajta könyvtárszolgáltatásokat amik válaszolni tudnak XML alapú kérdésekre az internet szolgáltatásokról, sokkal pontosabban mint a kereső motorok vagy a "yellow pages".

Ezek a szolgáltatások UDDI szabványra épülnek.

12.2.3 .NET Szoftver

.NET szabványok, technológiák és fejlesztőeszközök összessége

Windows.NET

Ma a Windows 2000 és a Windows XP formálja a .NET gerincét. A jövőben a .NET infrastruktúrája a Microsoft minden operációs rendszerébe, desktop és szerver termékbe be lesz integrálva.

A Windows.NET a következő generációs Windows. Támogatást biztosít minden .NET építőkőre és .NET digitális média részére.

A Windows.NET öntámogató rendszerű lesz az Internet frissítések révén a felhasználók igényei szerint.

Office.NET

A Microsoft Office egy új verziója az Office.NET, ami egy olyan új .NET architektúrával rendelkezik, ami internetes klienseken és web szolgáltatásokon alapszik.

A Office.NET segítségével a böngészés, kommunikáció, dokumentum kezelés és jogsultság kezelés egy XML alapú környezetbe lesz beintegrálva, amely lehetővé teszi a felhasználók részére, hogy az interneten raktározzák el a dokumentumaikat.

ASP.NET

ASP.NET a legfrissebb verziója az ASP-nek. Tartalmaz alkalmazásokhoz kötődő web szolgáltatásokat, szolgáltatásokat és eszközöket amik a HTTP, HTML, XML és SOAP technológiát használnak.

Újdonságok az ASP.NET-ben:

- új nyelv támogatás
- programozható vezérlők
- esemény vezérelt programozás
- XML alapú komponensek
- Felhasználó hitelesítés
- Felhasználói profilok
- Magas skálázhatóság
- Fordítási kód
- Könnyű konfigurálhatóság
- Könnyű telepíthetőség
- ASP-n kívüli kompatibilitás
- tartalmazza az ADO.NET-t

Visual Studio.NET

A legutóbbi kiadása a Visual Studionak- a Visual Studio.NET- magába foglalja az ASP.-NET, ADO.NET, Web Services, Web Forms, valamint nyelvi újításokat a Visual Basic részére. A fejlesztő eszközök erős XML támogatással rendelkeznek valamint XML alapú programozási modellt és új objektum-orientált programozási lehetőséget kínálnak.

Visual Basic.NET

A Visual Basic.NET nyelvi erősítést kapott, ami által egy teljesen objektum-orientált programozási nyelvvé vált.

12.2.4 SQL Server 2000

SQL Server 2000 egy teljesen webes adatbázis rendszer.

SQL Server 2000 erős XML és HTTP támogatással rendelkezik, amik a .NET két fő infrastruktúrális technológiáját képviselik.

Néhány a legfontosabb SQL Server jellemző közül: közvetlen adatbázis elérés böngészőből, relációs adatok lekérdezése XML-ből és XML-be való kiírása relációs formában.

Internet Information Services 6.0

IIS 6.0 több, szerveren működtethető programozási nyelvet támogat, így képessé téve az új web alkalmazásokat, hogy bármilyen platformon és bármilyen böngészőben fussanak.

12.3. Python

Források

- Web alkalmazások Python nyelven – alapok
<http://www.gluon.hu/node/41>
- Gérard Swinnen : Tanuljunk meg programozni Python nyelven
<http://learnpython.openproject.hu/>

12.4. Ruby on Rails

„A Ruby on Rails egy integrált, nyílt forrású, Ruby nyelvű webes keretrendszer, mellyel a legtöbb hasonló keretrendszerhez képest élvezetesebben és tömörebben készíthetünk webalkalmazásokat.”

A RoR fiozófiája ma a leggyakrabban másolt webalkalmazás-logika.

Források

- <http://www.rubyonrails.hu/>

13. Tartalomkezelő rendszerek

A CMS (Content Management System, magyarul: tartalomkezelő rendszer) általában kevés szaktudást igénylő internetes alkalmazás, mely lehetővé teszi, hogy szinte bárki összetett weboldalt birtokoljon.

Az alkalmazás jellegéből adódóan a weboldal rövid idő alatt összeáll, programozási tudást nem, vagy egészen keveset igényel.

Napjainkban a legtöbb fejlesztő cég nyílt forrású, vagy éppen saját fejlesztésű keretrendszert, vagy éppen CMS rendszert alkalmaz az újrafelhasználhatóság jegyében.

Magyar nyelven a Drupal⁶⁹ és a Joomla⁷⁰ talán a legjobban dokumentált, és a legnagyobb közösséggel rendelkező CMS, az ismerkedést érdemes talán ezekkel kezdeni. A további ingyenes CMS-ek érdekében érdemes a következő oldalakat meglátogatni:

- <http://cms.lap.hu/>
- <http://opensourcecms.com/>

Ez utóbbi oldal mind az átlag felhasználó, mind az adminisztrációs felület működésébe betekintést nyújt működő demók formájában.

⁶⁹ <http://drupal.hu/>

⁷⁰ <http://joomla.hu/>

14.Álláskereső próbafeladatok

Ebben a fejezetben néhány fejlesztő cég állásajánlata található. Állást keresve érdemes felmérni saját tudásunkat, másrészt a hiányos ismeretek köre is jól behatárolható segítségükkel.

14.1. Egyszerű beléptető-rendszer

A következő feladat jellemzőnek tekinthető a **PHP és adatbázis** programozó munkakör betöltéséhez. Komoly PHP és adatbázis ismeretekre van szükség a megoldáshoz. A feladat megfogalmazásából látszik, hogy a kliens oldali felület kialakítása nem hangsúlyos, valószínűleg annak fejlesztése nem lesz a feladat része.

A feladat

PHP: csak a belépés felületét kell elkészíteni, regisztrációhoz, adatmódosításhoz, törléshez elég csak az SQL lekérdezések listája.

A belépéshez szükséges adatokon kívül tárolva legyen a belépő neve és e-mail címe.

Minden felhasználót lehessen csoportokba (akár többbe is) rakni. Ehhez adminisztrációs felület nem szükséges, csak adatbázis terv, és a használathoz szükséges SQL lekérdezések.

Tegyük fel, hogy ez egy meghívásos rendszer. Hogyan tartsuk nyilván, hogy az adott felhasználót ki hívta meg, és kik az általa meghívott felhasználók?

Az SQL utasítások a MySQL vagy PostgreSQL utasításkészletet használják! Ha nincs teljes adatbázis dump (nem szükséges) akkor a táblák indexelését is meg kell adni.

A tervezés során elsődleges szempontok a biztonság és a sebesség. A beléptető kódjára és az SQL lekérdezésekre, illetve táblajellemzésekre lennénk kíváncsiak.

Ötletek

Egy lehetséges beléptető-rendszer egyes részei a korábbiakban már bemutatásra kerültek. Az adatbázis-tervezésre épülő rész kidolgozása (jelen verzióban) az olvasóra marad.

14.2. Dinamikus űrlapgeneráló osztály

A következő példa a webfejlesztés igen nagy palettáját lefedi: objektumorientált PHP, HTML, CSS, sablonkezelés, és adatbázis-kezelés. Egyik téma sem kap nagyon komoly szerepet, de a sokféle részfeladat miatt mégis igen komplex, nagy rálátást igénylő feladat.

A viszonylag részletes leírás ellenére is vannak olyan pontok, ahol egy fejlesztői munkakör esetén mindenképpen további részleteket kellene megtudni a projekt vezetőjétől.

A feladat

Írjon olyan űrlapgeneráló osztályt, ami megadott bemenő paraméterek alapján legenerál egy komplett, működő HTML űrlapot. Az űrlap végső alakját egy sablon fájlból vegye. Az űrlapot és a mezőket beágyazott CSS2 segítségével formázza.

Az űrlap legenerálásához szükséges adatokat adatbázisban tárolja. (Az adatbázis feltöltéséhez nem kell programot írni.) A legenerált űrlapot HTML fájlként tárolja el.

- Az űrlap paraméterei:
- címe,
- bevezető szövege,
- *action*,
- *method*,
- az űrlap elküldése után megjelenő ablak új ablakban jelenjen-e meg,
- az űrlapon megjelenő mezők adatai,
- a *submit* gomb felirata.

Az űrlapon megjelenő mezők paraméterei:

- sorszám (hanyadikként jelenjen meg);
- mező leírása (az elem előtt megjelenő szöveg);
- mezőnév (az űrlap name paramétere);
- mező hossza (szöveges mező esetén), hosszú szöveg esetén sorok és oszlopok száma.
- a mező típusa lehet: szöveges (text); hosszú szöveg (textarea); checkbox.

A megvalósításnál a fő szempont a modularitás, objektumorientált felépítés, és a paramétereizhetőség.

A megvalósítás technikai feltételei:

- Az osztályt PHP nyelven kell megírni.
- A Smarty⁷¹-t kell használni
- Adatbázis kezelést a PEAR::DB osztályon keresztül kell megvalósítani. Adatbázis bármilyen SQL relációs adatbázis-kezelő lehet (MySQL4 preferált).

A megoldott feladat az alábbiakat kell, hogy tartalmazza:

- Az űrlap generáló osztály forráskódja.
- Az űrlap adatait eltároló SQL tábla vagy táblák leírása (SQL).
- Egy vagy több sablon fájl, és a hozzá tartozó küldő CSS fájl.
- Bemelő teszt adatok, – legalább háromféle –, párosítva a kimenő adatokkal (HTML fájlok).

Ötletek

A feladat leírásából nem derül ki, hogy itt a munkaadó mit is preferálna: ha egyéni megoldás készítésével a tudását csillogtatná, vagy (közel) kész megoldás alkalmazásával, testreszabásával a hatékony munkavégzést demonstrálná az álláskereső.

Mivel egy konkrét munka esetén is érdemes ezt a projektvezetővel egyeztetni, nem szégyen az álláshirdetőt ez ügyben megkeresni.

⁷¹ <http://smarty.php.net>

Itt most csupán néhány megoldás címe következik, amelyek (felületes tájékozódás alapján) megfelelőnek látszanak az elinduláshoz:

- <http://www.phpformclass.com/>
- <http://www.devshed.com/c/a/PHP/The-Singleton-and-Factory-Patterns-in-PHP-Building-a-Form-Generator-Class/>
- <http://codewalkers.com/seecode/335.html>

14.3. Címjegyzék alkalmazás

Szükséges funkciók:

- kategóriák felvitele, módosítása, törlése
- címzettek felvitele
 - kategóriák kiválasztása, kötelezően legalább egy, de több is lehet
 - név, e-mail cím, felvitel időpontja kötelező mezők
 - mobiltelefonszám opcionális, de ha megadják, nemzetközi formátumban (36x01112222, x = 2,3,7) kötelező
- címzettek böngészése lapozható, a mezők alapján rendezhető listában
- címzettek módosítása, törlése
- kategóriák között létezik egy alapértelmezett ("besorolatlan" néven), ami nem törölhető. Ha olyan kategóriát törölnek, amelyben már vannak címzettek, azok ide kerülnek át.

Természetesen minden "best practice" megoldás alkalmazása előnyt jelent (pl. valid HTML/CSS; ha JavaScript-et használsz, diszkrét JavaScript; form használhatósági megoldások; stb).

Beállítások: `safe_mode=on`, `magic_quotes_gpc=off`, `error_reporting=E_ALL`. Szükség esetén PHP4 vagy 5 ág legújabb verzió választható.

Ötletek

Néhány nyílt forráskódú megoldás, ami alapján el lehet indulni:

- <http://www.corvalis.net/address/>
- http://php.about.com/od/finishedphp1/ss/address_book.htm
- <http://www.codythefreak.net/oab.php>

14.4. Álláshirdetések

Végül néhány jellemző álláshirdetés következik, amelyek a tantárgy során bevezetett témákra épülnek.

PHP fejlesztő

Feladat: A cég által gondozott weboldalak és adminisztrációs felületek tervezése, programozása.

Elvárások

- magas szintű PHP programozási ismeret és tapasztalat
- SQL ismeret
- HTML és CSS szabványok ismerete
- javascript alapos ismerete

Előnyt jelent

- grafikai programok használatában szerzett jártasság

Joomla fejlesztő

Feladat: Joomla portálok testreszabása és design illesztése.

- Új vagy meglévő Joomla portálok testreszabása
- Skinek definiálása, egyeztetések a designerekkel
- Designerek által készített HTML alapú látványterv Joomla-hoz igazítása
- Modulok telepítése

PHP fejlesztő

Elvárások

- szakirányú felsőfokú végzettség
- proaktivitás, önállóság, gyors tanulás
- minimum 2 éves releváns fejlesztői tapasztalat
- objektum orientált fejlesztői szemlélet
- PHP(5) beható ismerete
- SQL ismeret
- XML/XSL ismeret
- CSS 1-2; XHTML beható ismerete
- AJAX ismeretek
- Linux felhasználói szintű ismeretek
- angol nyelvtudás dokumentáció-értelmezés szinten

Előnyök

- Linux rendszeradminisztrációs ismeretek
- Smarty, AdoDB vagy hasonló technológiák ismerete
- PEAR ismerete
- alkalmazás terhelés-optimalizálási tapasztalat
- SQL terhelés-optimalizálási tapasztalat
- e-Learning technológiák, SCORM szabvány ismerete
- Java fejlesztési tapasztalat
- Oracle DB tapasztalat
- tervezési minták használata [design pattern]

- creole vagy egyéb ORM [object relation mapping] eszköz ismerete
- tárgyalóképes angol nyelvtudás

User Interface Developer

User Interface Developer feladatkörbe keresünk munkatársakat DHTML, AJAX, Flash, OpenLaszlo, és Adobe Flex technológiákra épülő Web 2.0 és RIA (Rich Internet Application) fejlesztésekhez.

PHP fejlesztő

- többéves fejlesztői tapasztalat
- a web és a PHP biztonsági kérdéseinek és megoldásainak ismerete
- (x)HTML, CSS
- SQL adatbáziskezelő(k) ismerete
- Linux tapasztalat (alapfokú)
- Smarty, AdoDB vagy hasonló technológiák ismerete előnyt jelent
- Jó angol szövegértés
- Működő referenciák

Webes alkalmazásfejlesztő

Feladat

- PHP-s keretrendszer fejlesztés, webes alkalmazások,
- Weboldalak, portálok specifikálása, fejlesztése,
- Meglévő weboldalak módosítása, továbbfejlesztése.

Elvárások

- többéves fejlesztői tapasztalat,
- PHP5
- (x)HTML, CSS,
- SQL adatbáziskezelő(k) ismerete
- MySQL
- XML (XSL előny)
-

Előnyt jelent

- PostgreSQL adatbáziskezelőkben való jártasság
- JS, DOM 2.0
- AJAX ismerete
- Template rendszerek ismerete (elsősorban Smarty)
- Crossbrowser problémák megoldásában tapasztalat

HTML programozó

Elvárások

- több éves szakmai gyakorlat:
- HTML, XHTML, DHTML, JavaScript, AJAX, CSS
- kreativitás
- megbízhatóság
- teljes munkaidős vállalás!
- csapatmunka

Előnyök

- JSP, ASP.NET ismeret
- angol nyelvtudás
- különböző platform ismeretek (Linux, Mac OSX)

AJAX/Ruby fejlesztő

Webes szolgáltatásainkhoz keresünk AJAX/Ruby fejlesztőt.

Kezdő és haladó Java fejlesztő

Elvárásaink

- Legalább 1 éves szakmai tapasztalat
- Java fejlesztési tapasztalat
- Internetes technológiák ismerete
- Alapszintű angol nyelvtudás

Előnyök

- J2EE, JSP fejlesztési tapasztalat
- Adatbázis tervezési, felhasználási tapasztalat
- Csoport- és projektvezetési tapasztalat
- Angol nyelv tárgyalás szintű ismerete

Kezdő és haladó .NET fejlesztő

Elvárásaink

- Legalább 1 éves szakmai tapasztalat
- ASP.NET fejlesztési tapasztalat
- Internetes technológiák ismerete
- Alapszintű angol nyelvtudás

Előnyök

- .NET C# fejlesztési tapasztalat
- Adatbázis tervezési, felhasználási tapasztalat

- SharePoint, BizTalk, InfoPath eszközök ismerete
- Csoport- és projektvezetési tapasztalat
- Angol nyelv tárgyalás szintű ismerete