

Matt Zandstra

Tanuljuk meg a PHP4 használatát 24 óra alatt

SAMS

Matt Zandstra



Tanuljuk meg

**a
PHP4
használatát**



24 óra alatt

A kiadvány a következő angol eredeti alapján készült:

Matt Zandstra: Teach Yourself PHP4 in 24 Hours

Copyright © 2000 by Sams Publishing. All rights reserved. No part of this book, including interior design, cover design, and icons, may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of the publisher.

Acquisitions Editor: *Jeff Schultz*

Project Editor: *Paul Schneider*

Development Editor: *Scott D. Meyers*

Copy Editor: *Geneil Breeze*

Managing Editor: *Charlotte Clapp*

Trademarked names appear throughout this book. Rather than list the names and entities that own the trademarks or insert a trademark symbol with each mention of the trademarked name, the publisher states that it is using the names for editorial purposes only and to the benefit of the trademark owner, with no intention of infringing upon that trademark.

A szerzők és a kiadó a lehető legnagyobb körültekintéssel járt el e kiadvány elkészítésekor. Sem a szerző, sem a kiadó nem vállal semminemű felelősséget vagy garanciát a könyv tartalmával, teljességével kapcsolatban. Sem a szerző, sem a kiadó nem vonható felelősségre bármilyen baleset vagy káresemény miatt, mely közvetve vagy közvetlenül kapcsolatba hozható e kiadvánnyal.

Fordítás és magyar változat © 2001 Szy György, Kiskapu Kft. Minden jog fenntartva!

Translation and Hungarian edition © 2001 György Szy, Kiskapu Kft. All rights reserved!

Sorozatszerkesztő: *Szy György*

Műszaki szerkesztő: *Csutak Hoffmann Levente*

Nyelvi lektor: *Rézműves László*

Szakmai lektor: *Hojtsy Gábor*

Felelős kiadó a Kiskapu Kft. ügyvezető igazgatója

© 2001 Kiskapu Kft.

1081 Budapest Népszínház u. 29.

Tel: (+36-1) 477-0443 Fax: (+36-1) 303-1619

<http://www.kiskapu.hu/>

e-mail: kiskapu@kiskapu.hu

ISBN: 963 9301 30 2

Készült a debreceni Kinizsi nyomdában

Felelős vezető: *Bördös János*

Apámnak, akinek ez a könyv biztosan tetszett volna...

A szerzőről

Matt Zandstra (matt@corrosive.co.uk) a Corrosive Web Design céget vezeti (<http://www.corrosive.co.uk/>) üzleti partnerével, Max Guglielminóval. Mint a parancsnyelvekért rajongó programozó, PHP, Java, JavaScript, Perl, Lingo és AppleScript nyelveken fejlesztett különböző programokat. Először bölcsészdiplomát szerzett, majd úgy tanulta ki mesterségét, hogy „újra feltalálta a kereket” és kielemezte, miért nem halad egyenesen. HTML, JavaScript, Perl és PHP tanfolyamokon oktatott, majd szerzőként közreműködött a *Dynamic HTML Unleashed* című könyvben. Amikor nem kódot ír, Matt elkötelezett városi biciklista, Guinness-ivó, megszállott olvasó és novellista, aki kiadhatatlan történeteket alkot. Azt állítja, egyszer talán még regényt is fog írni.

Köszönetnyilvánítás

A nyílt forráskódú programoknak köszönhetem karrieremet és e könyv létrejöttét is. Szeretném megköszönni mindazoknak, akiknek önzetlen segítőkészsége mindig felülmúlja a kapott elismerést.

Különösen köszönöm a PHP közösségnek, legfőképpen a PHP levelezőlisták résztvevőinek, akik leveleikkel feltárták a buktatókat, programozási módszereket ajánlottak és mindig ámulatba ejtettek.

A Macmillantól szeretném megköszönni Randi Rogernek, hogy engem ajánlott e könyv szerzőjéül, valamint Jeff Schultznak, Paul Schneidernek és Scott Meyersnek, támogatásukért és megértésükért, ahogy a határidő közeledett és kitört a pánik.

Köszönet illet mindenkit a Corrosive-nél, hogy eltűrték folyamatos hiányzásomat és nagyfokú határozatlanságomat a nem PHP-vel kapcsolatos kérdésekben. Legfőképpen üzlettársamnak, Massimo Guglieminónak vagyok hálás, hogy működtette a Corrosive céget a megterhelő körülmények között. Köszönet Dave Urmsonnak is, aki átvette a formázást, amikor a munka megkívánta. A Corrosive további munkatársai: Anisa Swaffield, Jeff Coburn, Mai Chokelumlerd és Moira Govern.

Meg kell köszönnöm a Small Planetnek (<http://www.smallpla.net/>), hogy további fejlesztési teret adtak nekem és engedélyezték a bétaprogramok kipróbálását. Kifejezetten hálás vagyok Mohamed Abbának és Clive Hillsnek, hogy oly sokszor újrafordították a PHP-t a Small Planet kiszolgálón.

Az oktatási anyagok kipróbálásának egyik legjobb módja a tanfolyamon való felhasználás. Köszönet tanulóimnak, akik kegyesen beleegyeztek, hogy kísérleti nyulak legyenek.

Köszönöm kedvesemnek, Louise-nek, és lányunknak, Hollynak, hogy jelen voltak és elviselték a könyv írása alatt kialakult zsörtölődő, visszahúzó és megszállott jellememet.

Amint magánéletem a PHP után második helyre került, menedékemmé a törzshegyem vált, ahol összekapcsoltam a sörözést a munkával. Köszönet Alannek és Dorának a kifogástalan Prince Arthur ivó fenntartásáért.

Végül köszönet a halaknak, akik mindig felvidítanak, ha bámulom őket.

Áttekintés

Bevezető

I. rész Az első lépések

1. óra	PHP: személyes honlaptól a portálig	3
2. óra	A PHP telepítése	11
3. óra	Első PHP oldalunk	23

II. rész A PHP nyelv

4. óra	Az alkotóelemek	35
5. óra	Vezérlési szerkezetek	57
6. óra	Függvények	75
7. óra	Tömbök	97
8. óra	Objektumok	119

III. rész Munka a PHP-vel

9. óra	Űrlapok	149
10. óra	Fájlok használata	171
11. óra	A DBM függvények használata	193
12. óra	Adatbázisok kezelése – MySQL	211
13. óra	Kapcsolat a külvilággal	237
14. óra	Dinamikus képek kezelése	259
15. óra	Dátumok kezelése	283
16. óra	Az adatok kezelése	301
17. óra	Karakterláncok kezelése	319
18. óra	A szabályos kifejezések használata	339
19. óra	Állapotok tárolása sütikkel és GET típusú lekérdezésekkel	361
20. óra	Állapotok tárolása munkamenet-függvényekkel	381
21. óra	Munka kiszolgálói környezetben	395
22. óra	Hibakeresés	409

IV. rész Összefoglaló példa

23. óra	Teljes példa (első rész)	429
24. óra	Teljes példa (második rész)	469
A függellék	Válaszok a kvízkérdésekre	489

Tárgymutató

Tartalomjegyzék

Bevezető

I. rész Az első lépések

1. óra PHP: személyes honlaptól a portálig 3

Mi a PHP?	4
A PHP fejlődése	4
A PHP 4 újdonságai	5
A Zend Engine	6
Miért a PHP?	7
A fejlesztés sebessége	7
A PHP nyílt forráskódú	7
Teljesítmény	8
Hordozhatóság	8
Összefoglalás	8
Kérdések és válaszok	8
Műhely	9
Kvíz	9
Feladatok	9

2. óra A PHP telepítése 11

Operációs rendszerek, kiszolgálók, adatbázisok	12
A PHP beszerzése	13
A PHP 4 telepítése Apache webkiszolgálót használó Linuxra	13
A configure néhány paramétere	14
--enable-track-vars	15
--with-gd	15
--with-mysql	15
Az Apache beállítása	16
php.ini	17

short_open_tag	18
Hibajelentések beállításai	18
Változókra vonatkozó beállítások	19
Segítség!	19
Összefoglalás	21
Kérdések és válaszok	21
Műhely	22
Kvíz	22
Feladatok	22

3. óra Első PHP oldalunk 23

Első programunk	24
PHP blokkok kezdése és befejezése	26
A print() függvény	27
HTML és PHP kód egy oldalon	28
Megjegyzések a PHP kódokban	30
Összefoglalás	30
Kérdések és válaszok	31
Műhely	31
Kvíz	31
Feladatok	31

II. rész A PHP nyelv

4. óra Az alkotóelemek 35

Változók	36
Dinamikus változók	37
Hivatkozások a változókra	39
Adattípusok	41
Típus módosítása a setType() segítségével	43
Típus módosítása típusátalakítással	44
Műveletjelek és kifejezések	45
Hozzárendelés	46
Aritmetikai műveletek	47
Összefűzés	47
További hozzárendelő műveletek	47
Összehasonlítás	48
Bonyolultabb összehasonlító kifejezések létrehozása	
logikai műveletek segítségével	49

Egész típusú változók értékének növelése és csökkentése	51
A műveletek kiértékelési sorrendje	52
Állandók	53
Minden programban használható állandók	54
Összegzés	54
Kérdések és válaszok	55
Műhely	55
Kvíz	55
Feladatok	56
 5. óra Vezérlési szerkezetek	57
Elágazások	58
Az if utasítás	58
Az if utasítás else ága	59
Az if utasítás elseif ága	60
A switch utasítás	62
Ciklusok	65
A while ciklus	66
A do..while ciklus	67
A for ciklus	68
Ciklus elhagyása a break utasítás segítségével	69
Következő ismétlés azonnali elkezdése	
a continue utasítás segítségével	71
Egymásba ágyazott ciklusok	72
Összegzés	73
Kérdések és válaszok	73
Műhely	74
Kvíz	74
Feladatok	74
 6. óra Függvények	75
Mit nevezünk függvénynek?	76
Függvények hívása	76
Függvények létrehozása	78
Függvények visszatérési értéke	80
Dinamikus függvényhívások	81
Változók hatóköre	82
Hozzáférés változókhoz a global kulcsszó segítségével	84
Állapot megőrzése a függvényhívások között	
a static kulcsszó segítségével	86
Paraméterek további tulajdonságai	89
Paraméterek alapértelmezett értéke	89

Hivatkozás típusú paraméterek	91
Összegzés	93
Kérdések és válaszok	94
Műhely	94
Kvíz	94
Feladatok	95

7. óra Tömbök 97

Mit nevezünk tömbnek?	98
Tömbök létrehozása	99
Tömbök létrehozása az array() függvény segítségével	99
Tömb létrehozása vagy elem hozzáadása a tömbhöz szögletes zárójel segítségével	99
Asszociatív tömbök	100
Asszociatív tömbök létrehozása az array() függvény segítségével	101
Asszociatív tömbök létrehozása és elérése közvetlen értékadással	101
Többdimenziós tömbök	102
Tömbök elérése	104
Tömb méretének lekérdezése	104
Tömb bejárása	104
Asszociatív tömb bejárása	106
Többdimenziós tömb bejárása	108
Műveletek tömbökkel	110
Két tömb egyesítése az array_merge() függvény segítségével	110
Egyszerre több elem hozzáadása egy tömbhöz az array_push() függvény segítségével	110
Az első elem eltávolítása az array_shift() függvény segítségével	112
Tömb részének kinyerése az array_slice() függvény segítségével	113
Tömbök rendezése	113
Számmal indexelt tömb rendezése a sort() függvény segítségével	114
Asszociatív tömb rendezése érték szerint az asort() függvény segítségével	115
Asszociatív tömb rendezése kulcs szerint a ksort() függvény segítségével	116
Összegzés	117
Kérdések és válaszok	117
Műhely	117
Kvíz	117
Feladatok	118

8. óra	Objektumok	119
	Mit nevezünk objektumnak?	120
	Objektum létrehozása	121
	Objektumtulajdonságok	121
	Az objektumok tagfüggvényei	122
	Egy bonyolultabb példa	126
	Az osztály tulajdonságai	126
	A konstruktor	127
	Az <code>ujSorO</code> tagfüggvény	127
	Az <code>ujNevesSorO</code> tagfüggvény	128
	A <code>kiirO</code> tagfüggvény	129
	Összeáll a kép	129
	Ami még hiányzik...	132
	Miért használjunk osztályt?	132
	Öröklés	133
	A szülő tagfüggvényeinek felülírása	134
	A felülírt tagfüggvény meghívása	135
	Egy példa az öröklésre	137
	A <code>HTMLTablázat</code> saját tulajdonságai	137
	A konstruktor	137
	A <code>cellaMargoAllitO</code> tagfüggvény	138
	A <code>kiirO</code> tagfüggvény	138
	A <code>Tablázat</code> és a <code>HTMLTablázat</code> osztályok	
	a maguk teljességében	139
	Miért alkalmazzunk öröklést?	142
	Összegzés	143
	Kérdések és válaszok	144
	Műhely	144
	Kvíz	144
	Feladatok	145

III. rész Munka a PHP-vel

9. óra	Űrlapok	149
	Globális és környezeti változók	150
	Adatok bekérése a felhasználótól	152
	Több elem kiválasztása a <code>SELECT</code> elemmel	153
	Az űrlap minden mezőjének	
	hozzárendelése egy tömbhöz	155
	Különbségek a <code>GET</code> és a <code>POST</code> továbbítás között	157

PHP és HTML kód összekapcsolása egy oldalon	159
Állapot mentése rejtett mezőkkel	161
A felhasználó átirányítása	163
Fájlfeltöltő űrlapok és programok	165
Összegzés	169
Kérdések és válaszok	169
Műhely	170
Kvíz	170
Feladatok	170

10. óra Fájlok használata 171

Fájlok beágyazása az include() függvénnyel	172
Fájlok vizsgálata	175
Fájl létezésének ellenőrzése a file_exists() függvénnyel	175
Fájl vagy könyvtár?	176
Fájl állapotának lekérdezése	176
Fájl méretének lekérdezése a filesize() függvénnyel	177
Különbféle fájlinformációk	177
Több fájl tulajdonságot egyszerre megadó függvény	178
Fájlok létrehozása és törlése	179
Fájl megnyitása írásra, olvasásra, hozzáfűzésre	180
Olvasás fájlból	181
Sorok olvasása fájlból az fgets() és feof() függvényekkel	181
Tetszőleges mennyiségű adat olvasása fájlból	182
Fájl karakterenkénti olvasása az fgetc() függvénnyel	184
Fájlba írás és hozzáfűzés	185
Fájlok zárolása az flock() függvénnyel	186
Munka könyvtárakkal	187
Könyvtár létrehozása az mkdir() függvénnyel	187
Könyvtár törlése az rmdir() függvénnyel	188
Könyvtár megnyitása olvasásra	188
Könyvtár tartalmának olvasása	188
Összegzés	190
Kérdések és válaszok	190
Műhely	190
Kvíz	190
Feladatok	191

11. óra A DBM függvények használata 193

DBM adatbázis megnyitása	194
Adatok felvétele az adatbázisba	195
Adatok módosítása az adatbázisban	196

Adatok kiolvasása DBM adatbázisból	197
Elemek meglétének lekérdezése	199
Elem törlése az adatbázisból	199
Összetett adatszerkezetek tárolása	
DBM adatbázisban	199
Egy példa	203
Összefoglalás	209
Kérdések és válaszok	209
Műhely	209
Kvíz	209
Feladatok	210

12. óra Adatbázisok kezelése – MySQL 211

(Nagyon) rövid bevezetés az SQL nyelvbe	212
Csatlakozás a kiszolgálóhoz	213
Az adatbázis kiválasztása	214
Hibakeresés	215
Adatok hozzáadása táblához	216
Automatikusan növekvő mező	
értékének lekérdezése	220
Adatok lekérdezése	221
Az eredménytábla sorainak száma	221
Az eredménytábla elérése	222
Adatok frissítése	225
Információk az adatbázisokról	227
Az elérhető adatbázisok kiírása	227
Adatbázistáblák listázása	229
Információk a mezőkről	229
Az adatbázis szerkezete – összeáll a kép	230
Összefoglalás	233
Kérdések és válaszok	234
Műhely	234
Kvíz	234
Feladatok	235

13. óra Kapcsolat a külvilággal 237

Környezeti változók	238
A HTTP ügyfél-kiszolgáló kapcsolat	
rövid ismertetése	241
A kérés	241
A válasz	243
Dokumentum letöltése távoli címről	245

Átalakítás IP címek és gépnevek között	246
Hálózati kapcsolat létesítése	247
NNTP kapcsolat létrehozása az fsockopen()-nel	251
Levél küldése a mail() függvénnyel	254
Összefoglalás	255
Kérdések és válaszok	255
Műhely	256
Kvíz	256
Feladatok	257

14. óra Dinamikus képek kezelése 259

Képek létrehozása és megjelenítése	260
A szín beállítása	261
Vonalak rajzolása	261
Alakzatok kitöltése	263
Körív rajzolása	264
Téglalap rajzolása	265
Sokszög rajzolása	266
A színek átlátszóvá tétele	268
Szövegek kezelése	269
Szövegírás az imageTTFtext() függvénnyel	269
Szöveg kiterjedésének ellenőrzése	271
az imageTTFbbox() függvénnyel	271
A fenti elemek összegyűrése	275
Összefoglalás	280
Kérdések és válaszok	280
Műhely	281
Kvíz	281
Feladatok	281

15. óra Dátumok kezelése 283

A dátum kiderítése a time() függvénnyel	284
Az időbélyeg átalakítása a getdate() függvénnyel	284
Az időbélyeg átalakítása a date() függvénnyel	286
Időbélyeg készítése az mktime() függvénnyel	288
A dátum ellenőrzése a checkdate() függvénnyel	289
Egy példa	290
A felhasználó által bevitt adatok ellenőrzése	290
A HTML űrlap létrehozása	291
A naptár táblázatának létrehozása	294
Összefoglalás	298

Kérdések és válaszok	298
Műhely	299
Kvíz	299
Feladatok	299
16. óra Az adatok kezelése	301
Újra az adattípusokról	302
Egy kis ismétlés	302
Összetett adattípusok átalakítása	303
Az adattípusok automatikus átalakítása	304
Az adattípusok ellenőrzése	306
Az adattípus-váltás további módjai	306
Miért olyan fontosak az adattípusok?	307
A változók meglétének és ürességének ellenőrzése	309
További tudnivalók a tömbökről	310
Tömbök bejárása más megközelítésben	310
Elem keresése tömbben	312
Elemek eltávolítása a tömbből	312
Függvények alkalmazása a tömb összes elemére	312
Tömbök egyéni rendezése	314
Összefoglalás	317
Kérdések és válaszok	317
Műhely	318
Kvíz	318
Feladatok	318
17. óra Karakterláncok kezelése	319
Karakterláncok formázása	320
A printf() függvény használata	320
A printf() és a típusparaméterek	321
A kitöltő paraméter	323
A mezőszélesség meghatározása	325
A pontosság meghatározása	326
Átalakító paraméterek (Ismétlés)	326
Formázott karakterlánc tárolása	328
Részletesebben a karakterláncokról	329
Szövegek indexelése	329
Szöveg hosszának megállapítása az strlen() függvénnyel	329
Szövegrész megkeresése az strstr() függvénnyel	330
Részlánc elhelyezkedésének meghatározása az strpos() függvénnyel	330

Szövegrészlet kinyerése a substr() függvénnyel	331
Karakterlánc elemekre bontása az strtok() függvénnyel	331
A karakterláncok kezelése	333
Szöveg tisztogatása a trim() típusú függvényekkel	333
Karakterlánc részének lecserélése	
a substr_replace() függvénnyel	334
Az összes részlánc lecserélése az str_replace() függvénnyel	334
Kis- és nagybetűk közti váltás	335
Karakterláncok tömbbé alakítása az explode() függvénnyel	336
Összefoglalás	336
Kérdések és válaszok	337
Műhely	337
Kvíz	337
Feladatok	338

18. óra A szabályos kifejezések használata . . . 339

A POSIX szabályos kifejezések függvényei	340
Minta keresése karakterláncokban az ereg() függvénnyel	340
Egynél többször előforduló karakter keresése	
mennységjelzővel	341
Karakterlánc keresése karakterosztályokkal	343
Az atomok kezelése	344
Elágazások	345
A szabályos kifejezés helye	345
A tagazonosítót kereső példa újragondolása	346
Minták lecserélése karakterláncokban	
az ereg_replace() függvénnyel	347
Visszaulrás használata az ereg_replace() függvénnyel	347
Karakterláncok felbontása a split() függvénnyel	348
Perl típusú szabályos kifejezések	348
Minták keresése a preg_match() függvénnyel	349
A Perl típusú szabályos kifejezések és a mohóság	349
A Perl típusú szabályos kifejezések és	350
a fordított perjeles karakterek	350
Teljeskörű keresés a preg_match_all() függvénnyel	351
Minták lecserélése a preg_replace() függvénnyel	354
Módosító paraméterek	355
Összefoglalás	358
Kérdések és válaszok	358
	358
Műhely	359
Kvíz	359
Feladatok	359

19. óra **Állapotok tárolása süttikkel és GET típusú lekérdezésekkel 361**

Süti	362
A süti felépítése	362
Süti beállítása a PHP-vel	363
Süti törlése	365
Munkamenet-azonosító süti	366
Példa: Webhelyhasználat nyomon követése	366
Lekérdező karakterláncok használata	374
Lekérdező karakterlánc készítése	375
Összefoglalás	378
Kérdések és válaszok	378
Műhely	378
Kvíz	378
Feladatok	379

20. óra **Állapotok tárolása munkamenet-függvényekkel 381**

Mik azok a munkamenet-függvények?	382
Munkamenet indítása	
a session_start() függvénnyel	382
Munkamenet-változók	384
A munkamenet és a változók	
bejegyzésének törlése	389
Munkamenet-azonosítók a lekérdező	
karakterláncban	390
Munkamenet-változók kódolása és visszafejtése	390
Munkamenet-változó bejegyzésének ellenőrzése	391
Összefoglalás	392
Kérdések és válaszok	392
Műhely	393
Kvíz	393
Feladatok	393

21. óra **Munka kiszolgálói környezetben 395**

Folyamatok összekötése a popen() függvénnyel	396
Parancsok végrehajtása az exec() függvénnyel	399
Külső programok futtatása	
a system() függvénnyel vagy	
a ' művelettel segítségével	401
Biztonsági rések megszüntetése	
az escapeshellcmd() függvény használatával	401

Külső programok futtatása	
a passthru() függvénnyel	404
Külső CGI program meghívása	
a virtual() függvénnyel	405
Összefoglalás	406
Kérdések és válaszok	407
Műhely	407
Kvíz	407
Feladatok	408

22. óra Hibakeresés 409

A phpinfo()	410
A forráskód megjelenítése színtkiemeléssel	413
PHP hibaüzenetek	415
Hibaüzenetek kiírása naplófájlba	418
A hibaüzenet elfogása	420
Kézi hibakeresés	421
Gyakori hibák	423
Összefoglalás	425
Kérdések és válaszok	425
Műhely	425
Kvíz	425
Feladatok	426

IV. rész Összefoglaló példa

23. óra Teljes példa (első rész) 429

A feladat rövid leírása	430
Az oldalak felépítése	430
Az adatbázis kialakítása	431
Tervezési döntésünk	433
A tagoknak szánt oldalak	433
csatlakozas.php és adatbazis.inc	434
klubfrissites.php	442
tagmenu.php	449
belepes.php	450
esemenyfrissites.php	453
esemenylista.php	462
Összefoglalás	466
Kérdések és válaszok	466

Műhely	467
Kvíz	467
Feladatok	467

24. óra Teljes példa (második rész) 469

Az eseménynaptár nyilvános oldalai	470
esemenyekinfo.php	470
klubokinfo.php	478
klubinfo.php	481
esemenyinfo.php	484
A jövő	486
Összefoglalás	487
Kérdések és válaszok	488
Műhely	488
Kvíz	488
Feladatok	488

A függellék Válaszok a kvízkérdésekre. 489

Tárgymutató

Bevezető

Ez a könyv egy nyílt forráskódú webes parancsnyelvről (szkriptről), a PHP-ről szól, amely csatlakozott a Perl-höz, az ASP-hez és a Javához a dinamikus webes alkalmazások készítéséhez rendelkezésre álló nyelvek palettáján. A kötet programozási ismeretekkel is foglalkozik. A rendelkezésre álló lapokon nem jut elég hely egy teljes PHP programozási útmutató közlésére vagy a PHP összes lehetőségeinek és eljárásának ismertetésére, mindazonáltal a könyvben található lépések elég információt adnak ahhoz, hogy használni kezdhessük a PHP-t, akár rendelkezünk programozói tapasztalattal, akár újak vagyunk a parancsnyelvek világában.

Kiknek szól ez a könyv?

A könyv az alaptól indulva hasznos gyakorlati tudást ad a PHP 4-es programozási nyelv használatához. Semmilyen korábbi programozási tapasztalatra nincs szükség, de ha a C vagy a Perl nyelvekkel már dolgoztunk korábban, az egyes órákon könnyebb lesz haladni.

A PHP 4 webes programozási nyelv. Ahhoz, hogy a lehető legtöbb hasznát vegyük a könyvnek, célszerű némi ismerettel rendelkezni a Világhálóval és a HTML-lel kapcsolatban. Ha nem rendelkezünk ilyen ismeretekkel, akkor is hasznos lehet e könyv, ám meggondolandó egy HTML ismertető beszerzése. Ha kényelmesen létre tudunk hozni egyszerű HTML dokumentumokat táblázatokkal, akkor elegendő tudással rendelkezünk.

A PHP 4-esben az adatbázisok kezelése igen egyszerű. A könyv néhány példájában a MySQL ingyenes adatbázisrendszert használtuk. Az SQL nyelvet röviden ismertetjük, de ha komolyabban kívánjuk használni az adatbáziskezelő szolgáltatásokat, célszerű elmélyednünk néhány kapcsolódó anyagban. Az Interneten számos bevezető szintű SQL ismertető érhető el. Ha mégsem a MySQL adatbázisrendszerrel kívánunk dolgozni, a könyv példáit könnyen más adatbázisokhoz igazíthatjuk.

Könyvünk szerkezete

Kötetünk négy fő részből áll:

- Az első rész bevezető a PHP alapjaihoz.
- A második rész az alapvető szolgáltatásokat mutatja be. Ha még nincs programozási tapasztalatunk, ezt a részt különös figyelemmel olvassuk!

- A harmadik rész részletebben ismerteti a PHP 4-es változatának lehetőségeit, felsorakoztatva a függvényeket és megoldásokat, melyekre szükségünk van, ha gyakorlott PHP programozók szeretnénk lenni.
- A negyedik rész egy teljesen önállóan működő példaprogramot mutat be.

Az első rész az elsőtől a harmadik óráig tart és egy egyszerű parancsfájl futtatásáig vezeti el az olvasót:

- Az első óra „PHP: személyes honlaptól a portálig” címmel bemutatja a PHP történetét és képességeit, valamint a PHP tanulása mellett néhány érvet sorol fel.
- A második óra „A PHP telepítése” címmel végigvezeti az olvasót a PHP telepítésén UNIX rendszeren, valamint azon fordítási és beállítási lehetőségekkel foglalkozik, amelyek a környezet kialakítása szempontjából fontosak lehetnek.
- A harmadik óra „Első PHP oldalunk” címmel bemutatja, hogyan építhetünk PHP kódot HTML oldalainkba és hogyan készíthetünk a böngésző számára kimenetet adó programot.

A második részben a negyedikől a nyolcadik óráig megismerjük a PHP alapvető elemeit:

- A negyedik óra „Az alkotóelemek” címmel a PHP alapjait mutatja be. Az óra anyagát változók, adattípusok, műveletjelek (operátorok) és kifejezések képezik.
- Az ötödik óra „Vezérlési szerkezetek” címmel a programok futását vezérlő elemek utasításformájával (nyelvtanával) foglalkozik. Az if és switch szerkezetek után megtanuljuk a for és while ciklusvezérlő elemek használatát is.
- A hatodik óra „Függvények” címmel a függvények készítését és használatát tárgyalja.
- A hetedik óra „Tömbök” címmel a lista jellegű adatok tárolására használható tömb adattípussal foglalkozik, valamint a tömbök használatához néhány PHP 4 függvényt is ismertet.
- A nyolcadik óra „Objektumok” címmel bemutatja a PHP 4 osztály- és objektumtámogatását. Ebben az órában egy működő példán keresztül vesszük górcső alá az objektumok használatát.

A harmadik részben a kilencediktől a huszonkettedik óráig alaposan megismerjük a nyelv szolgáltatásait és megoldási módszereit:

- A kilencedik óra „Űrlapok” címmel a HTML űrlapok használatát, vagyis a felhasználótól érkező adatok feldolgozását vezeti be. Megtanuljuk, miként érjük el a beérkező információkat.
- A tizedik óra „Fájlok használata” címmel bemutatja a fájlok és könyvtárak kezelésének lehetőségeit.

- A tizennegyedik óra „A DBM függvények használata” címmel a PHP DBM-támogatásával foglalkozik, amely a legtöbb operációs rendszeren elérhető.
- A tizenkettedik óra „Adatbázisok kezelése – MySQL” címmel az SQL alapjait tárgyalja, valamint bemutatja a PHP 4 MySQL adatbázisok kezelésére szolgáló függvényeit.
- A tizenharmadik óra „Kapcsolat a külvilággal” címmel a HTTP kéréseket veszi szemügyre, illetve a PHP hálózati függvényeit ismerteti.
- A tizennegyedik óra „Dinamikus képek kezelése” címmel a GIF, PNG és JPEG képek készítését lehetővé tevő függvényeket mutatja be.
- A tizenötödik óra „Dátumok kezelése” címmel a dátumműveletekhez használatos függvényeket és eljárásokat ismerteti. Ebben az órában egy naptár példát is elkészítünk.
- A tizenhatodik óra „Az adatok kezelése” címmel visszatér az adattípusokhoz és az addig nem említett, de a hatékony használathoz elengedhetetlen további függvényeket mutatja be. A tömbökkel kapcsolatban is újabb függvényeket említünk.
- A tizenhetedik óra „Karakterláncok kezelése” címmel a karakterláncok kezeléséhez jól használható PHP függvényekkel foglalkozik.
- A tizennyolcadik óra „A szabályos kifejezések használata” címmel bemutatja a bonyolultabb minták keresésére és cseréjére szolgáló szabályos kifejezéseket.
- A tizenkilencedik óra „Állapotok tárolása sütikkel és GET típusú lekérdezésekkel” címmel a különböző programok és HTTP kérések közötti információátadás néhány módját mutatja be.
- A huszadik óra „Állapotok tárolása munkamenet-függvényekkel” címmel az előző órában tanultakat a PHP 4-es beépített munkamenet-kezelő függvényeivel bővíti ki.
- A huszonegyedik óra „Munka kiszolgálói környezetben” címmel a külső programok futtatását és kimenetük felhasználásának lehetőségeit mutatja be.
- A huszonkettedik óra „Hibakeresés” címmel a kódok hibakeresésére ad ötleteket, valamint bemutat néhány szokásos hibát.

A negyedik részben – a huszonharmadik és huszonnegyedik órákon – a könyvben tanult módszerek felhasználásával egy működő példát készítünk.

- A huszonharmadik óra egy tervet mutat be helyi klubok számára készülő programunkhoz. Megírjuk a felhasználók bejegyzéséhez és a klubok programjának beviteléhez szükséges elemeket.
- A huszonnegyedik óra befejezésként a látogatók számára készült felület megvalósításához szükséges kódot tartalmazza, amely lehetővé teszi a klubok programjainak böngészését.

A könyvvel kapcsolatos észrevételeiket a <http://www.kiskapu.hu/konyvek/PHP4/> címen tehetik meg, ugyancsak ezen az oldalon található a hibajegyzék is.



I. RÉSZ

Az első lépések

- 1. óra PHP: személyes honlaptól a portálig
- 2. óra A PHP telepítése
- 3. óra Első PHP oldalunk



1. ÓRA

PHP: személyes honlaptól a portálig

Üdvözlét a PHP világában! Ebben a könyvben végigtekintjük a PHP nyelv majdnem minden elemét. Mielőtt azonban részletesebben megnéznénk, mire lehetünk képesek segítségével, tárjuk fel múltját, főbb tulajdonságait és jövőjét.

Ebben az órában a következőket tanuljuk meg:

- Mi a PHP?
- Hogyan fejlődött a nyelv?
- Mik a PHP 4 újdonságai?
- Hogyan tehetjük optimálissá a PHP-t?
- Miért pont a PHP-t válasszuk?

Mi a PHP?

A PHP nyelv túlnőtt eredeti jelentőségén. Születésekor csupán egy makrókészlet volt, amely személyes honlapok karbantartására készült. Innen ered neve is: Personal Home Page Tools. Később a PHP képességei kibővültek, így egy önállóan használható programozási nyelv alakult ki, amely képes nagyméretű webes adatbázis-alapú alkalmazások működtetésére is.

A PHP nyelv népszerűsége képességeinek bővülésével folyamatosan nőtt. A NetCraft elemző cég (<http://www.netcraft.com/>) felmérései szerint a PHP-t 2000 februárjában 1,4 millió kiszolgálón használták és októberre ez a szám 3,3 millióra ugrott. Ezzel megközelítette a Microsoft IIS kiszolgálók számát, ami 3,8 millió. Az E-Soft szerint a PHP a legnépszerűbb Apache modul, a ModPerl is maga mögé utasítva.

A PHP jelenleg hivatalosan a PHP: Hypertext Preprocessor elnevezést használja. Tulajdonképpen kiszolgálóoldali programozási nyelv, amit jellemzően HTML oldalakon használnak. A hagyományos HTML lapokkal ellentétben azonban a kiszolgáló a PHP parancsokat nem küldi el az ügyfélnek, azokat a kiszolgáló oldalán a PHP-értelmező dolgozza fel. A programjainkban lévő HTML elemek érintetlenül maradnak, de a PHP kódok lefutnak. A kódok végezhetnek adatbázis-lekérdezéseket, dinamikusan létrehozhatnak képeket, fájlokat olvashatnak és írhatnak, kapcsolatot létesíthetnek távoli kiszolgálókkal – a lehetőségek száma végtelen. A PHP kódok kimenete a megadott HTML elemekkel együtt kerül az ügyfélhez.

A PHP fejlődése

A PHP első változatát – amely néhány webalkalmazás-készítést segítő makrót tartalmazott – Rasmus Lerdorf készítette 1994-ben. Ezen eszközöket együttesen a Personal Home Page Tools névvel azonosították. Később, a kód újraírása után, egy friss elem került a csomagba, a Form Interpreter (Űrlapfeldolgozó), így PHP/FI néven vált ismertebbé. A felhasználók szemszögéből a PHP/FI nagyon hasznos segédeszköz volt, így népszerűsége töretlenül nőtt. Több fejlesztő is felfigyelt rá, így 1997-re már számos programozó dolgozott rajta.

A következő kiadás, a PHP 3-as, már egy csapat együttműködéséből született. Ehhez a változathoz Zeev Suraski és Andi Gutmans újáalkotta a teljes feldolgozóegységet, valamint újabb elemeket és szabályokat adott a nyelvhez. Ez a változat megalapozottá tette a PHP helyét a legjobb kiszolgálóoldali nyelvek között, így felhasználói tábora rendkívüli mértékben nőtt.

Az Apache- és MySQL-támogatás megerősítette a PHP pozícióját. Az Apache jelenleg a legnépszerűbb kiszolgáló a világon és a PHP 3-as már modulként illeszthető hozzá. A MySQL igen hatékony, ráadásul ingyenes SQL adatbázisrendszer, amelyhez a PHP átfogó támogatást nyújt. Az Apache-MySQL-PHP együttes egyszerűen verhetetlen.

Ez természetesen nem jelenti azt, hogy a PHP nem használható más környezetben, más eszközökkel. A PHP számos adatbázis-alkalmazással és webkiszolgálóval képes együttműködni.

A PHP népszerűségének növekedésére hatással volt a webes alkalmazások fejlesztésében történt váltás is. Az 1990-es évek közepén természetesnek számított, hogy akár egy nagyobb webhelyet is több száz, egyenként kézzel kódolt HTML lap felhasználásával készítsenek el. Mára azonban a fejlesztők egyre inkább kihasználják az adatbázisok nyújtotta kényelmi szolgáltatásokat, hogy a megjelenítendő tartalmat hatékonyan kezeljék és az egyes felhasználóknak lehetőséget adjanak a webhelyek testreszabására.

Egyre gyakoribb adatbázisok használata a tartalom tárolására és az információk visszakeresésére különböző felületeken. Az adatok egy központból több környezetbe is érkehetnek, beleértve a mobiltelefonokat, a digitális személyi titkárokat (PDA), digitális televíziókat és szélessávú internetes rendszereket is.

Ebben a környezetben már nem meglepő, hogy egy ilyen kifinomult és rugalmas nyelv, mint a PHP, ekkora népszerűsége tett szert.

A PHP 4 újdonságai

A PHP 4-es változata számos – a programozók életét megkönnyítő – új szolgáltatással rendelkezik. Nézzük ezek közül a legfontosabbakat:

- A Perl nyelvben találhatóhoz hasonló új foreach vezérlési szerkezet, ami leegyszerűsíti a tömbökön végrehajtandó ciklusok készítését. Ezt fogjuk használni a könyv legtöbb tömbbel kapcsolatos példájában. Ezen túl számos új tömbkezelő függvény került a nyelvbe, amelyek megkönnyítik a tömbökkel végzett műveleteket.
- A nyelv tartalmazza a boolean (logikai) adattípust.
- A PHP 3 felettébb hasznos szolgáltatása volt, hogy a HTML űrlap elemeit tömbnevekkel láthattuk el, így ezek neve és értéke a program számára egy tömbként került átadásra. Ez a szolgáltatás a többdimenziós tömbök támogatásával bővült.
- A PHP 3 csak kezdetleges szinten támogatta az objektumközpontú programozást. Ezen a téren is jelentős fejlesztés történt, a PHP 4-esben például már lehetséges egy felülírt metódus meghívása egy leszármazott osztályból.

- A PHP 4-be beépítették a felhasználói munkamenetek (session) támogatását is. Ezek kezelése sütik (cookie) vagy GET metódusú lekérdezések (query string) formájában történhet. Lehetőségünk van változókat rendelni egy munkamenethez és más oldalakon újra elérni ezeket.
- Két új összehasonlító műveletet vezettek be (=== és !==), melyekkel egyidőben értékek és típusok egyezését, illetőleg nem egyezését is ellenőrizhetjük.
- A kiszolgálói és környezeti adatok tárolására új „beépített” asszociatív tömböket hoztak létre, valamint egy új változót, amelyből információkat kaphatunk a feltöltött fájl(ok)ról.
- A PHP 4-es beépített támogatással rendelkezik a Java és XML nyelvekhez.

Ezek és más új szolgáltatások ugyan jelentősen bővítették a nyelvet, de a legfontosabb változás a felszín alatt következett be.

A Zend Engine

A PHP 3 készítésekor az alapoktól indulva teljesen új feldolgozóegységet írtak a nyelvhez. A PHP 4-esben hasonló változás figyelhető meg a programokat futtató magban, ez azonban jelentősebb.

A Zend Engine a PHP modulok mögött található, a programokat futtató mag elnevezése. Kifejezetten a teljesítmény jelentős növelésére fejlesztették ki.

A hatékonysági változások minden bizonnyal biztosítani fogják a PHP további sikerét. A PHP 3-as változata számára készült kódok legnagyobb része minden módosítás nélkül tovább működik, sőt, akár 200-szoros sebességgel futhat.

A Zend Technologies Ltd. (<http://www.zend.com/>) egyik kereskedelmi fejlesztése a PHP kódok fordítását teszi lehetővé. Ez további teljesítménynövekedést jelent, amivel a mérések szerint a PHP messze maga mögött hagyja legtöbb versenytársát.

A Zend Engine a teljesítmény és a rugalmasság növelésére íródott. A kiszolgálókapcsolatok továbbfejlesztésével lehetővé vált, hogy olyan PHP modulok készüljenek, amelyek a kiszolgálók széles körével képesek együttműködni. Míg CGI-feldolgozóként minden lekéréshez új PHP-értelmezőt kell elindítani, addig modulként a PHP folyamatosan a memóriában van. Ez gyorsabb futást jelent, hiszen nem kell mindig elindítani egy újabb feldolgozóprogramot, ha kérés érkezik.

Miért a PHP?

Van néhány megcáfolhatatlan érv, amiért a PHP 4-est érdemes választani. Ha más programnyelveket is ismerünk, számos alkalmazás fejlesztése során észlelni fogjuk, hogy a programozási szakasz érezhetően gyorsabb, mint várnánk. A PHP, mint nyílt forráskódú termék jó támogatással rendelkezik, amit a képzett fejlesztői gárda és az elkötelezett közösség nyújt számunkra. Ráadásul a PHP a legfontosabb operációs rendszerek bármelyikén képes futni, a legtöbb kiszolgálóprogrammal együttműködve.

A fejlesztés sebessége

Mivel a PHP lehetőséget ad a HTML elemek és a programkódok elkülönítésére, az alkalmazások fejlesztésekor lehetőség van elválasztani a kódolási, tervezési, és összeállítási szakaszt. Ez jelentősen megkönnyíti a programozók életét, azzal, hogy elmozdítja az akadályokat a hatékony és rugalmas alkalmazások kialakításának útjából.

A PHP nyílt forráskódú

Számos felhasználó szemében a „nyílt forráskódú” egyet jelent azzal, hogy ingyenes, ami természetesen már önmagában is előnyös. Egy idézet a PHP hivatalos webhelyéről (<http://www.php.net/>):

ÚJDONSÁG

Talán idegennek hangozhat azok számára, akik nem UNIX-háttérrel olvassák-e sorokat, hogy a PHP nem kerül semmibe. Használható kereskedelmi és/vagy nem kereskedelmi célra, ahogy tetszik. Odaadhatjuk barátainknak, kinyomtathatjuk és felakaszthatjuk a falra vagy akár elfogyaszthatjuk ebédre. Légy üdvözölve a nyílt forráskódú programok világában! Mosolyogj, légy boldog, a világ jó! További információkért lásd a hivatalos licenst.

A jól szervezett nyílt forráskódú projektek újabb előnyökkel szolgálnak a felhasználóknak. Felvehetjük a kapcsolatot a könnyen elérhető és elkötelezett felhasználói közösséggel, ahol számos nagy tapasztalattal rendelkező embert találunk. Nagy az esély rá, hogy bármilyen problémával is kerülünk szembe, némi kutatással gyorsan és könnyen választ találunk rá. Ha mégsem, egy levelezőlistára küldött üzenetre általában hamar érkezik intelligens és hiteles válasz.

Úgyszintén bizonyos, hogy a feldolgozóprogram hibáinak javítása nem sokkal felfedezésük után megtörténik és a felmerült új igényeket kielégítő szolgáltatások is hamar beépülnek a nyelvbe. Nem kell várni a következő hivatalos kiadásra, hogy a fejlesztések előnyeit élvezzük.

Nincs a PHP működtetéséhez egyedileg kiválasztott kiszolgáló vagy operációs rendszer. Szabadon választhatunk olyan rendszert, amely kielégíti saját vagy ügyfeleink igényeit. Biztos, hogy kódunk továbbra is futtatható lesz, bármi mellett is döntünk.

Teljesítmény

A hatékony Zend Engine-nek köszönhetően a PHP 4-es jól vizsgázik az ASP-vel szemben végzett méréseken, néhányban megelőzve azt. A lefordított PHP messze maga mögött hagyja az ASP-t.

Hordozhatóság

A PHP-t alapvetően úgy tervezték, hogy alkalmas legyen számos operációs rendszeren való használatra, együttműködve különböző kiszolgálókkal és adatbázis-kezelőkkel. Fejleszthetünk UNIX rendszerre és áttérhetünk NT alapokra minden probléma nélkül. A PHP alkalmazásokat kipróbálhatjuk Personal Web Serverrel és később telepíthetjük azokat egy UNIX rendszerre, ahol a PHP-t Apache modulként használjuk.

Összefoglalás

Ebben az órában bemutattuk a PHP-t. Láttuk, hogyan alakult át a nyelv egyszerű makrókészletből hatékony programnyelvvé. Megismertük a Zend Engine-t, és megnéztük, milyen új lehetőségeket teremt a PHP 4-es változatában. Végül áttekintettünk néhány tulajdonságot, amelyek ellenállhatatlanná teszik a PHP-t.

Kérdések és válaszok

Könnyű megtanulni a PHP nyelvet?

Röviden: igen! Valóban meg lehet tanulni a PHP alapjait 24 órában! A PHP megszámlálhatatlanul sok függvényt bocsát rendelkezésünkre, melyek megvalósításához más nyelvekben saját kódot kellene írni. A PHP automatikusan kezeli a különböző adattípusokat és memóiafoglalásokat (hasonlóan a Perl-höz).

Egy programozási nyelv nyelvtanának és szerkezeinek megértése azonban csak az út kezdetét jelenti. Végősoron a saját programok készítéséből és a hibák kijavításából lehet igazán sokat tanulni. Ezt a könyvet kiindulópontként érdemes tekinteni.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Mit jelentett eredetileg a PHP betűszó?
2. Ki készítette a PHP első változatát?
3. Mi az új mag neve a PHP 4-es változatában?
4. Soroljuk fel a PHP 4 néhány új szolgáltatását!

Feladatok

1. A könyvet átlapozva annak felépítése alaposabban szemügyre vehető. Gondolkozzunk el a témákon, és azon, hogyan segíthetnek jövőbeni alkalmazásaink elkészítésében.



2. ÓRA

A PHP telepítése

Mielőtt megkezdénénk az ismerkedést a PHP nyelvvel, be kell szerezniünk, telepíteniünk és beállítanunk a PHP-értelmezőt. A feldolgozóprogram számos operációs rendszerre elérhető és több kiszolgálóval is képes együttműködni.

Ebben az órában a következőket tanuljuk meg:

- Milyen platformokat, kiszolgálókat és adatbázisokat támogat a PHP?
- Honnan szerezhetjük be a PHP-t és más nyílt forráskódú programokat?
- Hogyan telepíthető a PHP Linux rendszerre?
- Hogyan állíthatók be a fontosabb paraméterek?
- Hol található segítség, ha nem sikerül a telepítés?

Operációs rendszerek, kiszolgálók, adatbázisok

A PHP teljesen platformfüggetlen, ami azt jelenti, hogy fut Windows operációs rendszeren, a legtöbb UNIX rendszeren – beleértve a Linuxot –, sőt még Macintosh gépeken is. A támogatott kiszolgálók köre igen széles. A legnépszerűbbek: Apache (szintén nyílt forráskódú és platformfüggetlen), Microsoft Internet Information Server, WebSite Pro, iPlanet Web Server, OmniHTTPD és Microsoft Personal Web Server. Az utóbbi kettő akkor tehet nagy szolgálatot, ha internetkapcsolat nélkül szeretnénk fejleszteni, bár az Apache is alkalmas erre Windows környezetben.

A PHP fordítható önálló alkalmazássá is, így az értelmező parancssorból is hívható. Ebben a könyvben webalkalmazások fejlesztéséhez fogjuk használni a PHP-t, de nem szabad alábecsülni a szerepét általános programozói eszközként sem.

A PHP-t alapvetően úgy tervezték, hogy könnyen összhangba hozható legyen a különböző adatbázisokkal. Ez az egyik oka a PHP népszerűségének a webalkalmazások készítése terén. Számos adatbázis – Adabas D, InterBase, Solid, dBASE, mSQL, Sybase, Empress, Microsoft SQL, MySQL, Velocis, FilePro, Oracle, UNIX dbm, Informix és PostgreSQL – közvetlenül csatlakoztatható a PHP-hez. A közvetlenül nem támogatott adatbázisok mellett a PHP-ben kapcsolatot létesíthetünk az ODBC szabványt használó programokkal is.

A könyv példáiban Linux operációs rendszer alatt Apache és MySQL programokat használunk. Ezek ingyenesen letölthetők az Internetről, valamint könnyen telepíthetők és használhatók majdnem minden PC-n. A Linux rendszer telepítéséről további információ a <http://www.linux.org/help/beginner/distributions.html> címen található. A Linux PowerPC gépen is használható, a LinuxPPC rendszerrel: <http://www.linuxppc.org/>. Magyarországon a Linux közösség honlapja a <http://www.linux.hu/>.

A MySQL adatbázisrendszer, amit ebben a könyvben használni fogunk, a <http://www.mysql.com/> címről tölthető le. Számos operációs rendszerre elérhető, beleértve a UNIX, Windows és OS/2 rendszereket. A MySQL magyar tükörkiszolgálója a <http://mysql.sote.hu/>.

Természetesen nyugodtan dolgozhatnánk Windows NT vagy MacOS rendszer alatt is, mivel a PHP platformfüggetlen nyelv.

A PHP beszerzése

A PHP 4-es változata a `http://www.php.net/` címről tölthető le. Mivel a PHP nyílt forráskódú, nem kell a bankkártyánkat kéznél tartanunk, amikor letöltjük az értelmezőt. Magyarországon a `http://hu.php.net/` tükörkiszolgálót érdemes meglátogatni.

A PHP webhelye kiváló információforrás PHP programozóknak. A `http://www.php.net/manual/` címen a teljes kézikönyv elolvasható, kiegészítve más programozók által írt hasznos megjegyzésekkel. Ezen a címen olvasható a magyar PHP kézikönyv is. A PHP honlapról a dokumentáció is letölthető különböző formátumokban.

A Zend Engine és más Zend termékek honlapja a `http://www.zend.com/`. Itt híreket, illetve cikkeket olvashatunk a PHP és a Zend világából.

A PHP 4 telepítése Apache webkiszolgálót használó Linuxra

Ebben a részben végigvezetünk egy PHP telepítést egy Apache webkiszolgálót használó Linux rendszeren. A folyamat többé-kevésbé ugyanez más UNIX platformokon is. Elképzelhető, hogy az általunk használt rendszerre készült előre fordított változat is, így még egyszerűbben telepíthetnénk az értelmezőt, a PHP fordítása azonban nagyobb szabadságot ad a feldolgozóba kerülő szolgáltatásokat illetően.

Mielőtt megkezdjük a telepítést, ellenőrizzük, hogy rendszergazdaként (root) jelentkeztünk-e be a rendszerbe. Ha a kiszolgálót nem érhetjük el root felhasználóként, forduljunk a rendszergazdához a PHP telepítésével kapcsolatos kéréseinkkel.

A PHP-t kétféleképpen lehet Apache modulként előállítani. Egyrészt újrafordíthatjuk a webkiszolgálót és beépíthetjük a PHP-értelmezőt, másrészt a PHP-t dinamikusan megosztott objektumként (DSO, Dynamic Shared Object) is fordíthatjuk. Ha Apache kiszolgálónk DSO-támogatással ellátott, képes lesz az új modul használatára anélkül, hogy újrafordítanánk a programot. Ez a legegyszerűbb módja annak, hogy beüzemeljük a PHP-t, ezért ezt az eljárást fogjuk tárgyalni.

Ha ellenőrizni kívánjuk, hogy az Apache rendelkezik-e DSO-támogatással, el kell indítanunk az Apache futtatható állományát (`httpd`) az `-l` paraméterrel.

```
/www/bin/httpd -l
```

A program ekkor egy listát ad a rendelkezésre álló beépített modulokról. Ha a `mod_so.c` elem szerepel a listában, akkor az Apache alkalmas az alább bemutatott módszerrel történő bővítésre. Egyéb esetben újra kell fordítani, amihez a dokumentáció tartalmazza az összes szükséges információt.

Ha még nem tettük meg, le kell töltenünk a PHP legfrissebb változatát. A `tar` fájl `gzip`-el tömörített, így első lépésben ki kell csomagolnunk:

```
tar -xvzf php-4.x.x.tar.gz
```

Ha sikeresen kibontottuk a csomagot, lépünk át a keletkezett könyvtárba:

```
cd ../php-4.x.x
```

Ebben a könyvtárban található a `configure` program, melynek a megfelelő paraméterekkel megadhatjuk, milyen szolgáltatásokat építsen be a PHP-be. Ebben a példában csak néhány hasznos parancssori paramétert adunk meg, de természetesen számos más lehetőség is rendelkezésre áll. Később megnézzünk néhány további elemet a `configure` paraméterei közül.

```
./configure --enable-track-vars \  
            --with-gd \  
            --with-mysql \  
            --with-apxs=/www/bin/apxs
```

Lehetséges, hogy a `--with-apxs` paraméternek átadott elérési útnak a rendszerünkön másnak kell lennie, mivel telepítéskor az `apxs` esetleg éppen az Apache futtatható állománnyal megegyező könyvtárba került.

Amikor a `configure` lefutott, elindítható a `make` program. Ennek futtatásához a rendszernek tartalmaznia kell egy C fordítót.

```
make  
make install
```

Ezekkel a parancsokkal a PHP fordítása és telepítése befejeződött, már csak az Apache beállítására és futtatására van szükség.

A configure néhány paramétere

Amikor lefuttattuk a `configure`-t, megadtunk néhány parancssori paramétert, melyek meghatározták, milyen lehetőségekkel ruházzuk fel a PHP-t. Ha a kibontott PHP csomag könyvtárában a következő parancsot adjuk ki, a `configure` megadja a lehetséges paramétereket:

```
./configure --help
```

Mivel a lista rendkívül hosszú, célszerű előbb egy szövegfájlba irányítani, így kényelmesebben elolvasható:

```
./configure --help > lehetosegek.txt
```

Annak ellenére, hogy a fenti parancs kimenete eléggé érthető, néhány fontos lehetőséget meg kell említenünk – mégpedig azokat, amelyek a könyv szempontjából számunkra érdekesek.

--enable-track-vars

Ez a szolgáltatás automatikusan előállítja számunkra a PHP oldalakon kívülről érkező adatokhoz tartozó asszociatív tömböket. Ezek a GET vagy POST kéréssel érkezett adatok, a visszaérkező süti-értékek, a kiszolgálói és környezeti változók. A tömbökkel a hetedik órában foglalkozunk bővebben, a HTTP kapcsolatokat a tizenharmadik órában részletezzük. A fenti rendkívül gyakran használt `configure` paraméter, mivel nagyon kellemes lehetőség a beérkező adatok követésére. A PHP 4.0.2-es változatától kezdve mindig be van kapcsolva, így nem kell külön megadni.

--with-gd

A `--with-gd` paraméter engedélyezi a GD könyvtár támogatását. Amennyiben a GD könyvtár telepítve van a rendszeren, ez a paraméter lehetőséget ad dinamikus GIF, JPEG vagy PNG képek készítésére a PHP programokból. A dinamikus képek előállításáról a tizennegyedik órában írunk. Ha a GD-t korábban nem az alapbeállítású könyvtárba telepítettük, megadható a szokásostól eltérő elérési út is:

```
--with-gd=/eleresi/ut/a/megfelelo/konyvtarhoz
```

--with-mysql

A `--with-mysql` engedélyezi a MySQL adatbázisok támogatását. Ha a rendszeren a MySQL nem az alapbeállítású könyvtárban található, megadható a szokásostól eltérő elérési út is:

```
--with-mysql=/eleresi/ut/a/megfelelo/konyvtarhoz
```

Mint már tudjuk, a PHP támogat számos más adatbázisrendszert is. Az 1.2-es táblázatban ezek közül láthatunk néhányat, a hozzájuk tartozó `configure` paraméterekkel.

2.1. táblázat Néhány adatbázis és a hozzá tartozó configure paraméter

<i>Adatbázis</i>	<i>configure paraméter</i>
Adabas D	--with-adabas
FilePro	--with-filepro
mSQL	--with-msql
Informix	--with-informix
iODBC	--with-iodbc
OpenLink ODBC	--with-openlink
Oracle	--with-oracle
PostgreSQL	--with-pgsql
Solid	--with-solid
Sybase	--with-sybase
Sybase-CT	--with-sybase-ct
Velocis	--with-velocis
LDAP	--with-ldap

Az Apache beállítása

Miután sikeresen lefordítottuk az Apache-t és a PHP-t, módosítanunk kell az Apache beállításait tartalmazó `httpd.conf` fájlt. Ez az Apache könyvtárának `conf` alkönyvtárban található.

A következő sorok hozzáadása szükséges:

```
AddType application/x-httpd-php .php .php3
AddType application/x-httpd-php-source .phps
```

Keressünk rá ezekre a sorokra a `httpd.conf` fájlban! Az újabb Apache kiadásokban ez már szerepel, csak egy megjegyzésjelet kell kitörölnünk a sorok elejéről. Ezzel biztosítjuk, hogy a PHP-elemző fel fogja dolgozni a `.php` és `.php3` kiterjesztéssel rendelkező fájlokat. A `.php3` kiterjesztésre azért lehet szükség, mert számos régebbi program ezt használja, így módosítás nélkül tovább alkalmazhatjuk ezeket is.

A `.phps` kiterjesztéssel rendelkező fájlok PHP forrásként kerülnek a böngészőhöz, ami azt jelenti, hogy a forráskód HTML formátumúvá alakul és a nyelvtani elemek színiemeléssel jelennek meg, ami hasznos segítség lehet a programok hibáinak felderítésében. Ha ügyfeleink miatt esetleg a hagyományos oldalaknál megszokott `.html` kiterjesztést választjuk a PHP számára, a következő beállítást kell alkalmaznunk:

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

Tulajdonképpen bármilyen kiterjesztéshez köthetjük a PHP-feldolgozót. Az ajánlott a `.php`, a `.html` kiterjesztés azonban nem feltétlenül jó választás, ugyanis ilyen beállítás esetén minden kiküldött HTML lap áthalad a PHP-elemzőn, ezáltal jelentősen csökkenhet a kiszolgálás sebessége.

Ha a PHP előretelepítve található meg a kiszolgálón és nincs elérésünk az Apache beállításait tartalmazó fájlhoz, létrehozhatunk egy `.htaccess` nevű állományt a saját könyvtárunkban és abban is megadhatjuk a fenti sorokat. A `.htaccess` fájlok hatása kiterjed az adott könyvtárra és annak minden alkönyvtárára is. Ez a megoldás azonban csak akkor működőképes, ha az Apache `AllowOverride` beállítása az adott könyvtárra a `FileInfo` vagy az `All` értéket tartalmazza.

A `.htaccess` az alapbeállítású fájlnev, amit a könyvtár speciális beállításaihoz használhatunk, de az adott rendszeren más is lehet. Ezt a `httpd.conf` állomány `AccessFileName` beállítása határozza meg. Ez a fájl általában akkor is olvasható, ha nem rendelkezünk rendszergazdai jogokkal a kiszolgálón.

A `.htaccess` fájl tökéletes módja annak, hogy testreszabjuk a tárhelyünket, ha a kiszolgáló beállítófájljában nem módosíthatjuk a paramétereket. A PHP működését közvetlenül azonban a `php.ini` szabályozza.

php.ini

A PHP működését a fordítás vagy telepítés után is befolyásolhatjuk, a `php.ini` használatával. UNIX rendszereken az alapbeállítású könyvtár a `php.ini` fájl számára a `/usr/local/lib`, Windows rendszereken a Windows könyvtára. Emellett a feldolgozásra kerülő PHP oldal könyvtárában – a munkakönyvtárban – elhelyezett `php.ini` fájlban felülbírálnak a korábban beállított értékeket, így akár könyvtáranként különböző beállításokat adhatunk meg.

A letöltött PHP csomag könyvtárában található egy minta `php.ini` fájl, amely a „gyári beállításokat” tartalmazza. Ezek az értékek lépnek érvénybe akkor, ha a PHP nem talál `php.ini` fájlt sem.

Az alapértékek elegendőek lehetnek ahhoz, hogy a könyv példait futtassuk, ám célszerű néhány módosítást elvégezni; ezeket a huszonkettedik órában tárgyaljuk.

A `php.ini` fájl beállításai egy névből, egy egyenlőségjelből és egy értékből állnak. A szóközőket a feldolgozó figyelmen kívül hagyja.

Ha a PHP előretelepítve állt rendelkezésre a rendszerünkön, ellenőrizzük a `php.ini` fájlban található beállításokat. Ha esetleg nem lenne jogosultságunk a fájl módosítására, a PHP programjaink könyvtárába helyezett saját `php.ini` fájlal felülírhatjuk

az alapbeállítást. Másik lehetőségünk, hogy létrehozunk egy PHPRC nevű környezeti változót, amely kijelöli `php.ini` fájlunkat.

A `php.ini` beállításait bármikor módosíthatjuk, de ha feldolgozónk Apache modulként fut, a változtatások érvénybe léptetéséhez újra kell indítani a webkiszolgálót.

short_open_tag

A `short_open_tag` beállítás határozza meg, hogy használhatjuk-e a rövid `<?>` kód formát a PHP kódblokkok írására. Ha ez ki van kapcsolva, az alábbi sorok valamelyikét láthatjuk:

```
short_open_tag = Off  
short_open_tag = False  
short_open_tag = No
```

Ahhoz, hogy engedélyezzük ezt a beállítást, a következő sorok egyikét kell használnunk:

```
short_open_tag = On  
short_open_tag = True  
short_open_tag = Yes
```

A PHP blokkok kezdő- és záróelemeiről a következő órában lesz szó.

Hibajelentések beállításai

Ha hibákat keresünk programjainkban, hasznos a hibaüzenetek kiírása a HTML oldalba a böngésző számára. Ez alapbeállításban bekapcsolt:

```
display_errors = On
```

Beállíthatjuk a hibajelentési szintet is. Ez azt jelenti, hogy mivel többféle hibaüzenet-típus is rendelkezésre áll, letilthatjuk egyik vagy másik típust, amennyiben nem szeretnénk PHP oldalaink kimenetében látni az abba a csoportba tartozó hibákat. A hibakezelés beállításával alaposabban a huszonkettedik órában foglalkozunk, addig az alábbi értékadás tökéletesen megfelel:

```
error_reporting = E_ALL & ~ E_NOTICE
```

Ezzel a PHP minden hibát jelezni fog a lehetséges problémákat jelölő figyelmeztetések kivételével. Ezek a figyelmeztetések megakadályoznák néhány szokásos PHP módszer alkalmazását, ezért ez az alapbeállítás.

Változókra vonatkozó beállítások

A PHP a GET és POST kérésekből, sütikből, kiszolgálói és környezeti értékekből létrehoz bizonyos változókat. Ennek működését is a `php.ini` fájlban szabályozhatjuk.

A `track_vars` beállítás azt adja meg, hogy létrejöjjenek-e asszociatív tömbök egy HTTP lekérés eredményeként. Ez alapbeállításban engedélyezett, a PHP 4.0.2-es változat óta nem is lehet kikapcsolni:

```
track_vars = On
```

A `register_globals` beállítás azt határozza meg, hogy a HTTP lekéréskor ezek a változók globális változókként jöjjenek-e létre. A PHP fejlődésével egyre inkább azt javasolják a programozóknak, hogy mellőzzék ennek a szolgáltatásnak a használatát, mivel így rendkívül sok változó jöhet létre és ez ütközéseket okozhat, ha nem jól választjuk meg a változók neveit. Ennek ellenére ma a PHP programok legnagyobb része – többek között a könyv számos példája is – arra épít, hogy ez a beállítás be van kapcsolva:

```
register_globals = On
```

Segítség!

A segítség mindig kéznél van az Interneten, különösen a nyílt forráskódú programokkal kapcsolatos problémák esetén. Ezért mielőtt a levelezőprogramunk „Küldés” gombját megnyomnánk, gondolkozzunk el egy kicsit. Akármennyire is működésképtelennek tűnhet a telepített értelmezőnk, beállításunk vagy programozási megoldásunk, jó esélyünk van rá, hogy nem vagyunk ezzel egyedül. Valaki talán már megválaszolta kérdésünket.

Ha falba ütközünk, az első hely, ahol segítséget kereshetünk, a PHP hivatalos honlapja a <http://www.php.net/> címen, különösen az ott található, olvasói kiegészítésekkel ellátott kézikönyv: <http://www.php.net/manual/>. Sok segítséget és információt találhatunk a Zend webhelyén is: <http://www.zend.com/>. A magyar PHP fejlesztők a weblabor.hu webmester honlapon találhatnak rengeteg információt: <http://weblabor.hu/php/>. Itt készítjük a magyar PHP dokumentációt is.

Ha nem sikerült megtalálni a megoldást, hasznos segítség lehet, hogy a PHP hivatalos webhelyén keresést is végezhetünk. A tanács, ami után kutatunk, talán egy sajtóközleményben, vagy egy FAQ-fájlban rejtőzik. Egy másik kitűnő forrás a PHP Knowledge Base: <http://www.faqs.com/knowledge-base/index.phtml>. Keresés itt is végezhető.

Még mindig sikertelenek próbálkozásaink? A PHP levelezőlisták kereshető archívumaira mutató hivatkozások megtalálhatóak a <http://www.php.net/support.php> oldalon, számos más hivatkozással együtt. Ezek az archívumok óriási mennyiségű információt tartalmaznak a PHP közösség legjobbjaitól. Eltölthetünk némi időt pár kulcsszó kipróbálásával.

Amennyiben ezek után is meg vagyunk győződve arról, hogy problémánk még nem merült fel korábban, jó szolgálatot tehetünk a PHP közösségnek, ha felhívjuk rá a figyelmet.

A PHP levelezőlistákra való jelentkezéshez az archívumokat felsoroló oldalon találunk hivatkozásokat. A listák gyakran nagy forgalmúak, de ezt ellensúlyozza, hogy rendkívül sokat lehet belőlük tanulni. Ha érdeklődünk a PHP programozás iránt, legalább egy kötegelt kézbesítésű (digest) listára iratkozzunk fel. A kötegeltség azt jelenti, hogy a listára érkező leveleket nem egyenként, hanem naponta egy-két levélben összefűzve kapjuk meg. Ha sikerül megtalálni az érdeklődési körünknek megfelelő levelezőlistát a számos lehetőség közül, beküldhetjük oda a problémánkat. A PHP honlapján nemzetközi levelezőlisták oldalaira mutató hivatkozásokat is találunk. A magyar PHP levelezőlista és annak archívuma a <http://weblabor.hu/wl-phplista/> címen érhető el.

Mielőtt elküldenénk egy kérdést, gyűjtsük össze a probléma szempontjából fontos információkat, de ne írjunk túl sokat! A következő elemek gyakran szükségesek:

- A használt operációs rendszer
- A telepítés alatt álló vagy futó PHP-változat száma
- A beállításkor használt `configure` paraméterek
- Bármilyen `configure` vagy `make` kimenet, ami előjelezhetette a telepítési hibát
- Egy ésszerűen teljes részlet a kódból, ami a problémát okozza

Miért kell ilyen sok szempontot figyelembe vennünk, mielőtt egy levelezőlistára postáznánk kérdésünket? Először is a fejlesztési problémák megoldásában szerzett tapasztalat előnyös lehet a későbbi munka során. Egy tapasztalt programozó általában gyorsan és hatékonyan tud problémákat megoldani. Egy alapvető kérdés feltevése technikai jellegű listán többnyire egy-két olyan választ eredményez, amelyben felhívják figyelmünket, hogy erre a kérdésre az archívumban megtalálható a válasz.

Másodszor, egy levelezőlista nem hasonlítható össze egy kereskedelmi termék támogatási központjával. Senki sem kap fizetést, hogy megválaszolja kérdéseinket. Ennek ellenére lenyűgöző szellemi erőforráshoz nyújt elérést, beleértve a PHP néhány fejlesztőjét is. A kérdések a válasszal együtt az archívumba kerülnek, hogy később más programozók segítségére lehessenek. Ezért felesleges olyan kérdések feltevése, amelyek már többször szerepeltek a listán.

Ezek megfontolása után ne vonakodjunk kérdéseket küldeni a levelezőlistákra. A PHP fejlesztők civilizált, segítőkész emberek és a probléma felvetésével esetleg másoknak is segíthetünk egy hasonló kérdés megoldásában.

Végül, ha úgy tűnik, hogy a probléma nem a mi kódunkban, hanem a PHP-értelmezőprogramban található, küldjünk egy hibajelentést a fejlesztőknek a `http://bugs.php.net/` címen. Ha a gubanc valóban új, a hibát a PHP következő kiadásában többnyire kijavítják.

Összefoglalás

A PHP 4 nyílt forráskódú. Nyílt abban az értelemben is, hogy nem szükséges egy meghatározott kiszolgálót, operációs rendszert vagy adatbázist használnunk a fejlesztéshez. Ebben az órában láttuk, honnan szerezhető be a PHP és más – a webhelyek szolgáltatásaiban segítő – nyílt forráskódú programok. Megnéztük, hogyan fordítható le a PHP Apache modulként Linux rendszeren. Ha nem a forráskódot töltjük le a hálózatról, hanem egy lefordított változatot, akkor a csomagban részletes információkat találunk a telepítéssel kapcsolatban. Áttekintettünk néhány `configure` paramétert, melyekkel a feldolgozó képességeit befolyásolhatjuk. Tanultunk a `php.ini` fájlról és néhány beállítási lehetőségről, amit megadhatunk benne. Végül megnéztük, hol találhatunk segítséget, ha problémáink akadnak. Ezzel készen állunk arra, hogy megbirkózzunk a PHP nyelvel.

Kérdések és válaszok

A telepítést Apache webkiszolgálót használó Linux operációs rendszeren vezettük végig. Ez azt jelenti, hogy a könyv nem megfelelő más rendszer vagy kiszolgálóprogram használata esetén?

A PHP egyik nagy erőssége, hogy több rendszeren is képes futni. Ha problémák adódnak az adott rendszeren a PHP telepítésével, olvassuk el a csomagban található dokumentációt és a PHP kézikönyv megfelelő fejezetét. Általában széles körű lépésről-lépésre leírt utasításokat kapunk a telepítéshez. Ha továbbra sem sikerül megoldani a problémát, a „Segítség!” című részben ismertetett módszerek célravezetőek lehetnek.

Hogyan érhető el böngészőből a PHP állományok, ha a gépre telepítettük a webkiszolgálót?

A PHP alkalmazások fejlesztéséhez és teszteléséhez nem szükséges, hogy számítógépünk az Internetre legyen kapcsolva, bár az éles környezetben való ellenőrzés hasznosabb lehet. Bármilyen operációs rendszert is használunk, akár hálózatba kötött gépen dolgozunk, akár nem, a saját gépünk IP-címe `127.0.0.1`, neve akkor is `localhost` lesz. Ezért ha a saját gépünkön lévő webkiszolgáló gyökérkönyvtárában lévő `elso.php` fájlt szeretnénk megnyitni, a `http://localhost/elso.php` címen érhetjük el. Windows operációs rendszeren feltétlenül telepíteni kell a TCP/IP támogatást, hogy ez a lehetőség rendelkezésre álljon.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hol érhető el a PHP kézikönyv?
2. UNIX rendszeren hogyan kaphatunk bővebb információkat a beállítási lehetőségekről (milyen paramétereket kell átadni a `configure`-nak)?
3. Hogy hívják általában az Apache beállításait tartalmazó fájlt?
4. Milyen sort kell az Apache beállítófájlhoz adni, hogy a kiszolgáló felismerje a `.php` kiterjesztést?
5. Mi a PHP beállításait tartalmazó fájl neve?

Feladatok

1. Telepítsük a PHP-t. Ha sikerült, nézzük át a PHP beállítófájlt és ellenőrizzük a telepítés helyességét.



3. ÓRA

Első PHP oldalunk

A PHP telepítése és beállítása után eljött az ideje, hogy elkezdjünk vele dolgozni. Ebben az órában elkészítjük az első programunkat és elemezzük a kódot. Az óra végére képesek leszünk olyan oldalak készítésére, amelyek HTML és PHP kódot is tartalmaznak.

Ebben az órában a következőket tanuljuk meg:

- Hogyan hozunk létre, töltünk fel és futtassunk egy PHP programot?
- Hogyan vegyítsünk HTML és PHP kódot egy dokumentumon belül?
- Hogyan tehetjük a kódunkat olvashatóbbá megjegyzések használatával?

Első programunk

Vágjunk a közepébe és kezdjük egy kész PHP oldallal! A kezdéshez nyissuk meg kedvenc szövegfájl-szerkesztőnket! A HTML dokumentumokhoz hasonlóan a PHP állományok is formázás nélküliek, tisztán szövegfájlok. Ez azt jelenti, hogy bármilyen szövegfájl szerkesztésére íródott programmal készíthetünk PHP állományokat. Ilyenek például a Windows Jegyzettömbje, a Simple Text vagy a BEdit MacOS alatt, vagy a VI és az Emacs UNIX esetében. A legnépszerűbb HTML-szerkesztők általában nyújtanak valamilyen különleges támogatást PHP-szerkesztéshez is. Ilyenek például a kódszínezés, a kifejezésszerkesztő vagy a tesztelő.

Gépeljük be a 3.1. programot, és mentjük `elso.php` néven.

3.1. program Az első PHP program

```
1: <?php
2: print ("Hello Web!");
3: ?>
```

A 3.1. ábra a 3.1. program kódját mutatja a KEdit programban.

3.1. ábra

*Első PHP oldalunk
begépelve a KEdit
programban*



A kiterjesztés igen fontos, mivel ez jelzi a kiszolgáló számára, hogy a fájl PHP kódot tartalmaz és futtatásához meg kell hívni a feldolgozót. Az alapbeállítású PHP kiterjesztés a `.php`, ez azonban a kiszolgáló beállításainak módosításával megváltoztatható. Ezzel az előző órában részletesen foglalkoztunk.

Ha nem közvetlenül a kiszolgálóprogramot futtató gépen dolgozunk, feltehetően az FTP nevű szolgáltatást kell használnunk arra, hogy PHP oldalunkat a kiszolgálóra juttassuk. A Windowsban többek között a WS-FTP használható erre a célra, MacOS használata esetén a Fetch lehet hasznos.

Ha sikerült elhelyeznünk a dokumentumot a kiszolgálón, elérhetjük egy webböngésző segítségével. Ha minden rendben ment, az ablakban a program kimenetét láthatjuk. A 3.2. ábra az `első.php` kimenetét mutatja.

3.2. ábra

Siker: a 3.1. program kimenete



Ha a PHP-t nem sikerült telepíteni a kiszolgálóra vagy nem a megfelelő kiterjesztést használtuk, feltehetően nem az előző ábrán látható kimenetet kapjuk. Ebben az esetben általában a PHP program forráskódját látjuk viszont. Ezt mutatja a 3.3. ábra.

3.3. ábra

Kudarcc: a kiterjesztés nem azonosítható



Ha ez történik, először ellenőrizzük a mentett fájl kiterjesztését. A 3.3. ábrán látható lapot véletlenül `első.nphp` névvel mentettük. Ha a kiterjesztéssel nincs probléma, ellenőriznünk kell, hogy a PHP sikeresen felkerült-e a rendszerre és hogy a kiszolgálót beállítottuk-e a megfelelő kiterjesztés felismerésére. Ezekkel a kérdésekkel az előző órában foglalkoztunk részletesen.

Miután sikerült feltölteni és kipróbálni első programunkat, kielemezhetjük az imént begépelt kódot.

PHP blokkok kezdése és befejezése

Amikor PHP oldalakat írunk, tudatnunk kell a feldolgozóval, mely részeket hajtsa végre. Ha nem adjuk meg, hogy a fájl mely részei tartalmazzanak PHP kódot, az értelmező mindent HTML-nek tekint és változtatás nélkül továbbküldi a böngésző számára. A 3.1. táblázat a PHP kód blokkjainak kijelölésére szolgáló elemeket sorolja fel:

3.1. táblázat PHP blokkok kezdő- és záróelemei

<i>Elnevezés</i>	<i>Kezdőelem</i>	<i>Záróelem</i>
Hagyományos	<?php	?>
Rövid	<?	?>
ASP stílusú	<%	%>
Script elem	<SCRIPT LANGUAGE=" PHP ">	</SCRIPT>

A 3.1. táblázatban látható lehetőségek közül csak az első és az utolsó áll minden beállítás esetén rendelkezésre. A rövid és ASP stílusú elemeket engedélyezni kell a `php.ini` fájlban. Az előző órában tárgyaltuk a beállítás módszerét.

A rövid kezdőelemek felismerését a `short_open_tag` beállítással engedélyezhetjük, ha „On” állapotba tesszük:

```
short_open_tag = On
```

A rövid kezdőelemek alapbeállításban engedélyezettek, ezért nem kell szerkesztenünk a `php.ini` fájlt, hacsak valaki előzőleg ki nem kapcsolta.

Az ASP stílusú elemek engedélyezéséhez az `asp_tags` beállítást kell bekapcsolni:

```
asp_tags = On
```

Ez akkor lehet hasznos, ha olyan termékkel készítünk PHP oldalakat, amely törli a PHP blokkokat, de az ASP részeket érintetlenül hagyja.

A `php.ini` fájl módosítása után lehetőségünk van választani a bekapcsolt típusok közül. Rajtunk múlik, hogy melyik elemet választjuk, de ha XML-lel is szeretnénk dolgozni, mindenképpen a hagyományos formát kell alkalmaznunk és le kell tiltanunk a rövid stílust. Mivel a hagyományos forma minden rendszeren rendelkezésre áll, a fejlesztők programjaikban általában ezt alkalmazzák, így a könyvben is ezzel fogunk találkozni.

Nézzük meg, hogyan fest egy PHP fájl, amely a fent említett formákat használja az előző program kibővítéseként. Ha engedélyeztük, bármely négy kezdő- és záróelemet használhatjuk a programban:

```
<?
print ("Hello Web!");
?>
```

```
<?php
print ("Hello Web!");
?>
```

```
<%
print ("Hello Web!");
%>
```

```
<SCRIPT LANGUAGE="PHP">
print ("Hello Web!");
</SCRIPT>
```

Ha PHP-ben egysoros kódot írunk, nem kell külön sorba tennünk a kezdő- és záróelemeket, illetve a programsort:

```
<?php print("Hello Web!"); ?>
```

Miután megtanultuk, hogyan határozhatunk meg egy PHP kódblokkot, nézzük meg még közelebbről a 3.1. programot.

A print() függvény

A `print()` függvény kiírja a kapott adatokat. A legtöbb esetben minden, ami a `print()` függvény kimenetén megjelenik, a böngészőhöz kerül. A függvények olyan parancsok, amelyek valamilyen műveletet végeznek, többnyire attól függően, hogy milyen adatokat kapnak. A függvényeknek átadott adatokat majd nem mindig zárójelbe kell tennünk a függvény neve után. Ebben az esetben a `print()` függvénynek egy karakterláncot adtunk át. A karakterláncokat mindig (egyes vagy kettős) idézőjelbe kell tenni.



A függvények általában megkövetelik, hogy zárójeleket helyezünk a nevük után, akár küldünk adatokat számukra, akár nem. A `print()` ebből a szempontból kivételes, mivel hívása esetén a zárójelek elhagyhatók. Ez a `print()` függvény megszokott formája, ezért a továbbiakban ezt alkalmazzuk.

A 3.1. program egyetlen sorát pontosvesszővel fejeztük be. A pontosvessző tudatja a feldolgozóval, hogy az utasítás véget ért.

ÚJDONSÁG

Az utasítás a feldolgozónak adott parancs. Általánosságban a PHP számára azt jelenti, mint az írott vagy beszélt nyelvben a mondat. Az utasításokat a legtöbb esetben pontosvesszővel kell lezárni, a mondatokat írásjellel. Ez alól kivételt képeznek azok az utasítások, amelyeket más utasítások vesznek körül, illetve azok, amelyek blokk végén állnak. A legtöbb esetben azonban az utasítás végéről lefelejtett pontosvessző megzavarja az elemzőt és hibát okoz.

Mivel a 3.1. programban az egyetlen megadott utasítás a blokk végén áll, a pontosvessző elhagyható.

HTML és PHP kód egy oldalon

3.2. program PHP program HTML tartalommal

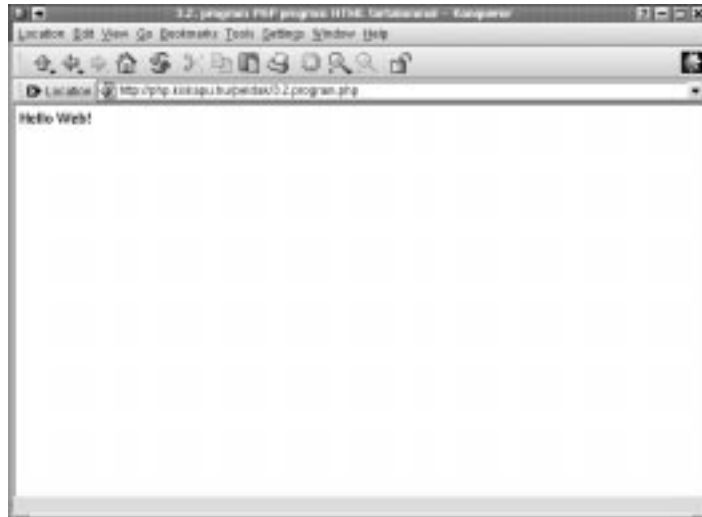
```
1: <html>
2: <head>
3: <title>3.2. program PHP program HTML
   tartalommal</title>
4: </head>
5: <body>
6: <b>
7: <?php
8: print ("Hello Web!");
9: ?>
10: </b>
11: </body>
12: </html>
```

Látható, hogy HTML kód beépítése a PHP oldalakba egyszerűen a HTML tartalom begépeléséből áll. A PHP feldolgozó figyelmen kívül hagy mindent a PHP nyitó- és záróelemeken kívül. Ha a 3.2. programot megnézzük böngészővel, a „Hello Web!” szöveget látjuk vastagon, mint ahogy ez a 3.4. ábrán megfigyelhető. Ha a dokumentum forrását is megnézzük, ahogy a 3.5. ábra mutatja, a kód hagyományos HTML dokumentumnak tűnik.

Egy dokumentumban a HTML elemek közé tetszőleges számú PHP kódblokk írható. Bár több kódblokkot helyezhetünk el egy dokumentumon belül, ezek együttesen alkotnak egy programot. Bármí, amit egy megelőző blokkban határoztunk meg, (változók, függvények vagy osztályok) a dokumentumon belül elérhető lesz a későbbi blokkokban is. A több együttműködő, összességében egy nagyobb programot megvalósító PHP fájlt nevezzük PHP alkalmazásnak.

3.4. ábra

Ha a 3.2 programot megnézzük böngészővel, a „Hello Web!” szöveget látjuk vastagon



3.5. ábra

Ha a dokumentum forrását nézzük, a kód hagyományos HTML dokumentumnak tűnik



Megjegyzések a PHP kódokban

Az íráskor átlátható programkód hat hónappal később reménytelenül kuszának tűnhet. Ha azonban megjegyzéseket fűzünk a kódhoz, miközben írjuk, a későbbiekben sok időt takaríthatunk meg magunknak és más programozóknak, akik az adott kóddal fognak dolgozni.

ÚJDONSÁG

A megjegyzések a kódban található olyan szövegrészek, amelyeket a feldolgozó figyelmen kívül hagy. Segítségükkel olvashatóbbá tehetjük a programot és jegyzeteket fűzhetünk hozzá.

A PHP egysoros megjegyzései két perjellel (//) vagy egy kettőskereszt karakterrel (#) kezdődhetnek. Az ezen jelzések után található szöveget a PHP-értelmező mindig figyelmen kívül hagyja, a sor vagy a PHP blokk végéig.

```
// Ez megjegyzés  
# Ez is megjegyzés
```

A többsoros megjegyzések egy perjelet követő csillag karakterrel kezdődnek (/*) és egy csillagot követő perjellel (*/) fejeződnek be.

```
/*  
Ez egy megjegyzés.  
A PHP-értelmező  
ezen sorok egyikét  
sem fogja feldolgozni.  
*/
```

Összefoglalás

Mostanra rendelkezésünkre állnak azok az eszközök, melyekkel képesek vagyunk egy egyszerű PHP program futtatására egy megfelelően beállított kiszolgálón.

Ebben az órában elkészítettük első PHP programunkat. Láttuk, hogyan használható egy szöveges szerkesztő, hogy létrehozzunk és mentünk egy PHP dokumentumot. Elemeztük a négy rendelkezésre álló blokk kezdő- és záróelemet. Megtanultuk, miként kell használnunk a `print()` függvényt, hogy a böngésző számára kimenetet küldjünk, majd összeállítottunk egy HTML és PHP kódot is tartalmazó állományt. Végül tanultunk a megjegyzésekről és arról, hogyan építhetjük be ezeket a PHP dokumentumokba.

Kérdések és válaszok

Melyik a legjobb kezdő- és záróelem?

Ez nagyrészt saját döntésünkön múlik. A hordozhatóság szem előtt tartásával a hagyományos `<?php ?>` megoldás a legjobb döntés. A rövid kezdőelemek alapesetben engedélyezettek és rendelkeznek a tömörség előnyös tulajdonságával.

Milyen szerkesztőprogramokat kell elkerülni a PHP kódok készítésekor?

Ne használjunk olyan szövegszerkesztőket, amelyek saját formátummal rendelkeznek, mint a Microsoft Word. Ha ilyen típusú szerkesztővel mentjük szöveges fájlként az elkészült dokumentumot, a kódban megbújó rejtett karakterek biztos gondot fognak okozni.

Mikor kell megjegyzéseket fűzni a kódokhoz?

Ez is nagyrészt a programozó döntésén múlik. A rövid programokhoz nem érdemes magyarázatot fűzni, mivel ezek még hosszú idő után is érthetőek maradnak. Ha a program azonban akár csak egy kicsit is bonyolult, már javasolt megjegyzésekkel bővíteni. Ez hosszú távon időt takarít meg számunkra és kíméli idegeinket.

3

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. A felhasználó böngészőjével elolvashatja-e a PHP kódot?
2. Adjuk meg a PHP hagyományos kezdő- és záróelemeit!
3. Adjuk meg az ASP stílusú kezdő- és záróelemeket!
4. Adjuk meg a Script kezdő- és záróelemeket!
5. Mely függvény használatos a böngésző számára küldött adatok kiírásához?

Feladatok

1. Gyakoroljuk a PHP oldalak elkészítését, feltöltését és futtatását!



II. RÉSZ

A PHP nyelv

- 4. óra Az alkotóelemek
- 5. óra Vezérlési szerkezetek
- 6. óra Függvények
- 7. óra Tömbök
- 8. óra Objektumok



4. ÓRA

Az alkotóelemek

Ebben az órában már mélyebben elmerülünk a nyelv rejtelseiben. Az alapokkal kezdünk, így a kezdő programozónak rengeteg új információt kell majd feldolgoznia. Aggodalomra azonban semmi ok, bármikor vissza lehet lapozni a könyvben. A lényeg, hogy a fejezetben leírtak megértésére törekedjünk, ne csupán a memorizálásra.

A gyakorlott programozóknak is ajánlatos legalábbis átlapozniuk ezen óra anyagát. Az órában a következőket tanuljuk meg:

- Mik a változók és hogyan használjuk azokat?
- Hogyan hozhatunk létre változókat és hogyan érhetjük el azokat?
- Mik azok az adattípusok?
- Melyek a leggyakrabban használt műveletek?
- Hogyan hozhatunk létre kifejezéseket műveletjelek használatával?
- Hogyan határozhatunk meg állandókat és hogyan használhatjuk azokat?

Változók

A változók különleges tárolók, amiket abból a célból hozunk létre, hogy értéket helyezzünk el bennük. A változók egy dollárjelből (\$) és egy „tetszőlegesen” választott névből tevődnek össze. A név betűket, számokat és aláhúzás karaktereket (_) tartalmazhat (számmal azonban nem kezdődhet!), szóközőket és más olyan karaktereket, amelyek nem számok vagy betűk, nem. Íme néhány érvényes változónév:

```
$a;  
$egy_hosszabb_valtozonev;  
$elalszom_zzzzzzzzzzzzzzz;
```

Ne feledjük, hogy a pontosvessző (;) a PHP utasítás végét jelzi, így az előzőekben a pontosvessző nem része a változók nevének.

ÚJDONSÁG

A változók adatokat – számokat, karakterláncokat, objektumokat, tömböket vagy logikai értékeket – tárolnak, tartalmuk bármikor módosítható.

Amint látjuk, rengeteg féle nevet adhatunk változóinknak, bár a csak számokból álló változónevek nem szokványosak. Változó létrehozásához (deklarálásához, vagyis bevezetéséhez) egyszerűen csak bele kell írni azt a programunkba. Létrehozáskor általában rögtön értéket is szoktunk adni a változónak.

```
$szam1 = 8;  
$szam2 = 23;
```

Itt két változót hoztunk létre és a hozzárendelő műveletjellel (=) értéket is adtunk azoknak. Az értékadásról bővebben a Műveletjelek és kifejezések című részben tanulunk, az óra későbbi részében. Miután értéket adtunk változóinknak, úgy kezelhetjük azokat, mintha maguk lennének az értékek. Vagyis a fenti példánál maradva a létrehozás után írt

```
print $szam1;
```

hatása megegyezik a

```
print 8;
```

hatásával, feltéve, hogy a \$szam1 változó értéke 8 maradt.

Dinamikus változók

Változót tehát úgy hozunk létre, hogy egy dollárjel után írjuk a változó nevét. Szokatlan, ám hasznos, hogy a változó nevét is tárolhatjuk változóban. Tehát ha az alábbi módon értéket rendelünk egy változóhoz

```
$felhasznalo = "Anna";
```

az megegyezik azzal, mintha ezt írnánk:

```
$tarolo = "felhasznalo";  
$$tarolo = "Anna";
```

A `$tarolo` változó a `"felhasznalo"` szöveget tartalmazza, ezért a `$$tarolo`-t úgy tekinthetjük, mint egy dollárjelet, melyet egy változó neve követ (`$tarolo`), a `$tarolo` viszont ismét csak egy változó, amit a PHP a változó értékével helyettesít, vagyis `"felhasznalo"`-val. Tehát a fenti kifejezés a `$felhasznalo`-val egyenértékű.



Dinamikus változókat karakterlánc-konstanssal (állandóként meghatározott karakterláncsal) is létrehozhatunk. Ekkor a változó nevéről szolgáló karakterláncot kapcsos zárójelbe tesszük:

```
${"felhasznalonev"} = "Anna";
```

Ez első ránézésre nem tűnik túl hasznosnak. Ha azonban a karakterlánc-összeűző műveletjellel ciklusban használjuk (lásd a Vezérlési szerkezetek című ötödik órát), e módszerrel dinamikus változók tucatjait hozhatjuk létre.

A dinamikus változók elérése ugyanúgy történik, mint a „hagyományos” változóké, tehát a

```
$felhasznalo = "Anna";  
print $felhasznalo;
```

azonos a

```
$felhasznalo = "Anna";  
$tarolo = "felhasznalo";  
print $$tarolo;
```

kóddal, eltekintve természetesen attól, hogy itt létrehoztunk egy `$tarolo` nevű változót, melynek értéke `"felhasznalo"`.

Ha egy dinamikus változót egy karakterláncon belül szeretnénk kiírni, az értelmezőnek némi segítséget kell adnunk. Az alábbi print utasítás hatására

```
$felhasznalo = "Anna";  
$starolo = "felhasznalo";  
print "$$starolo";
```

a böngésző nem az "Anna" szöveget jeleníti meg, mint ahogy várnánk, hanem kiírja a dollárjelet, majd a "felhasznalo" szöveget, vagyis összességében azt, hogy "\$felhasznalo".

ÚJDONSÁG

Ha egy változót kettős idézőjelek közé teszünk, a PHP szolgálatkészben beilleszti annak értékét.

A mi esetünkben a PHP a \$starolo karaktersorozatot a "felhasznalo" szövegre cserélte, az első dollárjelet pedig a helyén hagyta. Annak érdekében, hogy egyértelművé tegyük a PHP számára, hogy a karakterláncon belüli változó egy dinamikus változó része, kapcsos zárójelbe kell tennünk. Az alábbi kódrészlet print utasítása

```
$felhasznalo = "Anna";  
$starolo = "felhasznalo";  
print "${starolo}";
```

már az "Anna" szöveget írja ki, ami a \$felhasznalo nevű változó értéke.

A 4.1. példaprogram egy PHP oldalon belül tartalmazza a korábbi programrészteket és változóban tárolt karakterláncokat használ a \$felhasznalo kezdeti értékének beállítására és elérésére.

4.1. program Dinamikusán beállított és elért változók

```
1: <html>  
2: <head>  
3: <title>4.1. program Dinamikusán beállított és elért  
   változók</title>  
4: </head>  
5: <body>  
6: <?php  
7: $starolo = "felhasznalo";  
8: $$starolo = "Anna";  
9: // lehetne egyszerűen csak:  
10: // $felhasznalo = "Anna";  
11: // vagy
```

4.1. program (folytatás)

```
12: // ${"felhasznalo"} = "Anna"
13: // persze ekkor nem kellene dinamikus változók
14: print "$felhasznalo<br>";
15: print $$tarolo;
16: print "<br>";
17: print "${$tarolo}<br>";
18: print "${'felhasznalo'}<br>";
19: ?>
20: </body>
21: </html>
```

Hivatkozások a változókra

A PHP alapértelmezés szerint értékadáskor a változók értékeit használja. Ez azt jelenti, hogy ha az \$egyikValtozo-t hozzárendeljük egy \$masikValtozo-hoz, akkor a \$masikValtozo-ba az \$egyikValtozo értékének másolata kerül. Az \$egyikValtozo tartalmának későbbi módosítása nincs semmiféle hatással a \$masikValtozo-ra. A 4.2. példaprogram ezt mutatja be.

4

4.2. program Változók érték szerinti hozzárendelése

```
1: <html>
2: <head>
3: <title>4.2. program Változók érték szerinti
   hozzárendelése</title>
4: </head>
5: <body>
6: <?php
7: $egyikValtozo = 42;
8: $masikValtozo = $egyikValtozo;
9: // $masikValtozo-ba $egyikValtozo tartalmának
   másolata kerül
10: $egyikValtozo = 325;
11: print $masikValtozo; // kiírja, hogy 42
12: ?>
13: </body>
14: </html>
```

Itt a 42 értéket adjuk az \$egyikValtozo-nak, majd a \$masikValtozo-hoz hozzárendeljük az \$egyikValtozo-t. Ekkor a \$masikValtozo-ba az \$egyikValtozo értékének másolata kerül. A print utasítás igazolja ezt, mivel a böngészőben a 42 jelenik meg.

A PHP 4-es kiadásában ezt a viselkedésmódot megváltoztathatjuk. Kikényszeríthetjük, hogy a \$masikValtozo-ba ne az \$egyikValtozo értéke kerüljön, hanem egy hivatkozás, amely az \$egyikValtozo-ra mutat. Ezt illusztrálja a 4.3. példa-program.

4.3. program Változóra mutató hivatkozás

```
1: <html>
2: <head>
3: <title>4.3. program Változóra mutató
   hivatkozás</title>
4: </head>
5: <body>
6: <?php
7: $egyikValtozo = 42;
8: $masikValtozo = &$egyikValtozo;
9: // $masikValtozo-ba $egyikValtozo-ra mutató
   hivatkozás kerül
10: $egyikValtozo = 325;
11: print $masikValtozo; // kiírja, hogy 325
12: ?>
13: </body>
14: </html>
```

A 4.2. példaprogramhoz képest a változás mindössze egy karakter.

Az \$egyikValtozo elé tett & jel gondoskodik róla, hogy az érték másolata helyett a \$masikValtozo-ba a változóra mutató hivatkozás kerül. Ezután a \$masikValtozo elérésekor az \$egyikValtozo-ra vonatkozó műveletek eredményét láthatjuk. Más szavakkal: mind az \$egyikValtozo, mind a \$masikValtozo ugyanazt a „tárolódobozt” használja, így értékeik mindig egyenlők. Mivel ez az eljárás kiküszöböli az egyik változóból a másikba történő értékmásolás szükségességét, csekély mértékben növelheti a teljesítményt. Hacsak nem használunk nagyon gyakran értékadásokat, ez a teljesítménybeli növekedés alig érezhető.



A hivatkozások a PHP 4-es változatában kerültek a nyelvbe.

Adattípusok

A különféle típusú adatok több-kevesebb helyet foglalnak a memóriában, a nyelv pedig mindegyiket némileg más módon kezeli. Ezért néhány programozási nyelv megköveteli, hogy a programozó előre meghatározza a változótípusát. A PHP 4 gyengén típusos, ami azt jelenti, hogy az adattípusokat úgy kezeli, mintha a típus az adathoz rendelt kiegészítő információ lenne. A PHP vegyes megközelítést használ. Egyfelől ez azt jelenti, hogy a változók rugalmasan használhatók: egyszer karakterlánc, másszor esetleg szám lehet bennük. Másfelől, nagyobb méretű programokban zavar forrása lehet, ha egy adott típusú változót várunk egy változóban, miközben valami teljesen más van benne.

A 4.1. táblázat a PHP 4-ben elérhető hat fontosabb adattípust tartalmazza, rövid leírással és példával.

4.1. táblázat Adattípusok

<i>Típus</i>	<i>Példa</i>	<i>Leírás</i>
Integer	5	Egész szám
Double	3.234	Lebegőpontos szám
String	"hello"	Karakterek sorozata, karakterlánc
Boolean	true	Logikai változó. Értéke igaz vagy hamis (true vagy false) lehet
Object		Objektum, lásd a nyolcadik, objektumokkal foglalkozó órát
Array		Tömb, lásd a hetedik, tömbökkel foglalkozó órát

Az adattípusok közül a tömböket és az objektumokat későbbre hagyjuk.

A változó típusának meghatározására a PHP 4 beépített `gettype()` függvényét használhatjuk. Ha a függvényhívás zárójelei közé változót teszünk, a `gettype()` egy karakterláncsal tér vissza, amely a változó típusát adja meg. A 4.4. példaprogram egy változóhoz négy különböző típusú értéket rendel, majd meghívja a `gettype()` függvényt.



A függvényekről bővebben a hatodik órában, a „Függvények” című részben tanulunk.

4.4. program Változó típusának vizsgálata

```
1: <html>
2: <head>
3: <title>4.4. program Változó típusának
   vizsgálata</title>
4: </head>
5: <body>
6: <?php
7: $proba = 5;
8: print gettype( $proba ); // integer
9: print "<br>"; // új sor, hogy ne follyanak össze
   a típusnevek
10: $proba = "öt";
11: print gettype( $proba ); // string
12: print "<br>";
13: $proba = 5.0;
14: print gettype( $proba ); // double
15: print "<br>";
16: $proba = true;
17: print gettype( $proba ); // boolean
18: print "<br>";
19: ?>
20: </body>
21: </html>
```

Az előbbi program a következő kimenetet eredményezi:

```
integer
string
double
boolean
```

Az integer egész szám, vagyis olyan szám, amelyben nincs tizedesjegy („tizedes-pont”).



Könyvünk ugyan magyar nyelvű, a PHP 4 azonban az angol kifejezéseket használja. A gettype ezért adja meg az integer, string, double és boolean szavakat. Ugyanígy az angol írásmód szerint a szám egészrészét a törtrésztől nem tizedesvessző, hanem tizedespont választja el. Annak érdekében, hogy ne kövessünk elgépelési hibákat, a könyv hátralevő részében a magyarul kicsit szokatlanul csengő tizedespont kifejezést használjuk.

A string karakterek sorozata. Ha programunkban karakterláncokkal dolgozunk, mindig aposztrófok (') vagy macskakörmök (") közé kell azokat tennünk. A double lebegőpontos szám, vagyis olyan szám, amely tartalmazhat tizedespontot. A boolean a két logikai érték, a true (igaz) és a false (hamis) egyikét veheti fel.



A PHP-ben a 4-es változat előtt nem létezett a boolean típus. Ott is használhattuk a true értéket, de azt az értelmező egyszerűen integer típusú 1-re fordította.

Típus módosítása a settype() segítségével

A PHP a változó típusának módosítására a settype() függvényt biztosítja. A settype()-ot úgy kell használnunk, hogy a megváltoztatandó típusú változót, valamint a változó új típusát a zárójelek közé kell tennünk, vesszővel elválasztva. A 4.5. példaprogramban a 3.14-et (lebegőpontos szám, vagyis double) olyan típusokká alakítjuk, mely típusokat ebben a fejezetben részletesen tárgyalunk.

4.5. program Változó típusának módosítása a settype() függvény segítségével

```
1: <html>
2: <head>
3: <title>4.5. program Változó típusának módosítása
   a settype() függvény segítségével</title>
4: </head>
5: <body>
6: <?php
7: $ki_tudja_milyen_tipusu = 3.14;
8: print gettype( $ki_tudja_milyen_tipusu ); // double
9: print " - $ki_tudja_milyen_tipusu<br>"; // 3.14
10: settype( $ki_tudja_milyen_tipusu, "string" );
11: print gettype( $ki_tudja_milyen_tipusu ); // string
12: print " - $ki_tudja_milyen_tipusu<br>"; // 3.14
13: settype( $ki_tudja_milyen_tipusu, "integer" );
14: print gettype( $ki_tudja_milyen_tipusu ); // integer
15: print " - $ki_tudja_milyen_tipusu<br>"; // 3
16: settype( $ki_tudja_milyen_tipusu, "double" );
17: print gettype( $ki_tudja_milyen_tipusu ); // double
18: print " - $ki_tudja_milyen_tipusu<br>"; // 3.0
19: settype( $ki_tudja_milyen_tipusu, "boolean" );
20: print gettype( $ki_tudja_milyen_tipusu ); // boolean
21: print " - $ki_tudja_milyen_tipusu<br>"; // 1
```

4.5. program (folytatás)

```
22: ?>
23: </body>
24: </html>
```

A típusmódosítás után minden esetben a `gettype()` függvényt használtuk, hogy meggyőződjünk arról, hogy a típus módosítása sikerült, majd kiírjuk a `$ki_tudja_milyen_tipusu` nevű változó értékét a böngészőbe. Amikor a "3.14" karakterláncot egészzé alakítjuk, a tizedespont utáni információ elvész. Ezért történhet meg, hogy a `$ki_tudja_milyen_tipusu` változónak még akkor is 3 az értéke, amikor újból lebegőpontos számmá alakítjuk. A végén a `$ki_tudja_milyen_tipusu` változót logikai típusúvá alakítottuk. Az ilyen átalakításoknál a 0-tól különböző számok értéke – akárcsak a nem nulla hosszúságú karakterláncoké – mindig `true` lesz. Amikor a PHP-ben kiírunk egy logikai változót, akkor ha a változó értéke `true`, a kimeneten 1-et látunk, míg a `false` értékű változók semmilyen kimenetet nem eredményeznek. Így már érthető, hogy az utolsó kiíratás miért eredményezett 1-et.

Típus módosítása típusátalakítással

A változó neve elé zárójelbe írt adattípus segítségével a változó értékének általunk meghatározott típusúvá alakított másolatát kapjuk. A lényegi különbség a `settype()` függvény és a típusátalakítás között, hogy az átalakítás során az eredeti változó típusa és értéke változatlan marad, míg a `settype()` alkalmazása során az eredeti változó típusa és értéke az új adattípus értelmezési tartományához idomul. A 4.6. program ezt hivatott illusztrálni.

4.6. program Változó típusának módosítása típusátalakítás segítségével

```
1: <html>
2: <head>
3: <title>4.6. program Változó típusának módosítása
   típusátalakítás segítségével</title>
4: </head>
5: <body>
6: <?php
7: $ki_tudja_milyen_tipusu = 3.14;
8: $starolo = ( double ) $ki_tudja_milyen_tipusu;
9: print gettype( $starolo ) ; // double
10: print " - $starolo<br>";    // 3.14
```

4.6. program (folytatás)

```
11: $starolo = ( string ) $ki_tudja_milyen_tipusu;  
12: print gettype( $starolo ); // string  
13: print " - $starolo<br>"; // 3.14  
14: $starolo = ( integer ) $ki_tudja_milyen_tipusu;  
15: print gettype( $starolo ); // integer  
16: print " - $starolo<br>"; // 3  
17: $starolo = ( double ) $ki_tudja_milyen_tipusu;  
18: print gettype( $starolo ); // double  
19: print " - $starolo<br>"; // 3.14  
20: $starolo = ( boolean ) $ki_tudja_milyen_tipusu;  
21: print gettype( $starolo ); // boolean  
22: print " - $starolo<br>"; // 1  
23: ?>  
24: </body>  
25: </html>
```

A `$ki_tudja_milyen_tipusu` változó típusát a program egyik pontján sem változtattuk meg, az végig lebegőpontos szám marad. Csupán másolatokat hozunk létre, amelyek az általunk meghatározott típusúvá alakulnak és az új érték kerül aztán a `$starolo` nevű változóba. Mivel a `$ki_tudja_milyen_tipusu` másolatával dolgozunk, a 4.5. példaprogrammal ellentétben az eredeti változóban semmilyen információt nem veszítünk el.

4

Műveletjelek és kifejezések

Most már képesek vagyunk adatokat helyezni változóinkba, meghatározhatjuk, sőt meg is változtathatjuk egy változó típusát. Egy programozási nyelv azonban nem túl hasznos, ha a tárolt adatokkal nem végezhetünk műveleteket. A műveletjelek (operátorok) olyan jelek, amelyek azon műveleteket jelzik, melyek lehetővé teszik, hogy egy vagy több értékből egy új értéket hozzunk létre. Az értéket, amellyel a műveletet végezzük, operandusnak hívjuk.

ÚJDONSÁG

Az *operátor* (műveletjel) jel vagy jelsorozat, amelyet ha értékek összekapcsolására használunk, valamilyen műveletet végezhetünk, amely általában új értéket eredményez.

ÚJDONSÁG

Az operandus egy érték, amelyet a műveletjellel kapcsolatban használunk. Egy műveletjelhez általában két operandus tartozik.

Kapcsoljunk össze két operandust és hozzunk létre egy új értéket:

`4 + 5`

Itt a 4 és az 5 az operandusok. Az összeadó operátor (+) végez rajtuk műveletet és ez szolgáltatja a 9 értéket. A műveletjelek többsége két operandus között helyezkedik el, bár az óra későbbi részében látni fogunk néhány kivételt is.

Az operandusok és műveletjelek együttesét kifejezésnek hívjuk. Annak ellenére, hogy még a legegyszerűbb kifejezéseket is műveletjelek segítségével hozzuk létre, a kifejezésnek nem kell szükségszerűen operátort tartalmaznia. Valójában a PHP mindent, ami értéket határoz meg, kifejezésnek tekint. Így az állandók, például az 543; változók, mint a `$felhasznalo` és a függvényhívások, például a `gettype()` is kifejezések, a `4+5` kifejezés pedig két további kifejezésre (4 és 5) és egy operátorra bontható.

ÚJDONSÁG

A kifejezés a függvények, értékek és műveletjelek olyan együttese, amely értéket határoz meg. Általánosságban azt mondhatjuk, hogy ha értéként használhatunk valamit, akkor az kifejezés.

Most, hogy az alapelveket tisztáztuk, ideje megismerkednünk a PHP 4 alapvető műveleteivel.

Hozzárendelés

Már találkoztunk a hozzárendelő műveletjellel, valahányszor értéket adtunk változóinknak. A hozzárendelő operátor egyetlen karakterből áll, az egyenlőségjelből (=). Jelentése: a műveletjel jobb oldalán álló kifejezés értékét hozzárendeljük a bal oldali operandushoz.

```
$nev = "Kőműves Kelemen";
```

A `$nev` változó ekkor a "Kőműves Kelemen" szöveget fogja tartalmazni. Érdekes módon, ez az értékadás maga is egy kifejezés. Első látásra úgy tűnik, hogy a hozzárendelés csak a `$nev` értékét változtatja meg, de valójában a kifejezés, amely a hozzárendelő műveletjelből áll, mindig a jobb oldalon álló kifejezés értékének másolatát adja vissza. Így a

```
print ( $nev = "Kőműves Kelemen" );
```

utasítás kiírja a böngészőbe, hogy "Kőműves Kelemen", azon felül, hogy a "Kőműves Kelemen" szöveget a `$nev` változóhoz rendeli.

Aritmetikai műveletek

Az aritmetikai műveletek pontosan azt teszik, amit elvárunk tőlük. A 4.2. táblázat a megfelelő műveletjeleket sorolja fel. Az összeadó művelettel hozzáadja a jobb oldali operandust a bal oldalához, a kivonó művelettel a jobb oldali operandust kivonja a bal oldaliból, az osztó művelettel a bal oldali operandust elosztja a jobb oldalival, a szorzó művelettel pedig összeszorozza a bal és a jobb oldali operandusokat. A maradékképző (modulus) művelettel a bal és a jobb operandus egész osztásának maradékát adja.

4.2. táblázat Aritmetikai műveletek

Műveletjel	Név	Példa	Érték
+	Összeadás	10+3	13
-	Kivonás	10-3	7
/	Osztás	10/3	3.333333333333
*	Szorzás	10*3	30
%	Maradék	10%3	1

4

Összefűzés

Az összefűzés jele a pont (.). Mindkét operandust karakterláncnak tekintve, a jobb oldali elemet hozzáfűzi a bal oldalához. Vagyis a

```
"Para" . " Zita"
```

kifejezés értéke:

```
"Para Zita"
```

Az elemek típusuktól függetlenül karakterláncként értékelődnek ki és az eredmény is mindig karakterlánc lesz.

További hozzárendelő műveletek

Annak ellenére, hogy valójában csak egy hozzárendelő művelet van, a PHP 4 számos további műveletjelet biztosít, amelyek a bal oldali operandust módosítják. A műveletek az operandusokat általában nem változtatják meg, ez alól azonban a hozzárendelés kivétel.

Az összetett hozzárendelő műveletjelek egy hagyományos műveletjelből és az azt követő egyenlőségjelből állnak. Az összetett műveletjelek megkímélnék minket attól, hogy két operátort kelljen használnunk és az elgépelés esélyét is csökkentik. A

```
$x = 4;
$x += 4; // $x most 8
```

például egyenértékű a következővel:

```
$x = 4;
$x = $x + 4; // $x most 8
```

Hozzárendelő műveletjelet minden aritmetikai és összefűző jelhez kapcsolhatunk. A 4.3. táblázat a leggyakoribb párosításokat tartalmazza.

4.3. táblázat Néhány összetett hozzárendelő műveletjel

<i>Műveletjel</i>	<i>Példa</i>	<i>Egyenértékű kifejezés</i>
+=	<code>\$x += 5</code>	<code>\$x = \$x + 5</code>
-=	<code>\$x -= 5</code>	<code>\$x = \$x - 5</code>
*=	<code>\$x *= 5</code>	<code>\$x = \$x * 5</code>
/=	<code>\$x /= 5</code>	<code>\$x = \$x / 5</code>
%=	<code>\$x %= 5</code>	<code>\$x = \$x % 5</code>
.=	<code>\$x .= "próba"</code>	<code>\$x = \$x . "próba"</code>

A 4.3. táblázat minden példájában a `$x` változó értéke változik meg, a jobb oldali operandusnak megfelelően.

Összehasonlítás

Az összehasonlító műveletek az operandusokon vizsgálatokat végeznek. Logikai értékkel térnek vissza, vagyis értékük `true` lesz, ha a feltételezett viszony fennáll, és `false`, ha nem. Ez a típusú kifejezés az olyan vezérlési szerkezetekben hasznos, mint az `if` és `while` utasítások. Ezekkel az ötödik órában találkozunk majd.

Ha meg szeretnénk vizsgálni, hogy az `$x`-ben tárolt érték kisebb-e 5-nél, a kisebb, mint jelet használhatjuk:

```
$x < 5
```

Ha `$x` értéke 3, a kifejezés értéke `true`, ha `$x` értéke 7, a kifejezés értéke `false` lesz.

Az összehasonlító műveletjeleket a 4.4. táblázatban találhatjuk. `$x` értéke 4.

4.4. táblázat Összehasonlító műveletek

<i>Műveletjel</i>	<i>Név</i>	<i>Igaz, ha</i>	<i>Példa</i>	<i>Eredmény</i>
<code>==</code>	egyenlő	a két érték megegyezik	<code>\$x == 5</code>	<code>false</code>
<code>!=</code>	nem egyenlő	a két érték különböző	<code>\$x != 5</code>	<code>true</code>
<code>===</code>	azonos	a két érték és típus megegyezik	<code>\$x === 5</code>	<code>false</code>
<code>!==</code>	nem azonos	a két érték vagy típus különböző	<code>\$x !== 5</code>	<code>true</code>
<code>></code>	nagyobb, mint	a bal oldal nagyobb a jobb oldalnál	<code>\$x > 4</code>	<code>false</code>
<code>>=</code>	nagyobb, vagy egyenlő	a bal oldal nagyobb a jobb oldalnál, vagy egyenlő	<code>\$x >= 4</code>	<code>true</code>
<code><</code>	kisebb, mint	a bal oldal kisebb a jobb oldalnál	<code>\$x < 4</code>	<code>false</code>
<code><=</code>	kisebb, vagy egyenlő	a bal oldal kisebb a jobb oldalnál, vagy egyenlő	<code>\$x <= 4</code>	<code>true</code>

Ezeket a műveletjeleket többnyire egészekkel vagy lebegőpontos számokkal használjuk, bár az egyenlőség karakterláncok esetében is vizsgálható.

Bonyolultabb összehasonlító kifejezések létrehozása logikai műveletek segítségével

A logikai műveletjelek logikai értékeken végeznek műveleteket. A vagy operátor például `true`-val tér vissza, ha bal vagy jobb operandusa `true`.

```
true || false
```

A fenti eredménye `true`.

Az és operátor csak akkor ad `true`-t, ha mindkét operandusa `true`.

```
true && false
```

A fenti értéke `false`. Valószínűtlen azonban, hogy a gyakorlatban logikai állandókon akarunk műveleteket végezni. Sokkal több értelme van, hogy két vagy több kifejezést vizsgáljunk meg. Például:

```
( $x > 2 ) && ( $x < 15 )
```

Az eredmény itt `true`, ha az `$x`-ben tárolt érték nagyobb, mint 2, de kisebb, mint 15. A zárójeleket azért tettük be, hogy a programkód átláthatóbb legyen. A 4.5. táblázat a logikai műveleteket sorolja fel.

4.5. táblázat Logikai műveletek

Műveletjel	Név	Igaz, ha	Példa	Eredmény
<code> </code>	vagy	a bal vagy a jobb operandus igaz	<code>true false</code>	<code>true</code>
<code>or</code>	vagy	a bal vagy a jobb operandus igaz	<code>true or false</code>	<code>true</code>
<code>xor</code>	kizáró vagy	vagy a bal, vagy a jobb operandus igaz, de csak az egyikük	<code>true xor true</code>	<code>false</code>
<code>&&</code>	és	a bal és a jobb operandus is igaz	<code>true && false</code>	<code>false</code>
<code>and</code>	és	a bal és a jobb operandus is igaz	<code>true and false</code>	<code>false</code>
<code>!</code>	tagadás	az egyetlen operandus hamis	<code>!true</code>	<code>false</code>

Miért kell kétféle vagy és és műveletjel? A magyarázat a műveletek kiértékelési sorrendjében rejlik, amelyről a fejezet későbbi részében tanulunk.

Egész típusú változók értékének növelése és csökkentése

Amikor PHP-ben programozunk, gyakran kerülünk olyan helyzetbe, hogy egy egész típusú változó értékét kell eggyel növelnünk vagy csökkentenünk. Ez jellemzően abban az esetben fordul elő, amikor egy ciklusban számolunk valamit. Már két módját is tanultuk annak, hogyan tehetjük ezt meg. Egyrészt lehetőségünk van az összeadó műveletjellel növelni a változó értékét:

```
$x = $x + 1; // $x értéke eggyel nő
```

De használhatunk összetett értékadó-összeadó műveletjelet is:

```
$x += 1; // $x értéke eggyel nő
```

Az eredmény mindkét esetben `$x`-be kerül. Mivel az ilyen típusú kifejezések nagyon gyakoriak, ezért a PHP (a C nyelv mintájára) biztosít egy különleges műveletet, amely lehetőséget ad arra, hogy egy egész típusú változóhoz hozzáadjunk egyet vagy kivonjunk belőle egyet. A megfelelő műveletjelek a növelő, illetve csökkentő operátorok. Ezeknek utótagként (poszt-inkrementáló és poszt-dekrementáló) és előtagként (pre-inkrementáló és pre-dekrementáló) használt változata is létezik. Az utótagként használt növelő műveletjel a változó neve után írt két pluszjelből áll.

```
$x++; // $x értéke eggyel nő
```

Ha hasonló módon két mínuszjelet írunk a változó neve után, a változó értéke eggyel csökken.

```
$x--; // $x értéke eggyel csökken
```

Ha a növelő vagy csökkentő műveletjelet feltételes kifejezésen belül használjuk, fontos, hogy az operandus értéke csak a feltétel kiértékelése után változik meg:

```
$x = 3;  
$x++ < 4; // igaz
```

A fenti példában `$x` értéke 3, amikor a 4-gyel hasonlítjuk össze, így a kifejezés értéke igaz. Miután az összehasonlítás megtörtént, `$x` értéke eggyel nő. Bizonyos körülmények között arra lehet szükség, hogy a változó értéke a kiértékelés előtt csökkenjen vagy nőjön. Erre szolgálnak az előtagként használt változatok. Önma-gukban ezek a műveletjelek ugyanúgy viselkednek, mint utótagként alkalmazott formáik, csak a két plusz- vagy mínuszjelet ekkor a változó neve elé kell írunk.

```
++$x;    // $x értéke eggyel nő  
-$x;     // $x értéke eggyel csökken
```

Ha ezek a műveletjelek egy nagyobb kifejezés részei, a változó értékének módosítása a vizsgálat előtt történik meg.

```
$x = 3;  
++$x < 4;    // hamis
```

Ebben a kódrészletben `$x` értéke eggyel nő, mielőtt összehasonlítanánk 4-gyel. Az összehasonlítás értéke `false`, mivel a 4 nem kisebb 4-nél.

A műveletek kiértékelési sorrendje

Az értelmező a kifejezéseket általában balról jobbra olvassa. A több műveletjelet tartalmazó összetettebb kifejezéseknél már bonyolultabb a helyzet. Először vegyünk egy egyszerű esetet:

$4 + 5$

Itt minden tiszta és érthető, a PHP hozzáadja a 4-et az 5-höz. De mi a helyzet a következő esetben?

$4 + 5 * 2$

Itt már egy problémával találkozunk szembe: a kifejezés azt jelenti, hogy „vedd a négyet és az ötöt, add össze őket, majd az eredményt szorozd meg kettővel”, vagyis 18 az értéke? Esetleg azt, hogy „add hozzá a négyhez az öt és a kettő szorzatát”, vagyis az eredmény 14? Ha egyszerűen balról jobbra olvasnánk, akkor az első változatot kellene elfogadnunk. A PHP-ben azonban minden műveletjelhez tartozik egy „elsőbbségi tényező” (prioritás). Mivel a szorzás elsőbbséget élvez az összeadással szemben, így (iskolai tanulmányainkkal összhangban) a második válasz a helyes megoldás.

Természetesen van rá lehetőség, hogy kikényszerítsük, hogy először az összeadás hajtsódjon végre. Erre a zárójelek használata ad lehetőséget:

$(4 + 5) * 2$

Bármilyen sorrendben értékelődjenek is ki a műveletek, ha hosszú kifejezésekkel dolgozunk vagy nem vagyunk biztosak a dolgunkban, érdemes zárójeleket használnunk, így érthetőbbé válik a program és megkíméljük magunkat a hibakeresés fáradalmaitól. A 4.6. táblázatból az ebben az órában tárgyalt műveletek elsőbbségét tudhatjuk meg (csökkenő sorrendben).

4.6. táblázat A műveletek elsőbbsége csökkenő sorrendben.

Műveletjelek

! ++ -- (típusátalakítás)

/ * %

+ - .

< <= => >

== === != !==

&&

||

= += -= /= %= .=

and

xor

or

Mint látjuk, az `or` később értékelődik ki, mint a `||` műveletjel, az `and`-del szemben pedig elsőbbséget élvez a `&&`, így a kisebb prioritású logikai műveletjeleket használva az összetett logikai kifejezések olvasásmódját módosíthatjuk. Ez nem feltétlenül mindig jó ötlet. Bár a következő két kifejezés egyenértékű, a második kifejezés könnyebben olvasható:

```
$x || $y and $z
( $x || $y ) && $z
```

Állandók

A változók rugalmas adattárolási lehetőséget nyújtanak számunkra. Megváltoztathatjuk értéküket, sőt típusukat is, bármely pillanatban. Ha azonban azt szeretnénk, hogy egy adott név alatt tárolt érték ne változzon a program futása során, létrehozhatunk állandókat (konstansokat) is. Ezt a PHP beépített `define()` függvénye segítségével tehetjük meg. Miután az állandót létrehoztuk, annak értékét nem szabad (nem tudjuk) megváltoztatni. Az állandó nevét, mint karakterláncot és az értéket vesszővel elválasztva a zárójeleken belülre kell írunk.

```
define( "ALLANDO_NEVE", 42 );
```

Az állandó értéke természetesen lehet szám, karakterlánc vagy logikai típusú is. Az a szokás, hogy az állandók neve CSUPA NAGYBETŰBŐL áll. Az állandók neve nem tartalmaz dollárjelet, így az állandók elérése csupán a név leírásából áll.

A 4.7. példaprogramban láthatunk egy példát állandók elérésére és használatára.

4.7. program Állandó létrehozása

```
1: <html>
2: <head>
3: <title>4.7. program Állandó létrehozása</title>
4: </head>
5: <body>
6: <?php
7: define ( "FELHASZNALO", "Virág" );
8: print "Üdvözlöm ".FELHASZNALO;
9: ?>
10: </body>
11: </html>
```

Figyeljük meg, hogy az állandónak a karakterláncba ágyazásakor összefűzést kellett használnunk. Ez azért szükséges, mert az értelmező nem tudja megkülönböztetni a kettős idézőjelbe írt egyszerű szöveget az állandók nevétől.

Minden programban használható állandók

A PHP néhány beépített állandót automatikusan biztosít. Ilyen például a `__FILE__`, amely az értelmező által beolvasott fájl nevét tartalmazza. A `__LINE__` az aktuális sor számát tartalmazza. Ezek az állandók hibaüzeneteink kiírásánál hasznosak. Az éppen használt PHP változatszámát többek között a `PHP_VERSION` állandóból tudhatjuk meg. Ez akkor lehet előnyös, ha egy program csak a PHP egy adott változatával futtatható.

Összefoglalás

Ebben az órában végigvettük a PHP nyelv néhány alapvető tulajdonságát. Tanultunk a változókról és arról, hogyan rendelhetünk hozzájuk értéket. Hallottunk a dinamikus, vagyis „változó” változókról. Megtanultuk azt is, hogyan kell a változó értékének másolata helyett a változókra hivatkozni. Új műveletjeleket ismerünk meg és megtanultuk, hogyan kell azokat összetettebb kifejezésekké összegyűjteni. Végül megtanultuk, hogyan kell állandókat létrehozni és használni.

Kérdések és válaszok

Mikor kell tudnunk egy változó típusát?

Előfordul, hogy a változó típusa behatárolja, milyen műveleteket végezhetünk vele. Például mielőtt egy matematikai kifejezésben használunk egy változót, megnézhetjük, hogy egyáltalán egész vagy lebegőpontos számot tartalmaz-e. Az ehhez hasonló kérdésekkel kicsit később, a tizenhatodik fejezetben foglalkozunk.

Muszáj követnünk a változók nevére vonatkozó szokásokat?

A cél mindig az, hogy a program egyszerűen olvasható és érthető legyen. Az olyan változónevek, mint az \$abc12345 nem mondanak semmit a változó programbeli szerepéről és elgépelni is könnyű azokat. Ezért érdemes rövid és jellemző neveket választanunk.

Az \$f név, bár rövid, bizonyára nem árul el semmit a változó szerepéről.

Ezt most talán nehéz elhinni, de amikor egy hónap elteltével próbáljuk meg folytatni a program írását, majd megvizsgáljuk. A \$fajlnev már sokkal jobb választás.

Meg kell tanulnunk a műveletjelek kiértékelési sorrendjét?

Nincs semmi akadálya, hogy megtanuljuk, de tartogassuk energiánkat fontosabb dolgokra. Használjunk zárójeleket a kifejezésekben, így programunk jobban olvasható lesz és nem kell törődnünk a kiértékelési sorrenddel.

4

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Az alábbiak közül melyek NEM lehetnek változónevek?
`$egy_felhasznalo_altal_elkuldott_ertek`
`$44444444444xyz`
`$xyz44444444444`
`$_____szamlalo_____`
`$az_elso`
`$fajl-nev`
2. Hogyan használhatjuk fel az alábbi változót dinamikus változó létrehozására?
 A változónak adjuk a 4 értéket! Hogyan érhetjük el az új változót?
`$en_valtozom = "dinamikus";`
3. Milyen kimenetet eredményez az alábbi programsor?
`print gettype("4");`

4. Mit ír ki az alábbi néhány sor?

```
$proba_valtozo = 5.4566;  
settype ( $proba_valtozo, "integer");  
print $proba_valtozo;
```
5. Az alábbi sorok melyike nem tartalmaz kifejezést?

```
4;  
gettype(44);  
5/12;
```
6. Az előző kérdésben szereplő sorok melyikében van műveletjel?
7. Az alábbi kifejezés milyen értéket ad vissza?

```
5 < 2
```

Milyen típusú a kérdéses érték?

Feladatok

1. Készítsünk programot, amely legalább öt különböző változót tartalmaz. Adjunk nekik különböző típusú értékeket, majd használjuk a `gettype()` függvényt a változók típusainak meghatározására.
2. Rendeljünk értéket két változóhoz. Használjuk az összehasonlító műveleteket annak eldöntésére, hogy az első változó értéke
 - azonos-e a második változó értékével
 - kisebb-e a második változó értékénél
 - nagyobb-e a második változó értékénél
 - kisebb-e a második változó értékénél vagy egyenlő-e azzal

Írassuk ki az összehasonlítások eredményét a böngészőbe!
Változtassuk meg a változók értékét és futtassuk le újból a programot!



5. ÓRA

Vezérlési szerkezetek

Az előző órában létrehozott programok minden futtatáskor ugyanazt az eredményt adták, mindig ugyanazok az utasítások hajtottak végre ugyanabban a sorrendben. Ez nem biztosít túl nagy teret a rugalmasságnak.

Ebben az órában néhány olyan szerkezettel ismerkedünk meg, melyek segítségével programunk alkalmazkodhat a körülményekhez. A következőket tanuljuk meg:

- Hogyan használjuk az `if` szerkezetet arra, hogy bizonyos sorok csak adott feltételek teljesülése mellett hajtsanak végre?
- Hogyan adhatunk meg csak bizonyos feltételek nem teljesülése esetén végrehajtandó műveleteket?
- Hogyan használjuk a `switch` utasítást, hogy egy kifejezés értékétől függően hajtsunk végre utasításokat?
- Hogyan ismétljük egy kódrészlet végrehajtását a `while` utasítás segítségével?
- Hogyan készíthetünk elegánsabb ciklusokat a `for` utasítás segítségével?
- Hogyan lépünk ki a ciklusokból?
- Hogyan ágyazzuk egymásba a ciklusokat?

Elágazások

A legtöbb program feltételeket értékel ki és azok eredményének megfelelően változtatja viselkedését. A lehetőséget, hogy a PHP oldalak tartalma dinamikussá váljék, az teszi lehetővé, hogy a programok kimenete bizonyos feltételektől függjön. A legtöbb programozási nyelvhez hasonlóan a PHP 4-es változata is biztosítja erre a célra az `if` utasítást.

Az `if` utasítás

Az `if` utasítás kiértékeli a zárójelek közötti kifejezést. Ha a kifejezés értéke igaz, az utasításhoz tartozó programrész végrehajtódik. Ha a kifejezés hamis, a blokk egyszerűen figyelmen kívül marad. Ez teszi lehetővé a programoknak, hogy döntéseket hozzanak.

```
if ( kifejezés )
{
    // ha a kifejezés értéke igaz,
    // ez a blokk végrehajtódik
}
```

Az 5.1. példaprogramban csak akkor hajtódik végre az `if` utáni rész, ha a kifejezés értéke igaz

5.1. program Az `if` utasítás

```
1: <html>
2: <head>
3: <title>5.1. program Az if utasítás</title>
4: </head>
5: <body>
6: <?php
7: $hangulat = "boldog";
8: if ( $hangulat == "boldog" )
9:     {
10:         print "Hurrá, jó kedvem van!";
11:     }
12: ?>
13: </body>
14: </html>
```

A `$hangulat` változó értékét a `"boldog"`-gal az összehasonlító művelettel (`==`) segítségével hasonlítottuk össze. Ha egyeznek, a kifejezés értéke igaz, így az `if` kifejezéshez tartozó programblokk végrehajtódik. Bár az előbbi programban a `print` utasítást kapcsos zárójelek közé zártuk, ez csak akkor szükséges, ha az `if` kifejezéstől függően több utasítást akarunk végrehajtani. A következő két sor így elfogadható:

```
if ( $hangulat == "boldog" )
    print "Hurrá, jó kedvem van!";
```

Ha a `$hangulat` értékét `"szomorú"`-ra változtatjuk és újból lefuttatjuk programunkat, az `if` kifejezés értéke hamis lesz, így a `print` utasítás nem hajtódik végre és programunk nem ad kimenetet.

Az if utasítás else ága

Amikor az `if` utasítást használjuk, gyakran szeretnénk, hogy legyen egy olyan alternatív programrész, amely akkor hajtódik végre, ha a vizsgált feltétel nem igaz. Ezt úgy érhetjük el, hogy az `if` utasítás programrésze után kiírjuk az `else` kulcszót, majd az alternatív programrészt. A séma a következő:

```
if (feltétel)
{
    // itt következik az a programrész, amely akkor kerül
    // végrehajtásra, ha a feltétel értéke igaz
}
else
{
    // itt pedig az a programrész található, amely akkor fut le,
    // ha a feltétel értéke hamis
}
```

Az 5.2. példaprogram az 5.1. példaprogram kiegészített változata. Egy olyan programblokkot tartalmaz, amely akkor hajtódik végre, ha a `$hangulat` értéke nem `"boldog"`.

5.2. program Az else ággal kiegészített if utasítás

```
1: <html>
2: <head>
3: <title>5.2. program Az else ággal kiegészített if
   utasítás</title>
4: </head>
```

5.2. program (folytatás)

```
5: <body>
6: <?php
7: $hangulat = "szomorú";
8: if ( $hangulat == "boldog" )
9:     {
10:    print "Hurrá, jó kedvem van!";
11:    }
12: else
13:    {
14:    print "$hangulat vagyok, nem boldog.";
15:    }
16: ?>
17: </body>
18: </html>
```

A példában a `$hangulat` értéke "szomorú", ami persze nem "boldog", így az `if` utasítás feltétele hamis lesz, vagyis az első programrész nem kerül végrehajtásra. Az `else`-et követő programblokk azonban lefut és a böngészőbe a "szomorú vagyok, nem boldog." szöveg kerül.

Az `if` utasítás `else` ágának segítségével programunkban kifinomultabb döntéseket hozhatunk, de egy feltétel eredménye alapján még így is csak kétféle dolgot tehetünk. A PHP 4 azonban többre képes: több kifejezés értéke alapján sokféleképp reagálhatunk.

Az if utasítás elseif ága

Mielőtt az `else` ágban alternatív kódrészt adnánk meg, több kifejezés értékétől függően – az `if` - `elseif` - `else` szerkezet segítségével – a programmal mást és mást végeztethetünk. A használandó utasításforma a következő:

```
if ( feltétel )
{
    // ez a rész akkor fut le, ha a feltétel igaz
}
elseif ( másik feltétel )
{
    // ez a rész akkor fut le, ha a másik feltétel igaz,
    // és minden előző feltétel hamis
}
// itt még tetszőleges számú elseif rész következhet
```

```
else
{
    // ez a rész akkor kerül végrehajtásra, ha egyik
    // feltétel sem volt igaz
}
```



A Perl nyelvben gyakorlattal rendelkezők figyeljenek rá, hogy a kulcsszót itt `elseif`-nek hívják!

Ha az első feltétel értéke hamis, a hozzá tartozó programrész nem kerül végrehajtásra. Ezután a PHP megvizsgálja az `elseif` kifejezés értékét. Ha a kifejezés igaz, a hozzá tartozó programblokk fut le. Végül, ha egyik feltétel sem igaz, az `else` utáni rész kerül végrehajtásra. Annyi `elseif`-et írhatunk a programba, amennyi csak jólesik. Sőt, ha nincs szükségünk `else` ágra, vagyis olyan programrészre, amely akkor hajtódik végre, ha egyik feltétel sem igaz, akár el is hagyhatjuk.

Az 5.3. példaprogram az előzőeket egészíti ki egy `elseif` ággal.

5.3. program Egy `else` és `elseif` ággal bővített `if` utasítás

```
1: <html>
2: <head>
3: <title>5.3. program Egy else és elseif ággal bőví-
   tett if utasítás</title>
4: </head>
5: <body>
6: <?php
7: $hangulat = "szomorú";
8: if ( $hangulat == "boldog" )
9:     {
10:         print "Hurrá, jó kedvem van!";
11:     }
12: elseif ( $hangulat == "szomorú" )
13:     {
14:         print "Szomorú vagyok.";
15:     }
16: else
17:     {
18:         print "Sem boldog, sem szomorú nem vagyok, hanem
           $hangulat.";
```

5.3. program (folytatás)

```
19:      }  
20: ?>  
21: </body>  
22: </html>
```

A `$hangulat` értéke itt "szomorú". Ez nem azonos a "boldog"-gal, ezért az első blokk nem kerül végrehajtásra. Az `elseif` kifejezés a `$hangulat` változó értékét hasonlítja össze a "szomorú" szöveggel. Mivel az egyenlőség fennáll, ehhez a feltételhez tartozó programrész hajtódik végre.

A switch utasítás

A `switch` utasítás egy lehetséges módja annak, hogy egy kódrészletet egy kifejezés értékétől függően hajtsunk végre. Van azonban néhány különbség az imént tanult `if` és a `switch` között. Az `if`-et az `elseif`-fel használva több kifejezés értékétől tehetjük függővé, hogy mi történjen. A `switch` esetében a program csak egy kifejezést vizsgál meg és annak értékétől függően különböző sorokat futtat. A kifejezésnek egyszerű típusnak kell lennie (szám, karakterlánc vagy logikai érték). Az `if` feltétele csak igaz vagy hamis lehet, a `switch` kifejezését akárhány értékkel összehasonlíthatjuk. De nézzük inkább az általános szerkezetet:

```
switch ( kifejezés )  
{  
    case érték1:  
        // ez történjen, ha kifejezés értéke érték1  
        break;  
    case érték2:  
        // ez történjen, ha kifejezés értéke érték2  
        break;  
    default:  
        // ez történjen, ha a kifejezés értéke  
        // egyik felsorolt értékkel sem egyezett meg  
        break;  
        // az ördög nem alszik, jobban járunk, ha kitesszük  
        // ezt a "felesleges" break utasítást  
}
```

A `switch` utasítás kifejezése gyakran egyszerűen egy változó. A `switch`-hez tartozó programblokkban vannak a `case` címkék. Az utánuk írt érték kerül összehasonlításra a `switch` kifejezésének értékével. Ha értékük megegyezik, a program ott folytatódik, a `break` utasítás pedig azt eredményezi, hogy a program futása a `switch`

blokkja utáni részre kerül. Ha a `break`-et elfelejtjük, a program átlép a következő `case` kifejezéshez tartozó programrészre és azt is végrehajtja. Ha a kifejezés értéke egyik előző értékkel sem egyezik és a `switch` blokkján belül szerepel `default` címke, akkor az utána levő programrész kerül végrehajtásra. Ez sokszor bosszantó, de néha hasznos is lehet. Vegyük a következő kis példát.

```
switch( $hetnapja )
{
    case "Péntek":
        print "Kikapcsolni a vekkert, holnap nem kell
            dolgozni<br>";
    case "Hétfő":
    case "Szerda":
        print "Ma délelőtt dolgozom<br>";
    break;
    case "Kedd":
    case "Csütörtök":
        print "Ma délután dolgozom<br>";
    break;
    case "Vasárnap":
        print "Bekapcsolni a vekkert!<br>";
    case "Szombat":
        print "Hurrá, szabadnap!<br>";
    break;
    default:
        print "Azt hiszem ideje lenne egy új programo-
            zót és/vagy egy jobb naptárat
            keríteni<br>";
    break;
}
```

A fenti kis program azt közli, mikor kell mennünk dolgozni és az ébresztőóra kezelésében is segít. A programot úgy építettük fel, hogy több esetben is ugyanazt a kódot kelljen végrehajtani, így hasznos, hogy nem adtuk meg minden `case` címkénél a `break` utasítást. Elég gyakori azonban, hogy a kezdő programozó elfelejti megadni a `break` utasítást. Hasonló helyzetekben jusson eszünkbe ez a példa.



A `case` címkével elkezdett részeket ne felejtjük el `break` utasításokkal lezárni. Ha ezt nem tesszük meg, a program a következő `case` részt is végrehajtja és végül a `default` utáni rész is lefut. A legtöbb esetben nem ez a `switch` kívánatos viselkedése.

Az 5.4. példaprogram a korábbi, `if`-fel megoldott példát alakítja át a `switch` segítségével.

5.4. program A switch utasítás

```
1: <html>
2: <head>
3: <title>5.4. program A switch utasítás</title>
4: </head>
5: <body>
6: <?php
7: $hangulat = "szomorú";
8: switch ( $hangulat )
9:     {
10:    case "boldog":
11:        print "Hurrá, jó kedvem van!";
12:        break;
13:    case "szomorú":
14:        print "Szomorú vagyok.";
15:        break;
16:    default:
17:        prin "Sem boldog, sem szomorú nem
            vagyok, hanem $hangulat.";
18:    }
19: ?>
20: </body>
21: </html>
```

A `$hangulat` változónak a "szomorú" értéket adtuk. A `switch` utasítás kifejezése ez a változó lesz. Az első `case` címke a "boldog" szöveggel való egyezést vizsgálja. Mivel nincs egyezés, a program a következő `case` címkére ugrik. A "szomorú" szöveg megegyezik a `$hangulat` változó pillanatnyi értékével, így az ehhez tartozó programrész fog lefutni. A programrész végét a `break` utasítás jelzi.

A ?: műveletjel

A `?:` ternális (háromoperandusú) műveletjel egy olyan `if` utasításhoz hasonlít, amely értéket is képes visszaadni. A visszaadott értéket a vizsgált feltétel határozza meg:

```
( feltétel ) ? érték_ha_a_feltétel_igaz : érték_ha_a_feltétel_hamis ;
```

Ha a vizsgált feltétel igaz, a ? és a : közti kifejezés értékét adja, ha hamis, akkor a : utáni. Az 5.5. példaprogram ezt a műveletet használja, hogy egy változó értékét a \$hangulat változó értékétől függően állítsa be.

5.5. program A ?: műveletjel használata

```
1: <html>
2: <head>
3: <title>5.5. program A ?: műveletjel
   használata</title>
4: </head>
5: <body>
6: <?php
7: $hangulat = "szomorú";
8: $szoveg = ( $hangulat=="boldog" ) ? "Hurrá,
   jó kedvem van!" : "$hangulat vagyok, nem boldog.";
9: print "$szoveg";
10: ?>
11: </body>
12: </html>
```

A \$hangulat-ot "szomorú"-ra állítottuk, majd megnéztük, hogy értéke "boldog"-e. Mivel ez nem igaz, a \$szoveg változó értéke a : utáni szöveg lesz. Az e műveletet használó programokat eleinte nehéz megérteni, de a művelet hasznos lehet, ha csak két lehetőség közül lehet választani és szeretünk tömör programot írni.

Ciklusok

Most már láttuk, hogyan kell döntések eredményétől függően különböző programrészleteket futtatni. PHP programokkal arra is képesek vagyunk, hogy megmondjuk, hányszor kell lefuttatni egy adott programrészt. Erre valók a ciklusok. Segítségükkel elérhetjük, hogy egyes programrészletek ismétlődjenek. Szinte kivétel nélkül igaz, hogy egy ciklus addig fut, amíg egy feltétel teljesül, vagy meg nem mondjuk, hogy fejeződjön be az ismétlés.

A while ciklus

A `while` ciklus szerkezete rendkívül hasonlít az `if` elágazáséhez:

```
while ( feltétel )
{
    // valamilyen tevékenység
}
```

Amíg a `while` feltétele igaz, a hozzá tartozó programrész újból és újból végrehajthatódik. A programrészben belül általában megváltoztatunk valamit, ami hatással lesz a `while` feltételére; ha ezt nem tesszük meg, a ciklusunk a végtelenségig futni fog. Az 5.6. példaprogram a `while` ciklus segítségével írja ki a kettes szorzótáblát.

5.6. program A while ciklus

```
1: <html>
2: <head>
3: <title>5.6. program A while ciklus</title>
4: </head>
5: <body>
6: <?php
7: $szamlalo = 1;
8: while ( $szamlalo <= 12 )
9:     {
10:        print "$szamlalo kétszerese " . ($szamlalo*2) . "<br>";
11:        $szamlalo++;
12:    }
13: ?>
14: </body>
15: </html>
```

A példában létrehoztuk a `$szamlalo` nevű változót. A `while` kifejezés feltétele megvizsgálja, hogy ez a változó mekkora. Ha az érték nem nagyobb, mint 12, a ciklus folytatódik (vagy elkezdődik). A ciklusban a `$szamlalo` értéke és annak kétszerese kiírásra kerül, majd a `$szamlalo` értéke eggyel nő. Ez az utasítás rendkívül fontos, mert ha elfelejtjük, a `while` feltétele soha nem lesz hamis, ezért végtelen ciklusba kerülünk.

A do..while ciklus

A do..while ciklus kicsit hasonlít a while-hoz. A lényegi különbség abban van, hogy ebben a szerkezetben először hajtódik végre a kód és csak azután értékelődik ki a feltétel:

```
do
{
    // végrehajtandó programrész
} while ( feltétel );
```



A do..while ciklus feltételét tartalmazó zárójel után mindig ki kell tenni a pontosvesszőt.

Ez a ciklus akkor lehet hasznos, ha mindenképpen szeretnénk, hogy a ciklushoz tartozó programrész még akkor is legalább egyszer lefusson, ha a feltétel már az első végrehajtáskor hamis. Az 5.7. példaprogram a do..while szerkezet egy alkalmazását mutatja be. A programrész mindig legalább egyszer lefut.

5.7. program A do..while ciklus

```
1: <html>
2: <head>
3: <title>5.7. program A do..while ciklus</title>
4: </head>
5: <body>
6: <?php
7: $szam = 1;
8: do
9:     {
10:    print "Végrehajtások száma: $szam<br>\n";
11:    $szam++;
12:    } while ( $szam > 200 && $szam < 400 );
13: ?>
14: </body>
15: </html>
```

A do..while ciklus megnézi, hogy a \$szam változó értéke 200 és 400 között van-e. Mivel a \$szam változónak az 1 kezdeti értéket adtuk, így a feltétel hamis. Ennek ellenére a programblokk egyszer végrehajtódik, mégpedig a feltétel kiértékelése előtt, ezért a böngészőben egy sor kiírásra kerül.

A for ciklus

A `for` semmi újat nem nyújt a `while` ciklushoz képest. A `for` ciklus használatával azonban sokszor takarosabb, biztonságosabb módon közelíthetjük meg ugyanazt a problémát. Korábban – az 5.6. példaprogramban – létrehoztunk egy változót a `while` cikluson kívül, majd a `while` kifejezése megvizsgálta ennek a változónak az értékét. Végül a változó értékét a ciklus végén eggyel növeltük. A `for` ciklus mindezt egyetlen sorban teszi lehetővé. Ez tömörebb programot eredményez és ritkábban fordulhat elő, hogy a változót elfelejtjük növelni, ami végtelen ciklust okoz.

```
for ( változó_hozzárendelése; feltétel; számláló_növelése )
{
    // a végrehajtandó programblokk
}
```

Az ezt megvalósító egyenértékű `while`:

```
változó_hozzárendelése;
while ( feltétel )
{
    // a végrehajtandó programblokk
    számláló_növelése;
}
```

A zárójelekben levő kifejezéseket pontosvesszővel kell elválasztanunk egymástól. Az első kifejezés rendszerint egy számlálónak ad kezdeti értékét, a második egy feltétel, ami alapján eldől, hogy folytatódik-e a ciklus; a harmadik egy számlálót növelő utasítás. Az 5.8. példaprogram az 5.6. példaprogram `for` ciklusos megoldása.



A feltétel, a számláló_növelése és a változó_hozzárendelése paraméterek helyére bármilyen érvényes PHP állítás írható. A kezdők bánjanak óvatosan ezzel a lehetőséggel.

5.8. program A for ciklus használata

```
1: <html>
2: <head>
3: <title>5.8. program A for ciklus használata</title>
4: </head>
5: <body>
6: <?php
```

5.8. program (folytatás)

```
7: for ( $szamlalo = 1; $szamlalo <= 12; $szamlalo++ )
8:     {
9:         print "$szamlalo kétszerese " . ( $szamlalo * 2
           ) . "<br>";
10:    }
11: ?>
12: </body>
13: </html>
```

Az 5.6. és az 5.8. példaprogram kimenete teljesen azonos. A `for` ciklus használata a programot összefogottabbá tette. Mivel a `$szamlalo` nevű változó létrehozása és módosítása egy sorban van, így a ciklus egészének logikája első látásra világos. A `for` zárójelében az első utasítás a `$szamlalo` változót egyre állítja. A feltételes kifejezés megnézi, hogy a `$szamlalo` értéke nem nagyobb-e, mint 12. Az utolsó kifejezés a `$szamlalo`-t eggyel növeli.

Amikor a program a `for` ciklushoz ér, megtörténik az értékadás, majd rögtön utána a feltétel kiértékelése. Ha a feltétel igaz, a ciklus végrehajtódik, majd a `$szamlalo` értéke eggyel nő és a feltétel kiértékelése újból végrehajtódik. A folyamat addig folytatódik, amíg a feltétel hamissá nem válik.

Ciklus elhagyása a `break` utasítás segítségével

A `while` és `for` ciklusok lehetőséget biztosítanak arra, hogy egy beépített feltételes kifejezés segítségével kilépjünk belőlük. A `break` utasítás lehetővé teszi, hogy más feltételektől függően megszakítsuk egy ciklus futását. Ez jó lehet például hibakezeléskor vagy hibák megelőzésekor. Az 5.9. példaprogram egy egyszerű `for` ciklusból áll, amely egy nagy számot egy egyre növekvő számmal oszt el és a művelet eredményét meg is jeleníti.

5.9. program Egy `for` ciklus, amely a 4000-et tíz, egyre növekvő számmal osztja el

```
1: <html>
2: <head>
3: <title>5.9. program A for ciklus használata
   2.</title>
4: </head>
5: <body>
6: <?php
```

5.9. program (folytatás)

```
7: for ( $szamlalo=1; $szamlalo <= 10, $szamlalo++ )
8:     {
9:         $seged = 4000/$szamlalo;
10:        print "4000 $szamlalo részre osztva $seged.<br>";
11:    }
12: ?>
13: </body>
14: </html>
```

A példában a `$szamlalo` nevű változónak az 1 kezdeti értéket adjuk. A `for` utasítás feltételes kifejezése ellenőrzi, hogy a `$szamlalo` nem nagyobb-e, mint 10. A ciklus belsejében a 4000-et elosztjuk a `$szamlalo`-val és az eredményt kiírjuk a böngészőbe.

Ez elég célrätörőnek tűnik. De mi a helyzet akkor, ha a `$szamlalo` értéke a felhasználótól származik? A változó értéke lehet negatív szám vagy akár szöveg is. Vegyük az első esetet: változtassuk meg a `$szamlalo` kezdőértékét 1-ről -4-re. Így a ciklusmag ötödik futtatása során nullával kellene osztani, ami nem elfogadható. Az 5.10. példaprogram ezt azzal védi ki, hogy a ciklus futását befejezi, ha a `$szamlalo` változóban a nulla érték van.

5.10. program A break utasítás használata

```
1: <html>
2: <head>
3: <title>5.10. program A break utasítás
   használata</title>
4: </head>
5: <body>
6: <?php
7: $szamlalo = -4;
8: for ( ; $szamlalo <= 10; $szamlalo++ )
9:     {
10:        if ( $szamlalo == 0 )
11:            break;
12:        $seged = 4000/$szamlalo;
13:        print "4000 $szamlalo részre osztva $seged.<br>";
14:    }
15: ?>
16: </body>
17: </html>
```



A nullával való osztás nem eredményez végzetes hibát a PHP 4-ben, csak egy figyelmeztető üzenet jelenik meg a böngészőben. A program futása folytatódik.

Egy `if` utasítással megvizsgáljuk a `$szamlalo` értékét. Ha nullával egyenlő, azonnal kilépünk a ciklusból; a program ekkor a `for` ciklus utáni sorokat kezdi feldolgozni. Figyeljük meg, hogy a `$szamlalo`-t a cikluson kívül hoztuk létre, hogy olyan helyzetet utánozzunk, amikor a `$szamlalo` értéke „kívülről” származik, például egy űrlapról vagy egy adatbázisból.



A `for` ciklus fejéből bármelyik kifejezés elhagyható, de figyelniünk kell arra, hogy a pontosvesszőket mindig kiírjuk.

Következő ismétlés azonnali elkezdése a `continue` utasítás segítségével

A `continue` utasítás segítségével az éppen folyó ismétlést befejezhetjük, mégpedig úgy, hogy ez ne eredményezze az egész ciklusból való kilépést, csak a következő ismétlés kezdetét jelentse. Az 5.10. példában a `break` utasítás használata kicsit drasztikus volt. Az 5.11. példaprogramban a nullával való osztást úgy kerüljük el, hogy közben programunk nem lép ki az egész ciklusból.

5.11. program A `continue` utasítás használata

```
1: <html>
2: <head>
3: <title>5.11. program A continue utasítás
   használata</title>
4: </head>
5: <body>
6: <?php
7: $szamlalo = -4;
8: for ( ; $szamlalo <= 10; $szamlalo++ )
9:     {
10:         if ( $szamlalo == 0 )
11:             continue;
12:         $seged = 4000/$szamlalo;
13:         print "4000 $szamlalo részre osztva $seged.<br>";
14:     }
```

5.11. program (folytatás)

```
15: ?>
16: </body>
17: </html>
```

Itt a `break` utasítást `continue`-ra cseréltük. Ha a `$szamlalo` értéke nullával egyenlő, az éppen folyó ismétlés véget ér, a végrehajtás pedig rögtön a következőre kerül.



A `break` és `continue` utasítások a ciklus logikáját bonyolultabbá, a programot pedig nehezebben olvashatóvá teszik, így furcsa programhibákat idézhetnek elő, ezért óvatosan használandók.

Egymásba ágyazott ciklusok

A ciklusok törzsében is lehetnek ciklusok. Ez a lehetőség különösen hasznos, ha futási időben előállított HTML táblázatokkal dolgozunk. Az 5.12. példaprogramban két egymásba ágyazott `for` ciklus segítségével egy szorzótáblát írunk ki a böngészőbe.

5.12. program Két `for` ciklus egymásba ágyazása

```
1: <html>
2: <head>
3: <title>5.12. program Két for ciklus egymásba
   ágyazása</title>
4: </head>
5: <body>
6: <?php
7: print "<table border=1>\n"; // HTML táblázat kezdete
8: for ( $y=1; $y<=12; $y++ )
9:     {
10:        print "<tr>\n"; // sor kezdete a HTML táblázatban
11:        for ( $x=1; $x<=12; $x++ )
12:            {
13:                print "\t<td>"; // cella kezdete
14:                print ($x*$y);
15:                print "</td>\n"; // cella vége
16:            }
17:        print "</tr>\n"; // sor vége
18:    }
```

5.12. program (folytatás)

```
19: print "</table>\n"; // táblázat vége
20: ?>
21: </body>
22: </html>
```

A külső `for` ciklus az `$y` változónak az 1 kezdeti értéket adja. A ciklus addig fog futni, amíg a változó nem nagyobb 12-nél, az utolsó kifejezés pedig biztosítja, hogy `$y` értéke minden ismétlés után eggyel nőjön. A ciklus törzse minden ismétlés során kiír egy `tr` (`table row` – táblázatsor) HTML elemet, majd egy újabb `for` ciklus kezdődik. A belső ciklus az `$x` változó értékét a külső ciklushoz hasonlóan végigfuttatja 1-től 12-ig. A ciklus törzsében egy `td` (`table data` – táblázatcella) elemet ír ki, amelybe az `$x*$y` érték kerül. Az eredmény egy ízlésesen formázott szorzótábla lesz.

Összefoglalás

Ebben az órában a vezérlési szerkezetekről tanultunk és arról, hogyan segítenek minket programjaink változatosabbá és rugalmasabbá tételében. A legtöbb megtanult szerkezet újból és újból meg fog jelenni a könyv hátralevő részében. Megtanultuk, hogyan hozzuk létre `if` utasításokat, hogyan egészítjük ki azokat `elseif` és `else` ágakkal. Hallottunk arról, hogyan használjuk a `switch` utasítást, hogy egy kifejezés adott értékei esetén más és más történjen. Tanultunk a ciklusokról, pontosabban a `while` és a `for` ciklusokról, és arról, hogy a `break` és a `continue` utasítások segítségével hogyan léphetünk ki végleg a ciklusból, illetve hogyan hagyhatunk ki egy-egy ismétlést. Végül megtanultuk, hogyan kell ciklusokat egymásba ágyazni és erre gyakorlati példát is láttunk.

Kérdések és válaszok

A vezérlési szerkezetek feltételes kifejezéseinek mindenképpen logikai értéket kell adniuk?

Végző soron igen, bár a feltételes kifejezéseknél a kiértékelés szempontjából minden, ami nulla, üres karakterlánc vagy nem meghatározott változó, `false`-nak, minden egyéb `true`-nak számít.

A vezérlési szerkezetekhez tartozó programblokkot mindig kapcsos zárójelbe kell tenni?

Ha a programblokk csak egy utasításból áll, a kapcsos zárójel elhagyható, de ezt tenni nem célszerű, mert ha a programblokkot új utasítással egészítjük ki, véletlenül hibákat idézhetünk elő.

Ez az óra bemutatta az összes ciklust?

Nem, a hetedik, tömbökkel foglalkozó órában találkozunk még a `foreach` ciklussal is, melynek segítségével egy tömb elemein haladhatunk végig.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan használnánk az `if` vezérlési szerkezetet olyan program írására, hogy ha az `$letkor` változó értéke 18 és 35 között van, az "Üzenet fiataloknak" szöveget írja ki? Ha az `$letkor` értéke bármi más, az "Általános üzenet" szöveg jelenjen meg a böngészőben.
2. Hogyan egészíthetnénk ki az első kérdésbeli programunkat úgy, hogy az "Üzenet gyerekeknek" jelenjen meg akkor, ha az `$letkor` változó értéke 1 és 17 között van?
3. Hogyan készítenénk egy `while` ciklust, amely kiírja az 1 és 49 közötti páratlan számokat?
4. Hogyan valósítanánk meg az előző kérdésbeli programot `for` ciklus segítségével?

Feladatok

1. Nézzük végig a vezérlési szerkezetek utasításformáját! Gondoljuk végig, hogyan lehetnek ezek a szerkezetek a segítségünkre!
2. Nézzük meg a `?:` műveletet! Miben különbözik ez a többi vezérlési szerkezettől? Mikor lehet hasznos?



6. ÓRA

Függvények

A függvény a jól szervezett program lelke, mert a programot könnyen olvashatóvá és újrahasznosíthatóvá teszi. Függvények nélkül a nagy programok kezelhetetlenek lennének. Ebben az órában a függvényeket tanulmányozzuk és mutatunk rá néhány példát, hogyan kímélhetnek meg minket az ismétlődésekből adódó pluszmunkától. Az órában a következőket tanuljuk meg:

- Hogyan hozhatunk létre függvényeket?
- Hogyan adjunk át a függvényeinknek értékeket és hogyan kapjuk meg tőlük az eredményt?
- Hogyan hívunk meg függvényeket dinamikusán, változóban tárolt karakterlánc segítségével?
- Hogyan érjük el a függvényekből a globális változókat?
- Hogyan érjük el, hogy függvényeinknek „emlékezete” legyen?
- Hogyan adjunk át a függvényeknek hivatkozásokat?

Mit nevezünk függvénynek?

A függvényt egy gépnek tekinthetjük. A gép a bele töltött nyersanyagokkal addig dolgozik, amíg a kívánt terméket elő nem állítja vagy el nem éri kitűzött célját. A függvény értékeket vesz át tőlünk, feldolgozza azokat és végez velük valamit (például kiírja az eredményt a böngészőbe) vagy visszaad egy értéket, esetleg mindkettőt.

Ha a kedves Olvasónak egy süteményt kell csinálnia, maga süti meg. De ha több ezret, akkor esetleg készít vagy beszerz egy süteménysütő gépezetet. Hasonlóképp, amikor elhatározzuk, hogy függvényt írunk, a legfontosabb szempont, amit mérlegelnünk kell, az, hogy az ismétlődések csökkentésével akkorává zsugorodik-e a program, hogy rövidebb lesz a függvény használatával.

A függvény valójában egy zárt, önálló kódrészlet, melyet programunkból meghívhatunk. Amikor meghívjuk, a függvény törzse lefut. A függvénynek feldolgozás céljából értékeket adhatunk át. Amikor a függvény véget ér, a hívónak egy értéket ad vissza.

ÚJDONSÁG

A függvény olyan kódrészlet, amely nem közvetlenül hajtódik végre, hanem a programból hívhatjuk meg: onnan, ahol épp szükség van rá.

A függvények lehetnek beépítettek vagy felhasználó által megadottak. Működésükhöz szükségük lehet információkra és többnyire értéket adnak vissza.

Függvények hívása

Kétféle függvény létezik: a nyelvbe beépített függvény és az általunk létrehozott függvény. A PHP 4-ben rengeteg beépített függvény van. A könyv legelső PHP oldala egyetlen függvényhívásból állt:

```
print ("Hello Web!");
```



A print abból a szempontból nem jellegzetes függvény, hogy paramétereit nem kell zárójelbe tenni. A

```
print ("Hello Web!");
```

és

```
print "Hello Web!";
```

egyenként helyes megoldások. Ez egy különleges függvény. Szinte az összes többi függvélynél kötelező a zárójel; akár kell paramétert átadnunk, akár nem.

A fenti példában a `print()` függvényt a "Hello Web!" szövegparaméterrel hívtuk meg. A program a karakterlánc kiírását hajtja végre. A függvényhívás egy függvénynévből (ebben az esetben ez a `print`) és az utána tett zárójelekből áll. Ha a függvénynek információt szeretnénk átadni, azt a függvény utáni zárójelbe tesszük. Az információt, amit ily módon adunk át a függvénynek, paraméternek hívjuk. Néhány függvénynek több paramétert kell átadni. A paramétereket vesszővel választjuk el.

ÚJDONSÁG

A paraméter a függvénynek átadott érték. A paramétereket a függvényhívás zárójelén belülre kell írunk. Ha több paramétert kell átadnunk, az egyes paramétereket vesszővel kell elválasztanunk egymástól. A paraméterek a függvényeken belül helyi (lokális) változóként érhetők el.

```
valamilyen_fuggveny ( $első_parameter, $második_parameter );
```

A `print()` abból a szempontból tipikus függvény, hogy van visszatérési értéke. A legtöbb függvény, ha nincs „értelmes” visszatérési értéke, információt ad arról, hogy munkáját sikeresen befejezte-e. A `print()` visszatérési értéke logikai típusú (`true`, ha sikeres volt).

Az `abs()` függvény például egy szám típusú paramétert vár és a paraméter abszolútértékét adja vissza. Próbáljuk is ki a 6.1. példaprogrammal!

6.1. program A beépített `abs()` függvény használata

```
1: <html>
2: <head>
3: <title>6.1. program A beépített abs() függvény
   használata</title>
4: </head>
5: <body>
6: <?php
7: $szam = -321;
8: $ujszam = abs( $szam );
9: print $ujszam; // kiírja, hogy "321"
10: ?>
11: </body>
12: </html>
```

Ebben a példában a `$szam` nevű változóhoz a `-321`-es értéket rendeltük, majd átadtuk az `abs()` függvénynek, amely elvégzi a szükséges számításokat és visszaadja az eredményt. Ezt az új értéket rendeljük az `$ujjszam` nevű változóhoz és kiírjuk az eredményt. A feladatot megoldhattuk volna segédváltozók segítségével is, az `abs()` függvényt adva közvetlenül a `print()` paramétereként:

```
print( abs( -321 ) );
```

A felhasználó által megírt függvényeket teljesen hasonló módon kell meghívunk.

Függvények létrehozása

Függvényt a `function` kulcsszó segítségével hozhatunk létre:

```
function valamilyen_fuggveny( $parameter1, $parameter2 )
{
    // itt van a függvény törzse
}
```

A `function` kulcsszót a függvény neve követi, majd egy zárójelpár. Ha a függvény paramétereket igényel, a zárójelbe vesszővel elválasztott változókat kell tenni. A változók értékei a függvényhíváskor átadott paraméterek lesznek. A zárójelpárt akkor is ki kell írni, ha a függvény nem vesz át paramétereket.

A 6.2. példaprogram egy függvényt hoz létre.

6.2. program Függvény létrehozása

```
1: <html>
2: <head>
3: <title>6.2. program Függvény létrehozása</title>
4: </head>
5: <body>
6: <?php
7: function nagyHello()
8:     {
9:         print "<h1>HELLO!</h1>";
10:    }
11: nagyHello();
12: ?>
13: </body>
14: </html>
```

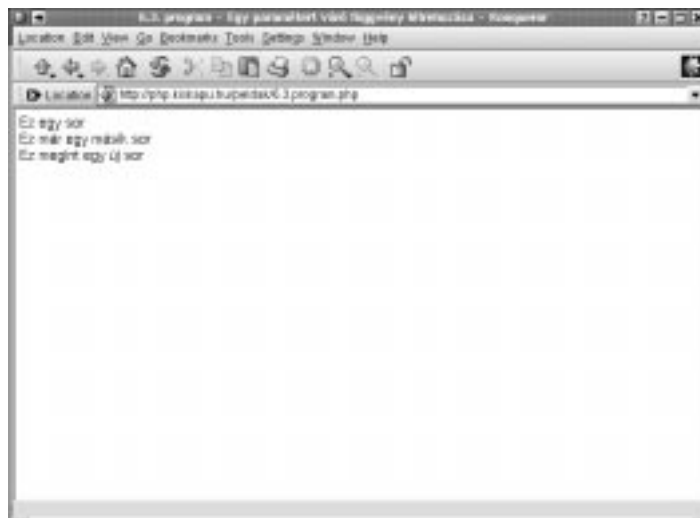
A fenti program egyszerűen kiírja a "HELLO" szöveget egy <H1> HTML elemben. A programban egy olyan nagyHello() nevű függvényt hoztunk létre, amely nem vesz át paramétereket. Ezért hagytuk üresen a zárójelpárt. Bár a nagyHello() működő függvény, mégsem túl hasznos. A 6.3. példaprogramban olyan függvényt láthatunk, amely egy paramétert fogad és valami hasznosat csinál vele.

6.3. program Egy paramétert váró függvény létrehozása

```
1: <html>
2: <head>
3: <title>6.3. program Egy paramétert váró függvény
   létrehozása</title>
4: </head>
5: <body>
6: <?php
7: function sorKiir( $sor )
8:     {
9:         print ("{$sor<br>\n"});
10:    }
11: sorKiir("Ez egy sor");
12: sorKiir("Ez már egy másik sor");
13: sorKiir("Ez megint egy új sor");
14: ?>
15: </body>
16: </html>
```

6.1. ábra

*Függvény, amely egy karakterláncot, majd egy
 HTML elemet ír ki.*



A 6.3. példaprogram kimenetét a 6.1. ábrán láthatjuk. A `sorKiir()` függvény egy karakterláncot vár, ezért tettük a `$sor` nevű változót a zárójelbe. Amit a `sorKiir()` függvény zárójelei közé teszünk, az kerül a `$sor` nevű változóba. A függvény törzsében kiírjuk a `$sor` nevű változót, majd egy `
` elemet és egy újsor karaktert (hogy a HTML kód is olvasható legyen, ne csak a kimenet).

Ha most ki szeretnénk írni egy sort a böngészőbe, akkor meghívhatjuk a `sorKiir()` függvényt, ahelyett, hogy a `print()` függvénnyel oldanánk meg a problémát, így nem kell a sorvégi jeleket minden sor kiírásakor begépelnünk.

Függvények visszatérési értéke

A függvényeknek visszatérési értéke is lehet, ehhez a `return` utasításra van szükség. A `return` befejezi a függvény futtatását és az utána írt kifejezést küldi vissza a hívónak.

A 6.4. példaprogramban létrehozunk egy függvényt, amely két szám összegével tér vissza.

6.4. program Visszatérési értékkel rendelkező függvény

```
1: <html>
2: <head>
3: <title>6.4. program Visszatérési értékkel rendelkező
   függvény</title>
4: </head>
5: <body>
6: <?php
7: function osszead( $elsoszam, $masodikszam )
8:     {
9:         $eredmeny = $elsoszam + $masodikszam;
10:        return $eredmeny;
11:    }
12: print osszead(3,5); // kiírja, hogy "8"
13: ?>
14: </body>
15: </html>
```

A 6.4. példaprogram a 8-as számot írja ki. Az `osszead()` függvényt két paraméterrel kell meghívni (itt a két paraméter a 3 és az 5 volt). Ezek az `$elsoszam` és a `$masodikszam` nevű változókban tárolódnak. Várhatóan az `osszead()` függvény e két változó eredményét adja össze és tárolja az `$eredmeny` nevű változóban. Az `$eredmeny` nevű segédváltozó használatát kiküszöbölhetjük:

```
function összead( $elsoszam, $masodikszam )
{
    return ( $elsoszam + $masodikszam );
}
```

A `return` segítségével értéket és objektumot is visszaadhatunk, vagy esetleg „semmit”. Ha semmit sem adunk meg a `return` után, az csak a függvény futtatásának befejezését eredményezi. A visszatérési érték átadásának módja többféle lehet. Az érték lehet előre „beégetett”

```
return 4;
```

de lehet kifejezés eredménye

```
return ( $a / $b );
```

vagy akár egy másik függvény visszatérési értéke is:

```
return ( masik_fuggveny( $parameter ) );
```

Dinamikus függvényhívások

Lehetőségünk van rá, hogy karakterláncba tegyük egy függvény nevét és ezt a változót pontosan úgy tekintsük, mint ha maga a függvény neve lenne. Ezt a 6.5. példaprogramon keresztül próbálhatjuk ki.

6.5. program Dinamikus függvényhívás

```
1: <html>
2: <head>
3: <title>6.5. program Dinamikus függvényhívás</title>
4: </head>
5: <body>
6: <?php
```

6.5. program (folytatás)

```
7: function koszon()  
8: {  
9:   print "Jó napot!<br>";  
10: }  
11: $fuggveny_tarolo = "koszon";  
12: $fuggveny_tarolo();  
13: ?>  
14: </body>  
15: </html>
```

Itt a `$fuggveny_tarolo` változóhoz a `koszon()` függvény névvel azonos karakterláncot rendeltünk. Ha ezt megtettük, a változót arra használhatjuk, hogy meghívja nekünk a `koszon()` függvényt, csupán zárójeleket kell tennünk a változó neve után.

Miért jó ez? A fenti példában csak még több munkát csináltunk magunknak azzal, hogy a `"koszon"` karakterláncot rendeltük a `$fuggveny_tarolo` nevű változóhoz. A dinamikus függvényhívás akkor igazán hasznos, ha a program folyását bizonyos körülményektől függően változtatni szeretnénk. Például szeretnénk mást és mást csinálni egy URL-kérés paraméterétől függően. Ezt a paramétert feldolgozhatjuk és értékétől függően más és más függvényt hívhatunk meg.

A PHP beépített függvényei még hasznosabbá teszik ezt a lehetőséget. Az `array_walk()` függvény például egy karakterláncot használ arra, hogy a tömb minden elemére meghívja a függvényt. Az `array_walk()` alkalmazására a tizenhatodik órában láthatunk példát.

Változók hatóköre

A függvényben használt változók az adott függvényre nézve helyiek maradnak. Más szavakkal: a változó a függvényen kívülről vagy más függvényekből nem lesz elérhető. Nagyobb projektek esetén ez megóvhat minket attól, hogy véletlenül két függvény felülírja azonos nevű változóik tartalmát.

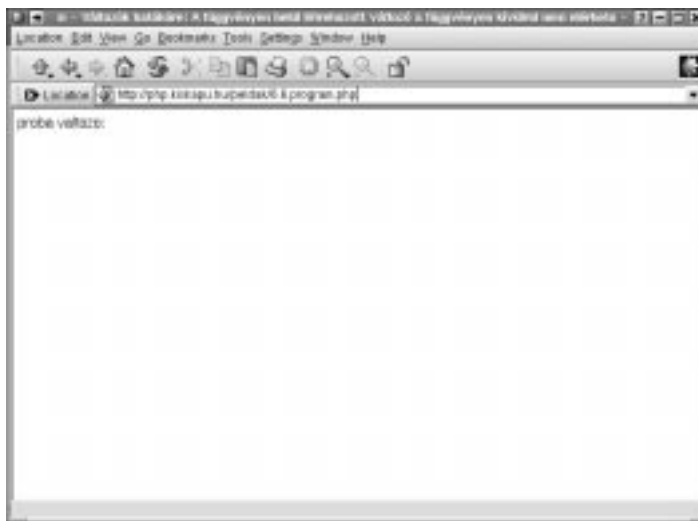
A 6.6. példaprogramban létrehozunk egy változót a függvényen belül, majd megpróbáljuk kiírni a függvényen kívül.

6.6. program Változók hatóköre: A függvényen belül létrehozott változó a függvényen kívülről nem elérhető

```
1: <html>
2: <head>
3: <title>6.6. program Változók hatóköre: A függvényen
  belül létrehozott változó a függvényen kívülről
  nem elérhető</title>
4: </head>
5: <body>
6: <?php
7: function proba()
8:     {
9:         $probavaltozo = "Ez egy proba valtozo";
10:    }
11: print "proba valtozo: $probavaltozo<br>";
12: ?>
13: </body>
14: </html>
```

6.2 ábra

*Kísérlet függvényen
belüli változóra
hivatkozásra.*



A 6.6. példaprogram kimenetét a 6.2. ábrán láthatjuk. A `$probavaltozo` értéke nem íródik ki. Ez annak a következménye, hogy ilyen nevű változó a `proba()` nevű függvényen kívül nem létezik. Figyeljük meg, hogy a nem létező változóra történő hivatkozás nem eredményez hibát.

Hasonlóképp, a függvényen kívül meghatározott változó nem érhető el automatikusan a függvényen belülről.

Hozzáférés változókhoz a global kulcsszó segítségével

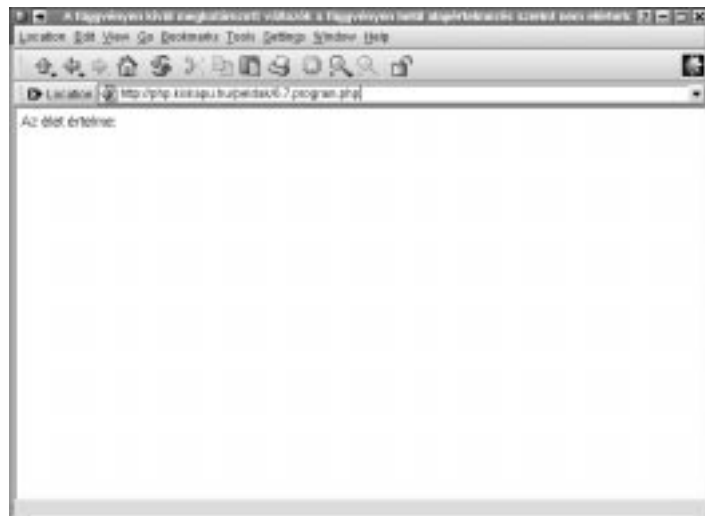
Alapértelmezés szerint a függvényeken belülről nem érhetjük el a máshol meghatározott változókat. Ha mégis megpróbáljuk ezeket használni, akkor helyi változót fogunk létrehozni vagy elérni. Próbáljuk ki ezt a 6.7. példaprogrammal:

6.7. program A függvényen kívül meghatározott változók a függvényen belül alapértelmezés szerint nem elérhetők

```
1: <html>
2: <head>
3: <title>6.7. program A függvényen kívül meghatározott
   változók a függvényen belül alapértelmezés szerint
   nem elérhetők</title>
4:
5: </head>
6: <body>
7: <?php
8: $elet = 42;
9: function eletErtelme()
10: {
11:     print "Az élet értelme: $elet<br>";
12: }
13: eletErtelme();
14: ?>
15: </body>
16: </html>
```

6.3. ábra

*Globális változó
elérési kísérlete
függvényen belülről*



A 6.7. példaprogram kimenetét a 6.3. ábrán láthatjuk. Amint azt vártuk, az `eletErtelme()` függvény nem tudta elérni az `$elet` változót; így az `$elet` a függvényen belül üres. Ezt láthatjuk, amikor a függvény kiírja a változó értékét. Mindent figyelembe véve ez egy jó dolog. Megmenekültünk az azonos nevű változók ütközésétől és a függvény paramétert igényelhet, ha meg szeretne tudni valamit a „külvilág”-ról. Esetenként azonban egy-egy fontos globális (függvényen kívüli) változót anélkül szeretnénk elérni, hogy azt paraméterként át kellene adnunk. Ez az a helyzet, ahol a `global` kulcsszóval létjogosultsága van. A 6.8. példaprogram a `global` kulcsszóval állítja vissza a világ rendjét.

6.8. program Globális változó elérése a global kulcsszó segítségével

```

1: <html>
2: <head>
3: <title>6.8. program Globális változó elérése
   a global kulcsszó segítségével</title>
4: </head>
5: <body>
6: <?php
7: $elet=42;
8: function eletErtelme()
9: {
10:  global $elet;
11:  print "Az élet értelme: $elet<br>";
12: }
13: eletErtelme();
14: ?>
15: </body>
16: </html>

```

6.4. ábra

*Globális változó
függvényből történő
sikeres elérése a global
kulcsszó segítségével*



A 6.8. példaprogram kimenetét a 6.4. ábrán láthatjuk. Miután az `eletErtelme()` függvényben az `$elet` változó elé a `global` kulcsszót tesszük, a függvényen belül a külső, globális `$elet` változót érhetjük el. Minden függvényben, amely globális változót szeretne elérni, használnunk kell a `global` kulcsszót.

Legyünk óvatosak! Ha most az `$elet` nevű változót a függvényen belül megváltoztatjuk, annak a program egészére hatása lesz. A függvénynek átadott paraméter általában valamilyen érték másolata; a paraméter megváltoztatásának a függvény vége után semmilyen hatása nincs. Ha azonban egy globális változó értékét változtatjuk meg egy függvényen belül, akkor az eredeti változót változtattuk meg és nem egy másolatot. Ezért a `global` kulcsszót lehetőleg minél kevesebbszer használjuk.

Állapot megőrzése a függvényhívások között a static kulcsszó segítségével

A függvényen belüli változóknak egészében véve rövid, de boldog életük van. Létrejönnek például akkor, amikor a `szaMoZottCimSor()` függvényt meghívjuk és eltűnnek, amikor a függvény futása befejeződik. Tulajdonképpen ennek így is kell lennie. A lehető legjobb úgy felépíteni egy programot, hogy az független és kis tudású függvényekből álljon. Esetenként azonban jól jönne, ha egy függvénynek lehetne valamilyen kezdetleges memóriája.

Tegyük fel például, hogy tudni szeretnénk, hányszor fut le egy adott függvény. Miért? Példánkban a függvény célja számozott címsor készítése, ami egy dinamikusan létrejövő dokumentációt eredményez.

Itt persze használhatnánk a `global` kulcsszóról újonnan szerzett tudásunkat. Ennek alapján a 6.9. példaprogramhoz hasonlót írhatnánk.

6.9. program Változó értékének függvényhívások közti megőrzése a global kulcsszó segítségével

```
1: <html>
2: <head>
3: <title>6.9. program Változó értékének függvényhívások
   közti megőrzése a global kulcsszó
   segítségével</title>
4: </head>
5: <body>
6: <?php
```

6.9. program (folytatás)

```
7: $fvHivasokSzama = 0;
8: function szamozottCimsor( $cimszoveg )
9:     {
10:         global $fvHivasokSzama;
11:         $fvHivasokSzama++;
12:         print "<h1>$fvHivasokSzama. $cimszoveg</h1>";
13:     }
14: szamozottCimsor("Alkatrészek");
15: print("Az alkatrészek széles skáláját gyártjuk<p>");
16: szamozottCimsor("Műtűrök");
17: print("A legjobbak a világon<p>");
18: ?>
19: </body>
20: </html>
```

6.5. ábra

A függvényhívások számának nyomon követése a global kulcsszó használatával



Ez úgy működik, ahogy azt vártuk. Létrehoztuk a `$fvHivasokSzama` nevű változót a `szamozottCimsor()` függvényen kívül. A változót a függvény számára a `global` kulcsszó segítségével tettük elérhetővé. A 6.9. példaprogram kimenetét a 6.5. ábrán láthatjuk.

Valahányszor meghívjuk az `szamozottCimsor()` nevű függvényt, a `$fvHivasokSzama` változó értéke eggyel nő, majd a megnövelt értékkel a címsort teljessé tesszük.

Ez nem igazán elegáns megoldás. A `global` kulcsszót használó függvények nem függetlenek, nem tekinthetők önálló programrésznek. Ha a kódot olvassuk vagy újrahazsnosítjuk, figyelniünk kell az érintett globális változókra. Ez az a pont, ahol a `static` kulcsszó hasznos lehet. Ha egy függvényen belül egy változót

a `static` kulcsszóval hozunk létre, a változó az adott függvényre nézve helyi marad. Másrészt a függvény hívásról hívásra „emlékszik” a változó értékére. A 6.10. példaprogram a 6.9. példaprogram `static` kulcsszót használó változata.

6.10. program Függvényhívások közötti állapot megőrzése a `static` kulcsszó használatával

```
1: <html>
2: <head>
3: <title>6.10. program Függvényhívások közötti állapot
   megőrzése a static kulcsszó
   használatával</title>
4: </head>
5: <body>
6: <?php
7: function szamozottCimsor( $cimszoveg )
8:     {
9:         static $fvHivasokSzama = 0;
10:        $fvHivasokSzama++;
11:        print "<h1>$fvHivasokSzama. $cimszoveg</h1>";
12:    }
13: szamozottCimsor("Alkatrészek");
14: print("Az alkatrészek széles skáláját gyártjuk<p>");
15: szamozottCimsor("Mütyűrök");
16: print("A legjobbak a világon<p>");
17: ?>
18: </body>
19: </html>
```

A `szamozottCimsor()` függvény így teljesen független lett. Amikor a függvényt először hívjuk meg, a `$fvHivasokSzama` nevű változó kezdeti értéket is kap. Ezt a hozzárendelést a függvény további meghívásainál figyelmen kívül hagyja a PHP, ehelyett az előző futtatáskor megőrzött értéket kapjuk vissza.

Így a `szamozottCimsor()` függvényt már beilleszthetjük más programokba és nem kell aggódnunk a globális változók miatt. Bár a 6.10. példaprogram kimenete teljesen megegyezik a 6.9. példaprograméval, a programot elegánsabbá és hordozhatóbbá tettük. Gondoljunk bele például, mi történne akkor, ha a függvény 6.9. példaprogrambeli változatát egy olyan programba illesztenénk bele, amelyben van egy `$fvHivasokSzama` nevű változó.

Paraméterek további tulajdonságai

Már láttuk, hogyan kell függvényeknek paramétereket átadni, de van még néhány hasznos dolog, amiről még nem beszéltünk. Ebben a részben megtanuljuk, miként lehet a függvényparamétereknek alapértelmezett értéket adni és megtanuljuk azt is, hogyan kell változóinkra mutató hivatkozást átadni a változó értékének másolata helyett.

Paraméterek alapértelmezett értéke

A PHP nyelv remek lehetőséget biztosít számunkra rugalmas függvények kialakításához. Eddig azt mondtuk, hogy a függvények többsége paramétert igényel. Néhány paramétert elhagyhatóvá téve függvényeink rugalmasabbá válnak.

A 6.11. példaprogramban létrehozunk egy hasznos kis függvényt, amely egy karakterláncot egy HTML font elembe zár. A függvény használójának megadjuk a lehetőséget, hogy megváltoztathassa a font elem size tulajdonságát, ezért a karakterlánc mellé kell egy \$meret nevű paraméter is.

6.11. program Két paramétert igénylő függvény

```
1: <html>
2: <head>
3: <title>6.11. program Két paramétert igénylő
   függvény</title>
4: </head>
5: <body>
6: <?php
7: function meretez( $szoveg, $meret )
8: {
9:     print "<font size=\"$meret\"
   face=\"Helvetica,Arial,Sans-Serif\">$szoveg</font>";
10: }
11: meretez("Egy címsor<br>",5);
12: meretez("szöveg<br>",3);
13: meretez("újabb szöveg<BR>",3);
14: meretez("még több szöveg<BR>",3);
15: ?>
16: </body>
17: </html>
```

6.6. ábra

Függvény, amely formázott karakterláncokat ír ki a böngészőbe



A 6.11. példaprogram kimenetét a 6.6. ábrán láthatjuk. Függvényünk ugyan hasznos, de valójában a \$meret nevű paraméter majdnem mindig 3. Ha kezdőértéket rendelünk ehhez a paraméterhez, akkor az elhagyható lesz. Így ha a függvényhívásban nem adunk értéket ennek a paraméternek, akkor az általunk meghatározott értéket fogja felvenni. A 6.12. példaprogram ennek a módszernek a segítségével teszi a \$meret paramétert elhagyhatóvá.

6.12. program Függvény elhagyható paraméterrel

```

1: <html>
2: <head>
3: <title>6.12. program Függvény elhagyható
   paraméterrel</title>
4: </head>
5: <body>
6: <?php
7: function meretez( $szoveg, $meret=3 )
8:     {
9:         print "<font size=\"\$meret\" face=\"Helvetica,
              Arial,Sans-Serif\">$szoveg</font>";
10:    }
11: meretez("Egy címsor<br>",5);
12: meretez("szöveg<br>");
13: meretez("újabb szöveg<BR>");
14: meretez("még több szöveg<BR>");
15: ?>
16: </body>
17: </html>

```


Ha a `meretez()` függvénynek van második paramétere, akkor a `$meret` nevű változót ez fogja meghatározni. Ha a függvényhíváskor nem adunk meg második paramétert, akkor a függvény a 3 értéket feltételezi. Annyi elhagyható paramétert adhatunk meg, ahányat csak akarunk, de arra vigyáznunk kell, hogy az elhagyható paramétereknek a paraméterlista végén kell szerepelniük.



Ezt a korlátozást kikerülhetjük, sőt még a paraméterek sorrendjét sem kell fejben tartanunk, ha a paramétereket egyetlen asszociatív tömbben adjuk át. (Erről bővebben a hetedik, tömbökről szóló fejezetben lesz szó).

Hivatkozás típusú paraméterek

Amikor a függvényeknek paramétereket adunk át, a helyi paraméter-változókba a paraméterek értékének másolata kerül. Az e változókban végzett műveleteknek a függvényhívás befejeződése után nincs hatásuk az átadott paraméterekre. Ennek igazolására futtassuk le a 6.13. példaprogramot.

6.13. program Függvényparaméter érték szerinti átadása

```
1: <html>
2: <head>
3: <title>6.13. program Függvényparaméter érték szerinti
   átadása</title>
4: </head>
5: <body>
6: <?php
7: function ottelTobb( $szam )
8:     {
9:         $szam += 5;
10:    }
11: $regiSzam = 10;
12: ottelTobb( $regiSzam );
13: print( $regiSzam );
14: ?>
15: </body>
16: </html>
```

Az `ottelTobb()` függvény egyetlen számot vár, majd ötöt ad hozzá. Visszatérési értéke nincs. A főprogramban a `$regiSzam` nevű változónak értéket adunk, majd ezzel a változóval meghívjuk az `ottelTobb()` függvényt. A `$regiSzam` változó

értékének másolata kerül a `$szam` nevű változóba. Amikor kiíratjuk a függvényhívás után a `$regiSzam` változó értékét, azt találjuk, hogy az érték még mindig 10. Alapértelmezés szerint a függvények paraméterei érték szerint adódnak át. Más szavakkal, a változók értékéről helyi másolat készül.

Lehetőségünk van arra is, hogy ne a változó értékét, hanem arra mutató hivatkozást adjunk át. (Ezt cím szerinti paraméterátadásnak is hívják.) Ez azt jelenti, hogy a változóra mutató hivatkozással dolgozunk a függvényben és nem a változó értékének másolatával. A paraméteren végzett bármilyen művelet megváltoztatja az eredeti változó értékét is. Hivatkozásokat függvényeknek úgy adhatunk át, hogy az átadandó változó vagy a paraméter neve elé egy `&` jelet teszünk. A 6.14. és a 6.15. példaprogram az előző problémára mutatja be a két előbb említett módszert.

6.14. program Cím szerinti paraméterátadás változóra mutató hivatkozás segítségével

```
1: <html>
2: <head>
3: <title>6.14. program Cím szerinti paraméterátadás
   változóra mutató hivatkozás segítségével</title>
4: </head>
5: <body>
6: <?php
7: function ottelTobb( $szam )
8: {
9:     $szam += 5;
10: }
11: $regiSzam = 10;
12: ottelTobb( &$regiSzam );
13: print( $regiSzam );
14: ?>
15: </body>
16: </html>
```

6.15. program Cím szerinti paraméterátadás a függvénydeklaráció módosításával

```
1: <html>
2: <head>
3: <title>6.15. program Cím szerinti paraméterátadás
   a függvénydeklaráció módosításával</title>
4: </head>
5: <body>
6: <?php
7: function ottelTobb( &$szam )
8:     {
9:         $szam += 5;
10:    }
11: $regiSzam = 10;
12: ottelTobb( $regiSzam );
13: print( $regiSzam );
14: ?>
15: </body>
16: </html>
```

Ha az átadandó paramétert alakítjuk hivatkozássá, akkor nem fogjuk elfelejteni, hogy az átadott paraméter értéke megváltozhat, hiszen mi tettük hivatkozássá. Ennek a változatnak viszont az a veszélye, hogy elfelejtjük kitenni az & jelet. (C programozók már biztosan tapasztalták...) Ha a második megoldást választjuk, akkor lehet, hogy elfelejtjük, hogy a paraméter értéke megváltozhat, viszont nem felejtjük el kitenni az & jelet, mivel nem is kell kitennünk. Mindent egybevéve talán több értelme van annak, hogy a függvénydeklarációban írunk & jelet a paraméter neve elé. Így biztosak lehetünk benne, hogy az egyes függvényhívások azonos módon viselkednek.

Összefoglalás

Ebben az órában megtanultuk, mik azok a függvények és hogyan kell őket használni. Megtanultuk, hogyan kell létrehozni és paramétereket átadni nekik. Elsajátítottuk a `global` és a `static` kulcsszavak használatát. Megtanultuk, hogyan kell a függvényeknek hivatkozásokat átadni és a paramétereknek alapértelmezett értéket adni.

Kérdések és válaszok

A global kulcsszón kívül van más mód arra, hogy egy függvény hozzá tudjon férni vagy módosítani tudjon egy globális változót?

A globális változókat a programon belül bárhol elérhetjük a `$GLOBALS` nevű asszociatív tömb segítségével. A `$proba` nevű globális változót például `$GLOBALS["proba"]`-ként érhetjük el. Az asszociatív tömbökről a következő órában tanulunk.

A globális változót a függvényen belül akkor is módosíthatjuk, ha a függvény paraméterként egy arra mutató hivatkozást kapott.

Kérdés: Lehet-e a függvényhívásokat a változókhoz hasonlóan karakterláncba ágyazni?

Nem. A függvényhívásoknak az idézőjeleken kívül kell lenniük.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Igaz-e, hogy ha egy függvénynek nem kell paramétert átadni, akkor a függvény neve után nem szükséges kitenni a zárójelpárt?
2. Hogyan lehet függvényből értéket visszaadni?
3. Mit ír ki az alábbi kódrészlet?
`$szam = 50;`

```
function tizszer()  
{  
    $szam = $szam * 10;  
}
```

```
tizszer();  
print $szam;
```

4. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer()
{
    global $szam;
    $szam = $szam * 10;
}

tizszer();
print $szam;
```

5. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer($sz)
{
    $sz = $sz * 10;
}

tizszer($szam);
print $szam;
```

6. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer(&$sz)
{
    $sz = $sz * 10;
}

$tizszer($szam);
print $szam;
```

Feladatok

1. Írjunk függvényt, amely négy karakterlánc típusú értéket fogad és olyan karakterlánccal tér vissza, amely paramétereit HTML cellaelemekbe (TD) zárja!



7. ÓRA

Tömbök

A tömbök és a kezelésüket segítő eszközök nagy mértékben növelik a PHP 4 programok rugalmasságát. Ha jól értünk a tömbökhöz, képesek vagyunk nagy méretű, összetett adatokat tárolni és kezelni.

Ebben az órában bemutatjuk a tömböket és néhány beépített függvényt, amelyek segítik a velük való boldogulást. Az órában a következőket tanuljuk meg:

- Mik a tömbök és hogyan hozhatók létre?
- Hogyan érhetjük el a tömbökben tárolt adatokat?
- Hogyan rendezhetjük a tömbökben tárolt adatokat?

Mit nevezünk tömbnek?

Mint már tudjuk, a változó egy „vödör”, amiben egy értéket lehet tárolni. Változók segítségével olyan programot írhatunk, amely információt tárol, dolgoz fel és ír ki. Egy változóban azonban sajnos csak egy értéket tárolhatunk. A tömb olyan különleges szerkezetű változó, amelyben nincs ilyen korlátozás. Egy tömbbe annyi adatot lehet beletenni, amennyit csak akarunk (amennyi memóriánk van). Minden elemet egy szám vagy egy karakterlánc segítségével azonosíthatunk. Ha a változó „vödör”, akkor a tömb „iratszekrény”, tehát olyan tároló, amelyben sok-sok elemet tárolhatunk.

ÚJDONSÁG

A tömb változók listája. Több változót tartalmaz, amelyeket számok vagy karakterláncok segítségével azonosíthatunk, így a különböző értékeket egyetlen névvel tárolhatjuk, rendezhetjük és érhetjük el.

Természetesen ha öt értéket kell tárolnunk, megadhatunk öt változót is. Akkor miért jó tömböt használni változók helyett? Először is azért, mert a tömb rugalmasabb adatszerkezet. Lehet benne két vagy akár kétszáz érték és ennek elérése érdekében nem kell további változókat létrehozni. Másodszor, a tömbök elemeit könnyedén kezelhetjük egységként is. Ha akarjuk, végighaladhatunk a tömbön egy ciklussal vagy elérhetjük az elemeit egyenként. Lehetőségünk van a tömböt rendezni szám szerint, szótári rendezés szerint vagy saját rendezőelv alapján.

A tömb elemeit az index segítségével könnyen elérhetjük. Az index lehet szám, de akár karakterlánc is.

Alapértelmezés szerint a tömböket számokkal indexeljük, mégpedig úgy, hogy az első elem indexe 0. Ebből az következik, hogy az utolsó tömbelem indexe mindig eggyel kisebb a tömb méreténél. Tehát az öt elemű tömb utolsó eleme a 4-es indexű elem. Ezt ajánlatos mindig fejben tartani!

A 7.1. táblázatban a felhasználók tömböt láthatjuk. Figyeljük meg, hogy például a tömb harmadik elemének indexe 2.

7.1. táblázat A felhasználók tömb elemei

<i>Index</i>	<i>Érték</i>	<i>Hányadik elem</i>
0	Berci	Első
1	Mariska	Második
2	Aladár	Harmadik
3	Eleonóra	Negyedik

A karakterlánccal való indexelés akkor lehet hasznos, ha egyedi nevek (kulcsok) mellett más értékeket is tárolni kell.

A PHP 4 mind a számokkal, mind a „nevekkel” indexelt tömbök elérésére biztosít segédeszközöket. Ezek közül néhányat ebben a fejezetben is találkozni fogunk, másokat a tizenhatodik, adatszervezővel foglalkozó fejezetben ismerünk majd meg.

Tömbök létrehozása

A tömbök alapértelmezés szerint értékek számmal indexelt listái. Értéket egy tömbhöz kétféleképpen is rendelhetünk: Az egyik mód az `array()` függvény, a másik a tömbazonosító használata szögletes zárójelekkel `[]`. A következő két részben mind a kétféle változattal találkozni fogunk.

Tömbök létrehozása az `array()` függvény segítségével

Az `array()` függvény akkor hasznos, ha egyszerre több értéket szeretnénk egy tömbhöz rendelni. Hozzunk létre egy `$felhasznalok` nevű tömböt és rendeljünk hozzá négy elemet!

```
$felhasznalok = array ("Berci", "Mariska", "Aladár", "Eleonóra");
```

Most a `$felhasznalok` tömb harmadik elemét, melynek indexe 2, írassuk ki!

```
print $felhasznalok[2];
```

Ez a következő karakterláncot fogja kiírni: "Aladár". Az indexet a tömb neve után közvetlenül következő szögletes zárójelbe kell írni. A tömbelem írásakor és olvasásakor is ezt a jelölést kell használni.

Ne felejtjük el, hogy a tömbök indexelése nullától indul, így bármely tömbelem indexe eggyel kisebb a tömbelem tömbbéli helyénél. (Az első elem indexe 0, a második elem indexe 1.)

Tömb létrehozása vagy elem hozzáadása a tömbhöz szögletes zárójel segítségével

Tömböket úgy is létrehozhatunk, illetve meglevő tömbökhöz új elemeket adhatunk, ha a tömb neve után olyan szögletes zárójelpárt írunk, amelynek belsejében nincs index.

Hozzuk létre újra a `$felhasznalok` nevű tömböt:

```
$felhasznalok[] = "Berci";  
$felhasznalok[] = "Mariska";  
$felhasznalok[] = "Aladár";  
$felhasznalok[] = "Eleonóra";
```

Figyeljük meg, hogy nem kellett számot írunk a szögletes zárójelbe. A PHP 4 automatikusan meghatározza az indexértéket, így nem kell nekünk bajlódni azzal, hogy kiszámítsuk a következő olyan indexet, amelyben még nincs érték.

Persze írhattunk volna számokat is a szögletes zárójelbe, de ez nem tanácsos. Nézzük a következő programrészletet:

```
$felhasznalok[0] = "Berci";  
$felhasznalok[200] = "Mariska";
```

A tömbnek így mindössze két eleme van, de az utolsó elem indexe 200. A PHP 4 nem fog értéket adni a köztes elemeknek, ami félreértésekre adhat okot az elemek elérésekor.

Tömbök létrehozása mellett a tömbváltozók szögletes zárójele segítségével az `array()` függvénnyel létrehozott tömb végéhez új elemet is adhatunk. Az alábbi kis programrészletben létrehozunk egy tömböt az `array()` függvény segítségével, majd új elemet adunk a tömbhöz szögletes zárójellel.

```
$felhasznalok = array ("Berci", "Mariska", "Aladár", "Eleonóra");  
$felhasznalok[] = "Anna";
```

Asszociatív tömbök

A számmal indexelt tömbök akkor hasznosak, ha abban a sorrendben szeretnénk tárolni az elemeket, amilyen sorrendben a tömbbe kerültek. Néha azonban jó lenne, ha a tömb elemeit meg tudnánk nevezni. Az asszociatív tömb egy karakterláncokkal indexelt tömb. Képzeljünk el egy telefonkönyvet: melyik a jobb megoldás: a név mezőt a 4-gyel vagy a "név"-vel indexelni?

ÚJDONSÁG

A karakterláncsal indexelt tömböket asszociatív tömböknek (néha hash-nek) hívják.

Asszociatív tömbök létrehozása a számokkal indexelt tömbökkel megegyezően történik, az `array()` függvény vagy a szógletes zárójelek segítségével.



A számmal és a karakterlánccal indexelt tömbök közti határvonal a PHP-ben nem éles. Nem különböző típusok, mint a Perlben, ennek ellenére helyes az a hozzáállás, ha külön-külön kezeljük őket, mert az egyes típusok más hozzáférési és kezelési módot igényelnek.

Asszociatív tömbök létrehozása az `array()` függvény segítségével

Ha asszociatív tömböt szeretnénk létrehozni az `array()` függvény segítségével, minden elemnek meg kell adni a kulcsát és az értékét. Az alábbi programrészlet egy `$karakter` nevű asszociatív tömböt hoz létre négy elemmel.

```
$karakter = array(
    "nev" => "János",
    "tevekenyseg" => "szuperhős",
    "eletkor" => 30,
    "kulonleges kepesseg" => "röntgenszem"
);
```

Most elérhetjük a `$karakter` elemeit (mezőit):

```
print $karakter["eletkor"];
```

Asszociatív tömbök létrehozása és elérése közvetlen értékadással

Asszociatív tömböt úgy is létrehozhatunk vagy új név–érték párt adhatunk hozzá, ha egyszerűen a megnevezett elemhez (mezőhöz) új értéket adunk. Az alábbiakban újra létrehozzuk a `$karakter` nevű tömböt, úgy, hogy az egyes kulcsokhoz egyenként rendelünk értékeket.

```
$karakter["nev"] => "János";  
$karakter["tevekenyseg"] => "szuperhős";  
$karakter["eletkor"] => 30;  
$karakter["kulonleges kepesseg"] => "röntgenszem";
```

Többsdimenziós tömbök

Eddig azt mondtuk, hogy a tömbök elemei értékek. A `$karakter` tömbünkben az elemek közül három karakterláncot tartalmaz, egy pedig egy egész számot. A valóság ennél egy kicsit bonyolultabb. Egy tömbelem valójában lehet érték, objektum vagy akár egy másik tömb is. A többsdimenziós tömb valójában tömbök tömbje. Képzeljük el, hogy van egy tömbünk, amelynek tömbök az elemei. Ha el akarjuk érni a második elem harmadik elemét, két indexet kell használnunk:

```
$tomb[1][2]
```

ÚJDONSÁG

A tömböt, amelynek elemei tömbök, többsdimenziós tömbnek hívjuk.

A tény, hogy egy tömbelem lehet tömb is, lehetőséget biztosít arra, hogy kifinomultabb adatszerkezeteket kezeljünk viszonylag egyszerűen. A 7.1. példaprogram egy tömböt ír le, amelynek elemei asszociatív tömbök.

7.1. program Többsdimenziós tömb létrehozása

```
1: <html>  
2: <head>  
3: <title>7.1. program Többsdimenziós tömb  
   létrehozása</title>  
4: </head>  
5: <body>  
6: <?php  
7: $karakter = array  
8: (  
9:   array (  
10:      "nev" => "János",  
11:      "tevekenyseg" => "szuperhős",  
12:      "eletkor" => 30,  
13:      "kulonleges kepesseg" => "röntgenszem"  
14:   ),
```

7.1. program (folytatás)

```
15:         array (
16:             "nev" => "Szilvia",
17:             "tevekenyseg" => "szuperhős",
18:             "eletkor" => 24,
19:             "kulonleges kepesseg" => "nagyon erős"
20:         ),
21:         array (
22:             "nev" => "Mari",
23:             "tevekenyseg" => "főgonosz",
24:             "eletkor" => 63,
25:             "kulonleges kepesseg" =>
                "nanotechnológia"
26:         )
27:     );
28:
29: print $karakter[0]["tevekenyseg"]; //kiírja, hogy
    szuperhős
30: ?>
31: </body>
32: </html>
```

Figyeljük meg, hogy `array()` függvényhívásokat építettünk egy `array()` függvényhívásba. Az első szinten adjuk meg a tömböt, melynek minden eleme asszociatív tömb.

A `$karakter[2]` tömbelemhez történő hozzáféréskor a legfelső szintű tömb harmadik elemét, egy asszociatív tömböt kapunk. Az asszociatív tömb bármely elemét elérhetjük, például a `$karakter[2]["nev"]` értéke "Mari" lesz, a `$karakter[2]["eletkor"]` pedig 63.

Ha a fogalmak tiszták, asszociatív és hagyományos tömbök párosításával egyszerűen hozhatunk létre összetett adatszerkezeteket.

Tömbök elérése

Eddig azt láttuk, hogyan kell tömböket létrehozni és elemeket adni azokhoz. Most végignézzünk néhány lehetőséget, amit a PHP 4 a tömbök elérésére biztosít.

Tömb méretének lekérdezése

A tömb bármely elemét elérhetjük indexének használatával:

```
print $felhasznalo[4];
```

A tömbök rugalmassága miatt nem mindig egyszerű feladat kideríteni, hány elem van a tömbben. A PHP a `count()` függvényt biztosítja erre a feladatra.

A `count()` a tömbben levő elemek számával tér vissza. Az alábbi programrészben egy számokkal indexelt tömböt hozunk létre, majd a `count()` függvényt használjuk a tömb utolsó elemének elérésére.

```
$felhasznalok = array ("Berci", "Marci", "Ödön", "Télapó");  
$felhasznalok[count($felhasznalok)-1];
```

Figyeljük meg, hogy egyet ki kellett vonnunk a `count()` által visszaadott értékből. Ez azért van így, mert a `count()` nem az utolsó elem indexét adja vissza, hanem a tömbelemek számát.

Semmi sem garantálja, hogy ezzel a módszerrel bármilyen (számokkal indexelt) tömb utolsó elemét megkapjuk. Vegyük például az alábbi programot:

```
$tomb = array("Ecc, pecc"); // Egyelemű tömb!  
$tomb[2] = "kimehetsz";  
print "Az utolsó elem: " . $tomb[count($tomb)-1];
```

A fenti kódot futtatva azt tapasztaljuk, hogy az "Az utolsó elem: " szöveg után semmi nem íródik ki. Ez azért van, mert a tömbnek nincs 1 indexű eleme. Az első elem a 0 indexen található, a második a 2 indexen, mivel az értékadáskor az indexet közvetlenül határoztuk meg.

A tömbök indexelése alapértelmezés szerint nullától indul, de ezt meg lehet változtatni. Ha azonban világos és következetes programokat szeretnénk írni, ne éljünk ezzel a lehetőséggel.

Tömb bejárása

Több módja van annak, hogy egy tömb minden elemét végigjárjuk. Mi most a PHP 4 igen hatékony `foreach` szerkezetét használjuk, de a tizenhatodik órában további módszereket is megvizsgálunk.



A foreach a PHP 4-es változatában került a nyelvbe.

A számmal indexelt tömbökre a foreach szerkezetet így kell használni:

```
foreach($tombnev as $atmeneti)
{
}
```

\$tombnev a tömb neve, amit végig szeretnénk járni, az \$atmeneti pedig egy változó, amelybe minden ismétlés során a tömb egy-egy eleme kerül.

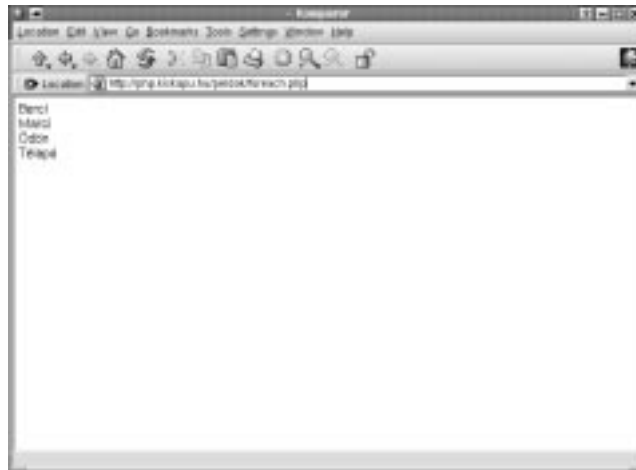
Az alábbi példában egy számmal indexelt tömböt hozunk létre és a foreach vezérlési szerkezet segítségével érjük el annak elemeit:

```
$felhasznalok = array ("Berci", "Marci", "Ödön");
$felhasznalok[10] = "Télapó";
foreach($felhasznalok as $szemely)
{
    print"$szemely<br>";
}
```

A program kimenetét a 7.1-es ábrán láthatjuk.

7.1. ábra

Tömb bejárása



A tömb minden eleme átmenetileg a \$szemely változóba került, amit a program kiír a böngészőbe. A Perlben gyakorlatilag rendelkezők vigyázzanak, a foreach szerkezet a két nyelvben jelentősen különbözik! A Perlben ha az átmeneti változó értékét megváltoztatjuk, a megfelelő tömbelem is megváltozik.

Ha ugyanezt az előző példában tesszük, a `$felhasznalok` tömb értéke nem fog megváltozni. A tizenhatodik órában látni fogjuk, hogyan módosíthatjuk a `foreach` segítségével a tömbök tartalmát.

Figyeljük meg, hogy annak ellenére, hogy a tömb „lyukas”, a program a `foreach` szerkezet segítségével bejárt tömb minden elemét kiírta és az utolsó sor előtt nem jelentek meg üres sorok, tehát a 3 és 9 indexek közötti tömbelemeket (melyek nem meghatározottak, vagyis üres karakterláncok) nem írja ki a program.



Ahol csak mód van rá, ne használjunk tömbindexeket. A tömböket sokszor csak egyszerű listaként használjuk, ahol nem számít az, hogy egy adott elemnek mi az indexe. A PHP a tömbök elérésére a tömbindexek használatánál hatékonyabb és áttekinthetőbb függvényeket biztosít, programunk ezáltal rugalmasabb, könnyebben módosítható és jól olvasható lesz.

Asszociatív tömb bejárása

Ha az asszociatív tömb kulcsát és értékét is el szeretnénk érni, egy kicsit másképpen kell a `foreach` szerkezet használnunk.

Asszociatív tömböknél a `foreach` így alkalmazható:

```
foreach( $tomb as $kulcs => $ertek )  
{  
    // a tömbelem feldolgozása  
}
```

ahol `$tomb` a tömb neve, amin végig szeretnénk menni, `$kulcs` a változó, amelyben az elem kulcsa jelenik meg, a `$ertek` pedig a kulcshoz tartozó tömb-elem értéke.

A 7.2. példaprogramban létrehozunk egy asszociatív tömböt, majd egyesével elérjük és kiírjuk a tömb elemeit

7.2. program Asszociatív tömb bejárása a foreach segítségével

```
1: <html>
2: <head>
3: <title>7.2 példaprogram Asszociatív tömb bejárása
  a foreach segítségével</title>
4: </head>
5: <body>
6: <?php
7: $karakter = array (
8:     "nev" => "János",
9:     "tevekenyseg" => "szuperhős",
10:    "eletkor" => 30,
11:    "kulonleges kepesseg" => "röntgenszem"
12: );
13:
14: foreach ( $karakter as $kulcs => $ertek )
15: {
16:     print "$kulcs = $ertek<br>";
17: }
18: ?>
19: </body>
20: </html>
```

A 7.2. példaprogram kimenetét a 7.2. ábrán láthatjuk.

7.2. ábra

*Asszociatív tömb
bejárása*



Többszemes tömb bejárása

Most már ismerünk olyan módszereket, amelyek segítségével a 7.1. példaprogramban létrehozott többszemes tömböt bejárhatjuk. A 7.3. példaprogramban létrehozunk egy többszemes tömböt és a `foreach` segítségével végigjárjuk az elemeit.

7.3. program Többszemes tömb bejárása

```
1: <html>
2: <head>
3: <title>7.3. program - Többszemes tömb
   bejárása</title>
4: </head>
5: <body>
6: <?php
7: $karakterek = array
8:     (
9:         array (
10:             "nev" => "János",
11:             "tevekenyseg" => "szuperhős",
12:             "eletkor" => 30,
13:             "kulonleges kepesseg" => "röntgenszem"
14:         ),
15:         array (
16:             "nev" => "Szilvia",
17:             "tevekenyseg" => "szuperhős",
18:             "eletkor" => 24,
19:             "kulonleges kepesseg" => "nagyon erős"
20:         ),
21:         array (
22:             "nev" => "Mari",
23:             "tevekenyseg" => "főgonosz",
24:             "eletkor" => 63,
25:             "kulonleges kepesseg" =>
               "nanotechnológia"
26:         )
27:     );
```

7.3. program (folytatás)

```
28: foreach ( $karakterek as $karakter )
29:     {
30:         foreach ( $karakter as $kulcs => $ertek )
31:             {
32:                 print "$kulcs: $ertek<br>";
33:             }
34:         print "<br>";
35:     }
36: ?>
37: </body>
38: </html>
```

A 7.3. példaprogram kimenetét a 7.3. ábrán láthatjuk. Két `foreach` ciklust használunk. A külső ciklusban a számmal indexelt `$karakterek` tömb elemeit vesszük végig, az egyes elemek pedig a `$karakter` nevű változóba kerülnek. A `$karakter` maga is egy tömb, ezért ennek értékeit is egy `foreach` ciklussal járjuk végig. Mivel a `$karakter` egy asszociatív tömb, a `foreach` szerkezetet az ennek megfelelő formában használjuk, a kulcs-érték párok pedig a `$kulcs` és az `$ertek` változókba kerülnek.

Annak érdekében, hogy ez a módszer jól működjön, biztosnak kell lennünk abban, hogy a `$karakter` változó valóban mindig tömböt tartalmaz. A kódot megbízhatóbbá tehetnénk, ha meghívnánk a `$karakter` változóra az `is_array()` függvényt. Az `is_array()` függvény számára egy paramétert kell megadnunk. A visszatérési érték `true` lesz, ha a változó típusa tömb, minden más esetben `false` értéket ad.



Bonyolultabb programok esetében gyakran előfordul, hogy kíváncsiak vagyunk egy tömb vagy valamilyen más összetett változó tartalmára. Ilyenkor általában nem kell saját tömblistázó kódot használnunk, hiszen a PHP 4 biztosítja számunkra a `print_r()` függvényt, amely a paramétereként megadott változó tartalmát írja ki.

Műveletek tömbökkel

Most már fel tudunk tölteni tömböket elemekkel, elérhetjük azokat, de a PHP rengeteg függvénnyel segíti a tömbök feldolgozását is. A Perlben jártas olvasó bizonyára ismerősnek fogja találni ezen függvények nagy részét.

Két tömb egyesítése az `array_merge()` függvény segítségével

Az `array_merge()` függvény legalább két paramétert vár: az egyesítendő tömböket. A visszatérési érték az egyesített tömb lesz. Az alábbi példában létrehozunk két tömböt, egyesítjük azokat, majd végigjárjuk a keletkező harmadik tömböt:

```
$első = array( "a", "b", "c" );
$masodik = array( 1, 2, 3 );
$harmadik = array_merge( $első, $masodik );
foreach( $harmadik as $érték )
{
    print "$érték<br>";
}
```

A `$harmadik` tömb az `$első` és a `$masodik` tömbök elemeinek másolatát tartalmazza. A `foreach` ciklus ennek a tömbnek az elemeit ("a", "b", "c", 1, 2, 3) írja ki a böngészőbe. Az egyes elemeket a `
` (sörtörés) választja el egymástól.



Az `array_merge()` függvény a PHP 4-es változatában került a nyelvbe.

Egyszerre több elem hozzáadása egy tömbhöz az `array_push()` függvény segítségével

Az `array_push()` függvény egy tömböt és tetszőleges számú további paramétert fogad. A megadott értékek a tömbbe kerülnek. Figyeljük meg, hogy az `array_merge()` függvénnyel ellentétben az `array_push()` megváltoztatja első paraméterének értékét. E függvény visszatérési értéke a tömbben lévő elemek száma (miután az elemeket hozzáadta). Hozzunk létre egy tömböt, majd adjunk hozzá néhány elemet:

```
$elso = array( "a", "b", "c" );
$elemszam = array_push( $elso, 1, 2, 3 );

print "Összesen $elemszam elem van az \$elso tömbben<P>";
foreach( $elso as $ertek )
{
    print "$ertek<br>";
}
```

Mivel az `array_push()` függvény visszaadja a módosított tömb elemeinek számát, az értéket (ami ebben az esetben 6) egy változóban tárolhatjuk és kiíratjuk a böngészőbe. Az `$elso` tömb ekkor az eredeti három betűt tartalmazza, melyekkel az első sorban feltöltöttük, illetve a három számot, amit az `array_push()` függvény segítségével adtunk hozzá. Az így létrehozott tömb elemeit a `foreach` ciklus segítségével kiíratjuk a böngészőbe.

Figyeljük meg, hogy a `"$elso"` karakterlánc elé egy fordított perjelet kellett tennünk. Ha a PHP egy macskakörmös (kettős idézőjeles) karakterláncban egy dollárjelet követő betű- és/vagy számsorozatot talál, változóként próbálja értelmezni azt és megkísérli az értékét beírni. A fenti példában mi a `'$elso'` karakter sorozatot szeretnénk volna megjeleníteni, ezért kellett a dollárjel elé a fordított perjel karakter. Így a PHP már nem próbálja meg változóként értelmezni, hanem kiírja a karaktereket. Ezt a módszert a szakirodalomban gyakran `escape`-ként emlegetik.



A Perlben gyakorlattal rendelkezők figyeljenek arra, hogy ha a PHP-ben az `array_push()` második paramétereként tömböt adnak meg, a tömb egy tömb típusú elemmel bővül, ezáltal többdimenziós tömb keletkezik. Ha két tömb elemeit szeretnénk egyesíteni, használjuk az `array_merge()` függvényt.



Ha egy karakterláncban nem szeretnénk, hogy a változók behelyettesítődjenek, a karakterlánc jelölésére egyszeres idézőjeleket használjunk; ekkor a `'` és a `\` karakterek helyett `'\'`-t és `\\`-t kell írunk.

Az első elem eltávolítása az `array_shift()` függvény segítségével

Az `array_shift()` eltávolítja a paraméterként átadott tömb első elemét és az elem értékével tér vissza. A következő példában az `array_shift()` függvény segítségével eltávolítjuk a tömb első elemét. Ezt a műveletet egy `while` ciklusba tesszük és a műveletet addig ismételjük, amíg el nem fogy a tömb. Annak ellenőrzésére, hogy van-e még elem a tömbben, a `count()` függvényt használjuk.

```
<?php
$egy_tomb = array( "a", "b", "c" );

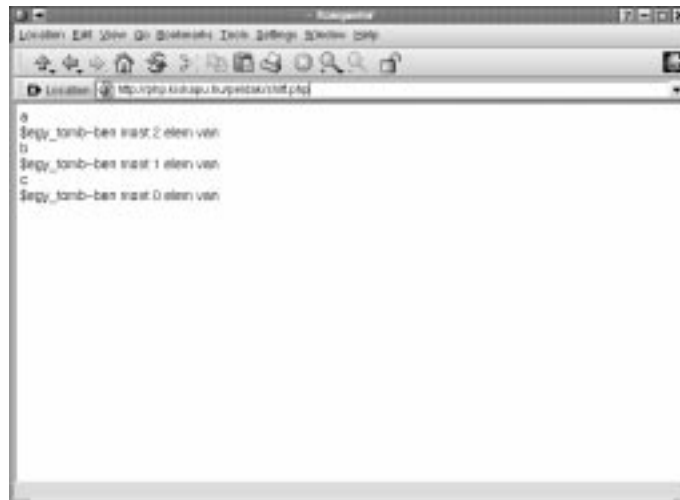
while ( count( $egy_tomb ) )
{
    $ertekek = array_shift( $egy_tomb );
    print "$ertekek<br>";
    print '$egy_tomb-ben most ' . count( $egy_tomb )
        . ' elem van<br>';
}

?>
```

A program kimenetét a 7.4. ábrán láthatjuk.

7.4. ábra

*A tömb egyes elemeinek törlése és kiírata-
sa az `array_shift()`
függvény segítségével*



Az `array_shift()` akkor lehet hasznos, amikor egy sor elemein kell valamilyen tevékenységet végezni, amíg a sor ki nem ürül.



Az `array_shift()` függvény a PHP 4-es változatában került a nyelvbe.

Tömb részének kinyerése az `array_slice()` függvény segítségével

Az `array_slice()` függvény egy tömb egy darabját adja vissza. A függvény egy tömböt, egy kezdőpozíciót (a szelet első elemének tömbbeli indexét) és egy (elhagyható) hossz paramétert vár. Ha ez utóbbit elhagyjuk, a függvény feltételezi, hogy a kezdőpozíciótól a tömb végéig tartó szeletet szeretnénk megkapni. Az `array_slice()` nem változtatja meg a paraméterként átadott tömböt, hanem újat ad vissza, amelyben a hívó által kért elemek vannak.

Az alábbi példában létrehozunk egy tömböt, majd egy új, három elemű tömböt állítunk elő belőle:

```
$első = array( "a", "b", "c", "d", "e", "f" );
$masodik = array_slice( $első, 2, 3 );

foreach( $masodik as $ertek )
{
    print "$ertek<br>";
}
```

A fenti példa a "c", "d" és "e" elemeket írja ki, a `
` kódelemmel elválasztva. Mivel kezdőpozícióként a kettőt adtuk meg, így az első elem, ami a `$masodik` tömbbe kerül, az `$első[2]`.

Ha kezdőpozícióként negatív számot adunk meg, a tömbszelet első eleme hátulról a megadott számú elemtől kezdődik. Ez azt jelenti, hogy ha a második paraméter értéke -1, a tömbszelet az utolsó elemtől fog kezdődni.

Ha hossz paraméterként nullánál kisebb számot adunk meg, a visszakapott tömb-szelet a tömb hátulról a megadott számú eleméig tart.



Az `array_slice()` függvény a PHP 4-es változatában jelent meg.

Tömbök rendezése

Talán a rendezés a legnagyszerűbb művelet, amit egy tömbön el lehet végezni. A PHP 4 függvényeinek köszönhetően egykettőre rendet teremthetünk a káoszban. A következő rész néhány olyan függvényt mutat be, amelyek számmal és karakterlánccal indexelt tömböket rendeznek.

Számmal indexelt tömb rendezése a sort() függvény segítségével

A `sort()` függvény egy tömb típusú paramétert vár és a tömb rendezését végzi el. Ha a tömbben van karakterlánc, a rendezés (alapbeállításban angol!) ábécésorrend szerinti lesz, ha a tömbben minden elem szám, szám szerint történik. A függvénynek nincs visszatérési értéke, a paraméterként kapott tömböt alakítja át. Ebből a szempontból különbözik a Perl hasonló függvényétől. Az alábbi programrészlet egy karakterekből álló tömböt hoz létre, rendezi, majd a rendezett tömb elemeit egyenként kiírja:

```
$tomb = array( "x", "a", "f", "c" );
sort( $tomb );

foreach ( $tomb as $elem )
{
    print "$elem<br>";
}
```

Nézzük egy kis példát egy tömb kétféle rendezési módjára!

```
$tomb = array( 10, 2, 9 );
sort( $tomb );

print '___Rendezés szám szerint___<br>';
foreach ( $tomb as $elem )
{
    print "$elem<br>";
}
$tomb[]="a";
sort( $tomb );

print '___Rendezés ábécésorrend szerint___<br>';
foreach ( $tomb as $elem )
{
    print "$elem<br>";
}
```

A fenti példa kimenete:

```
___Rendezés szám szerint___
2
9
10
```


___Rendezés ábécésorrend szerint___

10

2

9

a



A `sort()` függvényt ne használjuk asszociatív (karakterlánccal indexelt) tömbökre! Ha mégis meg tesszük, azt fogjuk tapasztalni, hogy a tömb elemei ugyan rendezettek lesznek, de az elemek kulcsai elvesznek, az egyes elemek így nullától kezdődő számokkal érhetők el, ugyanis a tömb számmal indexelt tömbbé alakul át.

Ha csökkenő sorrendben szeretnénk rendezni egy tömb elemeit, használjuk a `sort()` függvényhez hasonlóan működő `rsort()` függvényt.

Asszociatív tömb rendezése érték szerint az `asort()` függvény segítségével

Az `asort()` függvény egy asszociatív tömb típusú paramétert vár és a tömböt a `sort()` függvényhez hasonlóan rendezi. Az egyetlen különbség, hogy az `asort()` használata után megmaradnak a karakterlánc-kulcsok:

```
$a_tomb = array( "első"=>5, "második"=>2, "harmadik"=>1 );

asort( $a_tomb );
foreach( $a_tomb as $kulcs => $ertekek )
{
    print "$kulcs = $ertekek<br>";
}
```

Ennek a programocskának a kimenetét a 7.5. ábrán láthatjuk.

7.5. ábra

*Asszociatív tömb
érték szerinti ren-
dezése az asort() függ-
vény segítségével*



Ha csökkenő sorrendben szeretnénk rendezni egy asszociatív tömb elemeit, használjuk a `arsort()` függvényt.

Asszociatív tömb rendezése kulcs szerint a `ksort()` függvény segítségével

A `ksort()` a paraméterben megadott asszociatív tömböt rendezi kulcs szerint. A többi tömbrendező függvényhez hasonlóan ez a függvény sem ad vissza semmi-féle visszatérési értéket, hanem a tömböt alakítja át:

```
$tomb = array( "x" => 5, "a" => 2, "f" => 1 );

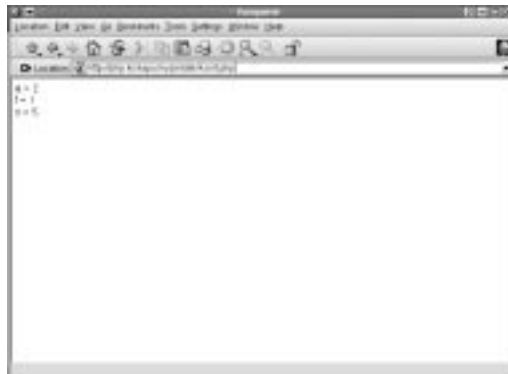
ksort( $tomb );

foreach( $tomb as $kulcs => $ertek )
{
    print "$kulcs = $ertek<BR>";
}
```

A program kimenetét a 7.6. ábrán láthatjuk.

7.6. ábra

*Asszociatív tömb
rendezése kulcs
szerint a ksort() függ-
vény segítségével*



Összefoglalás

Ebben az órában a tömbökről és a hozzájuk kapcsolódó eszközökről tanultunk, melyeket a PHP 4 a rajtuk végzett műveletekhez nyújt. Most már tudunk normál (számmal indexelt) és asszociatív (karakterlánccal indexelt) tömböket is létrehozni, illetve tömböket bejárni a `foreach` ciklus segítségével.

Többdimenziós tömböket is létre tudunk hozni, a bennük levő információt pedig ki tudjuk nyerni egymásba ágyazott `foreach` ciklusokkal. A tömbökhöz egyszerre több elemet is adhatunk, illetve onnan kivehetünk. Végül megtanultuk, hogy a PHP 4-es változata milyen lehetőségeket nyújt a tömbök rendezésére.

Kérdések és válaszok

Ha a `foreach` ciklus csak a PHP 4-esben került a nyelvbe, akkor a PHP 3-as változatát használó programozók hogyan jártak végig egy tömböt?

A PHP 3-asban az `each()` függvényt használták egy `while()` ciklusban. Erről bővebben a tizenhatodik órában olvashatunk.

Van olyan tömböt kezelő függvény, amellyel nem foglalkoztunk ebben az órában?

A PHP 4-es változatában nagyon sok tömbkezelő függvény van. Erről bővebben a tizenhatodik órában fogunk tanulni. A kézikönyv erről szóló része megtalálható az Interneten, a <http://www.php.net/manual/en/ref.array.php> címen. (A kézikönyv magyar változata a <http://hu.php.net/manual/hu/> oldalon található.)

Ha a tömb elemeinek számát ismerem, normál tömb bejárására a `for` ciklust használjam? (Ekkor számláló segítségével címezem meg a tömb elemeit.)

Nagyon óvatosan kell bánni ezzel a megközelítéssel. Nem lehetünk biztosak abban, hogy az általunk használt tömb indexei 0-tól indulnak és egyesével nőnek.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvénnyel hozhatunk létre tömböket?
2. Az alább létrehozott tömb utolsó elemének mi az indexe?

```
$torpek = array( "Tudor", "Vidor", "Kuka" );
```

3. Függvények használata nélkül hogyan tudnánk "Hapci"-t az imént létrehozott \$torpek nevű tömbhöz adni?
4. Melyik függvény segítségével tudnánk "Morgó"-t a \$torpek nevű tömbbe felvenni?
5. Hogyan lehet egy tömb elemszámát megtudni?
6. A PHP 4-es változatában mi a legegyszerűbb módja egy tömb bejárásának?
7. Milyen függvény segítségével lehet két tömböt egyesíteni?
8. Hogyan rendeznénk egy asszociatív tömböt kulcsai szerint?

Feladatok

1. Hozzunk létre egy többdimenziós tömböt, amely filmeket tartalmaz, zsáner szerint rendezve! Pontosabban hozzunk létre egy asszociatív tömböt, amelynek kulcsai filmtípusok ("Sci fi", "Akció", "Romantikus"...)! A tömb elemei legyenek tömbök, amelyek filmcímeket tartalmaznak ("2001", "Alien", "Terminator",...)!
2. Járjuk végig az előző feladatban létrehozott tömböt és írjuk ki az összes filmtípust és a hozzá tartozó filmcímeket!



8. ÓRA

Objektumok

Az objektumközpontú programozás veszélyes. Megváltoztatja a programozásról való gondolkodásmódunkat és ha egyszer a hatalmába kerít, nem tudunk tőle szabadulni. A PHP a Perlhöz hasonlóan fokozatosan építette be az objektumközpontúságot a nyelvtanba. A PHP 4-es változatának megjelenésével képessé váltunk arra, hogy programjainkat teljes mértékben objektumközpontú szemléletmódban írjuk.

Ebben az órában a PHP 4-es változatának objektumközpontú tulajdonságaival foglalkozunk és azokat valós életből vett példákra alkalmazzuk. Az órában a következőket tanuljuk meg:

- Mik az osztályok és objektumok?
- Hogyan lehet osztályokat és objektumpéldányokat létrehozni?
- Hogyan lehet tagfüggvényeket és tulajdonságokat létrehozni és elérni?
- Hogyan lehet olyan osztályt létrehozni, mely más osztály leszármazottja?
- Miért jó az objektumközpontú programozás és hogyan segíti munkánk szervezését?

Mit nevezünk objektumnak?

Az objektum változók és függvények egybezárt csomagja, amely egy különleges sablonból jön létre. Az objektum belső működése nagy részét elrejtí az objektumot használó elől, úgy, hogy egyszerű hozzáférési felületet biztosít, melyen keresztül kéréseket küldhetünk és információt kaphatunk. A felület különleges függvényekből, úgynevezett tagfüggvényekből (metódusokból) áll. A tagfüggvények mindegyike hozzáférhet bizonyos változókhoz, amelyeket tulajdonságoknak nevezünk.

A típus jellegzetességeit objektumtípusok (osztályok, `class`) létrehozásával határozzuk meg. Objektumpéldányok (vagy röviden objektumok) létrehozásával pedig egyedeket hozunk létre. Ezen egyedek rendelkeznek a típusuknak megfelelő jellemzőkkel, de természetesen a jellemzők értéke más és más lehet. Például ha létrehozunk egy `gepkocsi` osztályt, akkor az osztálynak lehet egy jellemzője, amelynek a `szin` nevet adjuk. Az objektumpéldányokban a `szin` tulajdonság értéke lehet például `"kék"` vagy `"zöld"`.

Az osztály tagfüggvények és tulajdonságok együttese. Az osztályokat a `class` kulcsszó segítségével hozhatjuk létre. Az osztályok sablonok, amelyekből objektumokat állíthatunk elő.

ÚJDONSÁG

Az objektum az osztály egy példánya, vagyis egy objektum nem más, mint az osztályban rögzített működési szabályok megtestesülése.

Egy adott típusú objektum a `new` kulcsszó után írt objektumtípus nevének segítségével testesíthető meg. Amikor egy objektumpéldány létrejön, összes tulajdonsága és tagfüggvénye elérhetővé válik.

Az objektumközpontú program legnagyobb előnye valószínűleg a kód újrahasznosíthatósága. Mivel az osztályokat egységbezárt objektumok létrehozására használjuk, egyszerűen vihetők át egyik projektből a másikba, ráadásul lehetőség van arra is, hogy gyermekosztályokat hozzunk létre, amelyek a szülő tulajdonságait öröklik és képesek azt kiegészíteni vagy módosítani. E módszer segítségével rengeteg összetett vagy egyedi tulajdonsággal rendelkező objektumot, vagyis objektumcsaládot lehet létrehozni, amelyek egy alaposztályra épülnek, de közben egyéni tulajdonságokkal is rendelkeznek, egyedi tagfüggvényeket is biztosítanak.

Az objektumközpontú programozás bemutatásának talán legjobb módja, ha a bemutatott példák alapján kísérletezgetünk.

Objektum létrehozása

Ahhoz, hogy létre tudjunk hozni egy objektumot, először létre kell hozni a sablont, amelyre épül. Ezt a sablont hívjuk osztálynak. A PHP 4-es változatában ilyen sablont a `class` kulcsszóval hozhatunk létre:

```
class elso_osztaly
{
    // ez egy nagyon buta osztály
}
```

Az `elso_osztaly` a minta, amely alapján tetszőleges számú `elso_osztaly` típusú objektumot hozhatunk létre. Objektumot létrehozni a `new` kulcsszó segítségével lehet:

```
$obj1 = new elso_osztaly();
$obj2 = new elso_osztaly();
print "\$obj1 " . gettype($obj1) . " típusú<br>";
print "\$obj2 " . gettype($obj2) . " típusú<br>";
```

A PHP `gettype()` függvényével ellenőriztük, hogy az `$obj1` és az `$obj2` változóknak valóban objektum van-e. A `gettype()` számára egy tetszőleges típusú változót kell átadnunk, a függvény pedig egy karakterlánccal tér vissza, ami a változó típusát tartalmazza. Az előző programrészletben a `gettype()` visszatérési értéke `"object"`, amit a `print` segítségével a böngésző számára kiírtunk.

Meggyőződhattünk róla, hogy két objektumunk van. A példának első ránézésre túl sok hasznát még nem látjuk, de segítségével újabb nézőpontból vizsgálhatjuk az osztályokat. Az osztályokra úgy tekinthetünk, mint öntőmintára: annyi objektumot önthetünk a segítségével, amennyit csak szeretnénk. Adjunk néhány további szolgáltatást az osztályhoz, hogy objektumaink kicsit érdekesebbek legyenek.

Objektumtulajdonságok

Az objektumok a tulajdonság néven említett különleges változók révén működnek. Ezek az osztályban bárhol létrehozhatók, de az átláthatóság kedvéért az osztálymeghatározás elejére célszerű gyűjteni azokat, így mi is ezt tesszük. A tulajdonságok lehetnek értékek, tömbök, de más objektumok is:

```
class elso_osztaly
{
    var $nev = "Gizike";
}
```

Figyeljük meg, hogy a változót a `var` kulcsszó segítségével hoztuk létre; ha ezt egy osztályon belül nem írjuk ki, értelmezési hibát (parse error) kapunk. Most, hogy ezt a változót létrehoztuk, minden `elso_osztaly` típusú objektumnak lesz egy `nev` tulajdonsága, melynek "Gizike" lesz a kezdeti értéke. Ehhez a tulajdonsághoz az objektumon kívülről is hozzá lehet férni, sőt meg is lehet változtatni:

```
class elso_osztaly
{
    var $nev = "Gizike";
}
$obj1 = new elso_osztaly();
$obj2 = new elso_osztaly();
$obj1->nev = "Bonifác";
print "$obj1->nev<br>";
print "$obj1->nev<br>";
```

A `->` művelettel teszi lehetővé, hogy egy objektum tulajdonságait elérhessük vagy módosíthassuk. Bár az `$obj1` és az `$obj2` objektumokat "Gizike" nével hoztuk létre, rábírtuk az `$obj2` objektumot, hogy meggondolja magát, a `nev` tulajdonsághoz a "Bonifác" értéket rendelve, mielőtt a `->` jelet ismét használva kiírtuk volna mindkét objektum `nev` tulajdonságának értékét.



Az objektumközpontú nyelvek, mint a Java, megkívánják, hogy a programozó beállítsa a tulajdonságok és tagfüggvények biztonsági szintjét. Ez azt jelenti, hogy a hozzáférés úgy korlátozható, hogy csak bizonyos, az objektumot kezelő magas szintű függvények legyenek elérhetők, a belső használatra szánt tulajdonságok, tagfüggvények pedig biztonságosan elrejtethetők. A PHP-nek nincs ilyen védelmi rendszere. Az objektum összes tulajdonsága elérhető, ami problémákat okozhat, ha a tulajdonságot (kívülről) nem lenne szabad megváltoztatni.

Az objektumot adattárolásra is használhatjuk, de az objektumok többre képesek, mint egy egyszerű asszociatív tömb utánzása. A következő részben áttekintjük az objektumok tagfüggvényeit, amelyek segítségével objektumaink életre kelnek.

Az objektumok tagfüggvényei

A tagfüggvény olyan függvény, amelyet egy osztályon belül hozunk létre. Minden objektumpéldány képes meghívni osztálya tagfüggvényeit. A 8.1. példaprogram az `elso_osztaly` nevű osztályt egy tagfüggvénnyel egészíti ki.

8.1. program Egy osztály tagfüggvénnyel

```
1: <html>
2: <head>
3: <title>8.1. program Egy osztály
   tagfüggvénnyel</title>
4: <body>
5: <?php
6: class elso_osztaly
7:     {
8:         var $nev;
9:         function koszon()
10:            {
11:                print "Üdvözlöm!";
12:            }
13:     }
14: $obj1 = new elso_osztaly();
15: $obj1->koszon(); // kiírja, hogy "Üdvözlöm!"
16: ?>
17: </body>
18: </html>
```

Amint látjuk, a tagfüggvény a szokványos függvényekhez nagyon hasonlóan viselkedik. A különbség csak annyi, hogy a tagfüggvényt mindig osztályon belül hozzuk létre. Az objektumok tagfüggvényeit a `->` művelettel segítségével hívhatjuk meg. Fontos, hogy a tagfüggvények hozzáférnek az objektum változóihoz. Már láttuk, hogyan lehet egy objektumtulajdonságot az objektumon kívülről elérni, de hogyan hivatkozhat egy objektum saját magára? Lássuk a 8.2. példa-programot:

8.2. program Tulajdonság elérése tagfüggvényből

```
1: <html>
2: <head>
3: <title>8.2. program Tulajdonság elérése
   tagfüggvényből</title>
4: <body>
5: <?php
6: class elso_osztaly
7:     {
8:         var $nev = "Krisztián";
9:         function koszon()
```

8.2. program (folytatás)

```
10:         {
11:             print "Üdvözlöm! $this->nev vagyok.<BR>";
12:         }
13:     }
14:
15: $obj1 = new elso_osztaly();
16: $obj1->koszon(); // kiírja, hogy "Üdvözlöm! Krisztián
                    vagyok."
17: ?>
18: </body>
19: </html>
```

Az osztálynak van egy különleges változója, a `$this`, ami az objektumpéldányra mutat. Tekinthetjük személyes névmásnak. Bár programunkban az objektumra hivatkozhatunk azzal a változóval, amihez hozzárendeltük (például `$obj1`), az objektumnak hivatkoznia kell saját magára, erre jó a `$this` változó. A `$this` változó és a `->` műveletjel segítségével az osztályon belülről az osztály minden tulajdonságát és tagfüggvényét elérhetjük.

Képzeljük el, hogy minden egyes `elso_osztaly` típusú objektumhoz más és más nev tulajdonságot szeretnénk rendelni. Ezt megtehetnénk „kézzel”, az objektum nev tulajdonságát megváltoztatva, de létrehozhatunk egy tagfüggvényt is, ami ezt teszi. Az utóbbi megoldást láthatjuk a 8.3. példaprogramban.

8.3. program Tulajdonság értékének módosítása tagfüggvényen belülről

```
1: <html>
2: <head>
3: <title>8.3. program Tulajdonság értékének módosítása
   tagfüggvényen belülről</title>
4: </head>
5: <body>
6: <?php
7: class elso_osztaly
8:     {
9:         var $nev = "Krisztián";
10:        function atkeresztel( $n )
11:        {
12:            $this->nev = $n;
13:        }
```

8.3. program (folytatás)

```
14:         function koszon()  
15:         {  
16:             print "Üdvözlöm! $this->nev vagyok.<BR>";  
17:         }  
18:     }  
19:  
20: $obj1 = new elso_osztaly();  
21: $obj1->atkeresztel("Aladár");  
22: $obj1->koszon(); // kiírja, hogy "Üdvözlöm! Aladár  
                        vagyok."  
23: ?>  
24: </body>  
25: </html>
```

Az objektum nev tulajdonsága a létrehozáskor még "Krisztián" volt, de aztán meghívtuk az objektum `atkeresztel()` tagfüggvényét, ami "Aladár" értékre változtatta azt. Láthatjuk, hogy az objektum képes saját tulajdonságait módosítani. Figyeljük meg, hogy a tagfüggvénynek a hagyományos függvényekhez hasonlóan adhatunk át paramétereket.

Még mindig van egy fogás, amit nem ismertünk meg. Ha egy metódusnak éppen azt a nevet adjuk, mint az osztálynak (az előző példában ez az `elso_osztaly`), akkor a metódus automatikusan meghívódik, amikor egy új objektumpéldány létrejön. E módszer segítségével már születésükkor testreszabhatjuk objektumainkat. A létrejövő példány az ezen függvénynek átadott paraméterek vagy más tényezők alapján hozhatja magát alapállapotba. Ezeket a különleges metódusokat konstruktoroknak (létrehozó függvényeknek) hívjuk. A 8.4. példaprogramban az `elso_osztaly` objektumtípus konstruktorral kiegészített változatát láthatjuk.

8.4. program Konstruktorral rendelkező osztály

```
1: <html>  
2: <head>  
3: <title>8.4. program Konstruktorral rendelkező  
   osztály</title>  
4: </head>  
5: <body>  
6: <?php
```

8.4. program (folytatás)

```
7: class elso_osztaly
8:     {
9:         var $nev;
10:        function elso_osztaly( $n="Anonymous" )
11:            {
12:                $this->nev = $n;
13:            }
14:        function koszon()
15:            {
16:                print"Üdvözlöm! $this->nev vagyok.<BR>";
17:            }
18:    }
19:
20: $obj1 = new elso_osztaly("Krisztián");
21: $obj2 = new elso_osztaly("Aladár");
22: $obj1->koszon(); // kiírja, hogy "Üdvözlöm! Krisztián
                vagyok."
23: $obj2->koszon(); // kiírja, hogy "Üdvözlöm! Aladár
                vagyok."
24: ?>
25: </body>
26: </html>
```

Amikor egy `elso_osztaly` típusú objektumot létrehozunk, az `elso_osztaly()` konstruktor automatikusan meghívásra kerül. Ha az objektum létrehozásakor nem adunk meg nevet, a paraméter értéke "anonymus" lesz.

Egy bonyolultabb példa

Most készítsünk együtt egy kicsit bonyolultabb és hasznosabb programot: egy táblázat osztályt hozunk létre, amelyben az egyes oszlopokat nevük alapján is elérhetjük. Az adatszerkezet soralapú lesz, a böngészőben való megjelenítést pedig egy butácska függvényre bízuk. A táblázat takaros formázása a probléma megoldásának ebben a fázisában nem szükséges.

Az osztály tulajdonságai

Először is el kell döntenünk, hogy milyen tulajdonságokat tároljunk az objektumban. Az oszlopok neveit egy tömbben tároljuk, a sorokat pedig egy kétdimenziós tömbben. A táblázat oszlopainak számát külön, egy egész típusú változóba helyezzük.

```
class Tablázat
{
    var $tablázatSorok = array();
    var $oszlopNevek = array();
    var $oszlopszam;
}
```

A konstruktor

A táblázat létrehozásakor a programnak már tudnia kell, mik lesznek a feldolgozandó oszlopok nevei. Ez az információ az objektumhoz a konstruktor egy karakterlánc típusú tömbparamétere segítségével fog eljutni. Ha az oszlopok neveit a konstruktor már ismeri, könnyűszerrel meghatározhatja az oszlopok számát és beírhatja az \$oszlopszam tulajdonságba:

```
function Tablázat( $oszlopNevek )
{
    $this->oszlopNevek = $oszlopNevek;
    $this->oszlopszam = count ( $oszlopNevek );
}
```

Ha feltesszük, hogy a konstruktor számára helyes információt adtunk át, az objektum már tudni fogja oszlopainak számát és nevét. Mivel ez az információ tulajdonságokban (mezőkben) tárolódik, minden tagfüggvény számára hozzáférhető.

Az ujSor() tagfüggvény

A Tablázat osztály a sorokat tömbök formájában kapja meg. Ez persze csak akkor működik helyesen, ha a tömbben és a fejlécben az oszlopok sorrendje azonos:

```
function ujSor( $sor )
{
    if ( count ($sor) != $this->oszlopszam )
        return false;
    array_push($this->tablázatSorok, $sor);
    return true;
}
```

Az ujSor() tagfüggvény egy tömböt vár, amit a \$sor nevű változóban kap meg. A táblázat oszlopainak számát már az \$oszlopszam tulajdonságba helyeztük, így most a count() függvénnyel ellenőrizhetjük, hogy a kapott \$sor paraméter megfelelő számú oszlopot tartalmaz-e. Ha nem, akkor a függvény false értéket ad vissza.

A PHP 4-es változatának `array_push()` függvénye segítségével az új sort a `$tablazatSor` tömb végéhez fűzhetjük. Az `array_push()` két paramétert vár: a tömböt, amihez az új elemet hozzá kell adni, illetve a hozzáadandó elemet. Ha a második paraméter maga is egy tömb, akkor az egy elemként adódik az első tömbhöz, így többdimenziós tömb jön létre. Ezen eljárás segítségével tehát többdimenziós tömböket is létrehozhatunk.

Az `ujNevesSor()` tagfüggvény

Az `ujNevesSor()` tagfüggvény csak akkor használható kényelmesen, ha a tömbként átadott paraméterben az elemek sorrendje megfelelő. Az `ujNevesSor()` tagfüggvény ennél több szabadságot ad. A metódus egy asszociatív tömböt vár, ahol az egyes elemek kulcsa az oszlopok neve. Ha olyan kulcsú elem kerül a paraméterbe, amely a `Tablázat` objektum `$oszlopNevek` tulajdonságában nem szerepel, az adott elem egyszerűen nem kerül bele a táblázatba. Ha egy oszlopnév nem jelenik meg a paraméterben kulcsként, akkor az adott oszlopba egy üres karakterlánc kerül.

```
function ujNevesSor( $asszoc_sor )
{
    if ( count ( $asszoc_sor ) != $this->oszlopszam )
        return false;
    $sor = array();
    foreach ( $this->oszlopNevek as $oszlopNev )
    {
        if ( ! isset( $asszoc_sor[ $oszlopNev ] ) )
            $asszoc_sor[ $oszlopNev ] = "";
        $sor[] = $asszoc_sor[ $oszlopNev ];
    }
    array_push( $this->tablazatSorok, $sor );
}
```

Az `ujNevesSor()`-nak átadott asszociatív tömb az `$asszoc_sor` nevű változóba kerül. Először létrehozunk egy üres `$sor` tömböt, amely majd a táblázatba beillesztendő sort tartalmazza, az elemek helyes sorrendjével. Majd végigvesszük az `$oszlopNevek` tömb elemeit, és megnézzük, hogy a tömbben kaptunk-e ilyen kulcsú elemet az `$asszoc_sor` paraméterben. Ehhez a PHP 4-es `isset()` függvényét használjuk, amely egy változót vár. Ha a változóhoz már rendeltünk értéket, a függvény visszatérési értéke `true`, egyébként `false` lesz. A függvénynek az `$asszoc_sor` tömb azon elemét kell átadnunk, melynek kulcsa megegyezik a következő oszlop nevével. Ha nem létezik ilyen elem az `$asszoc_sor` tömbben, létrehozunk egy ilyet és üres karakterláncot írunk bele. Most már folytathatjuk a végleges `$sor` tömb felépítését és hozzáadhatjuk az `$asszoc_sor` megfelelő elemét, hiszen épp az imént biztosítottuk, hogy az elem létezik. Mire befejezzük

az `$oszlopNevek` tömb bejárását, a `$sor` tömb már tartalmazni fogja az `$asszoc_sor` tömbben átadott elemeket, mégpedig a megfelelő sorrendben, a nem meghatározott értékeket üres karakterláncokkal kitöltve.

Most már van két tagfüggvényünk, amelyek segítségével sorokat adhatunk a `Tablázat` típusú objektumok `$tablazatSorok` nevű mezőjéhez. Erről azonban jó lenne valahogy meggyőződni: hiányzik még egy olyan tagfüggvény, amely képes megjeleníteni a táblázatot.

A kiir() tagfüggvény

A `kiir()` tagfüggvény kiírja a böngészőbe a táblázat fejlécét és a `$tablazatSorok` tömböt. A függvény csak nyomkövetési célokat szolgál. Az óra későbbi részében egy sokkal szebb megoldást fogunk látni.

```
function kiir()
{
    print "<pre>";
    foreach ( $this->oszlopNevek as $oszlopNev )
        print "<B>$oszlopNev</B> ";
    print "\n";
    foreach ( $this->tablazatSorok as $y )
    {
        foreach ( $y as $xcella )
            print "$xcella ";
        print "\n";
    }
    print "</pre>";
}
```

A program magáért beszél. Először az `$oszlopNevek` elemeit írjuk ki, majd a `$tablazatSorok` sorait. Mivel a `$tablazatSorok` tulajdonság kétdimenziós tömb, vagyis a tömb minden eleme maga is egy tömb, kettős ciklust kell használnunk.

Összeáll a kép

A 8.5. példaprogram eddigi munkánk gyümölcsét, a `Tablázat` osztályt tartalmazza, majd a program végén létrehoz egy `Tablázat` típusú objektumot és meghívja valamennyi tagfüggvényét.

8.5. program A Tablázat osztály

```
1: <html>
2: <head>
3: <title>8.5. program A Tablázat osztály</title>
4: </head>
5: <body>
6: <?php
7: class Tablázat
8:     {
9:         var $tablázatSorok = array();
10:        var $oszlopNevek = array();
11:        var $oszlopszam;
12:        function Tablázat( $oszlopNevek )
13:            {
14:                $this->oszlopNevek = $oszlopNevek;
15:                $this->oszlopszam = count ( $oszlopNevek );
16:            }
17:
18:        function ujSor( $sor )
19:            {
20:                if ( count ($sor) != $this->oszlopszam )
21:                    return false;
22:                array_push($this->tablázatSorok, $sor);
23:                return true;
24:            }
25:
26:        function ujNevesSor( $asszoc_sor )
27:            {
28:                if ( count ($asszoc_sor) != $this->oszlopszam )
29:                    return false;
30:                $sor = array();
31:                foreach ( $this->oszlopNevek as $oszlopNev )
32:                    {
33:                        if ( ! isset( $asszoc_sor[$oszlopNev] ) )
34:                            $asszoc_sor[$oszlopNev] = "";
35:                        $sor[] = $asszoc_sor[$oszlopNev];
36:                    }
37:                array_push($this->tablázatSorok, $sor);
38:            }
39:
```


8.5. program (folytatás)

```

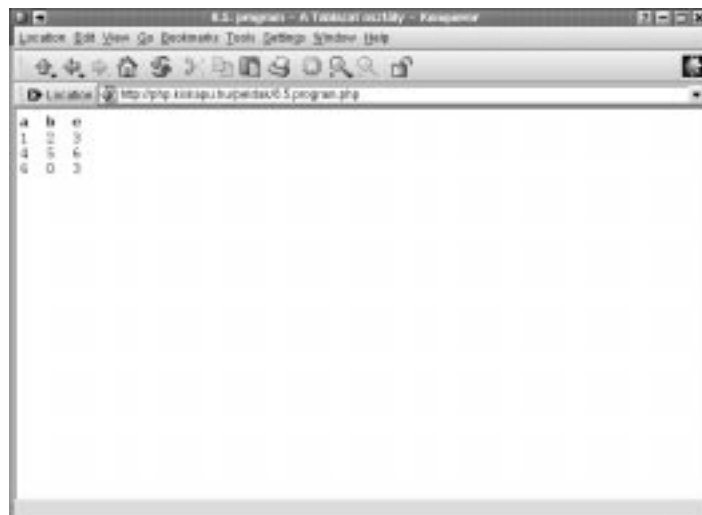
40:     function kiir()
41:     {
42:         print "<pre>";
43:         foreach ( $this->oszlopNevek as $oszlopNev )
44:             print "<B>$oszlopNev</B>  ";
45:         print "\n";
46:         foreach ( $this->tablazatSorok as $y )
47:             {
48:                 foreach ( $y as $xcella )
49:                     print "$xcella  ";
50:                 print "\n";
51:             }
52:         print "</pre>";
53:     }
54: }
55:
56: $proba = new Tablazat( array("a","b","c"));
57: $proba->ujSor( array(1,2,3));
58: $proba->ujSor( array(4,5,6));
59: $proba->ujNevesSor( array ( "b"=>0, "a"=>6, "c"=>3 ));
60: $proba->kiir();
61: ?>
62: </body>
63: </html>

```

A 8.5. példaprogram kimenetét a 8.1. ábrán láthatjuk.

8.1. ábra

Egy Tablazat típusú objektum kimenete



A kimenet csak akkor jelenik meg helyesen, ha az egy oszlopban levő elemek egyforma hosszúak.

Ami még hiányzik...

Bár az osztály hatékonyan elvégzi a munkát, amit neki szántunk, ha több időnk és helyünk lenne, kibővíthetnénk néhány további szolgáltatással és adatvédelmi lehetőséggel.

Mivel a PHP gyengén típusos nyelv, a mi felelősségünk megbizonyosodni arról, hogy a tagfüggvényeknek átadott paraméterek a megfelelő típusúak-e. Ehhez a tizenhatodik, az adatkezelésről szóló fejezetben tárgyalt függvényeket használhatjuk. Ezen kívül írhatnánk tagfüggvényeket a táblázat bármely oszlop szerinti rendezésére is.

Miért használjunk osztályt?

Miért jobb osztályt használni erre a célra ahelyett, hogy egyszerűen írnanék néhány tömbkezelő függvényt? Vagy miért ne írhatnánk bele egyszerűen a programba a tömbkezelést? Azért, mert ez nem lenne hatékony, mert így az elvont adatok tárolását végző programokba valami egészen más is bekerülne. De vajon miért okoz ez nekünk problémát?

Először is a kódot újra fel lehet használni. Ennek célja jól meghatározott: bizonyos általános adatkezelést tesz lehetővé, amelyet ezután beilleszthetünk bármely programba, melynek arra van szüksége, hogy ilyen típusú adatokat kezeljen és kiírjon.

Másodszor, a `Tablázat` osztály tevékeny: megkérhetjük arra, hogy adatokat írjon ki, anélkül, hogy bajlódnunk kellene azzal, hogy végigjárjuk a `$tablázatSorok` elemeit.

Harmadszor, az objektumba egy felületet is beépítettünk. Ha elhatározzuk, hogy később javítunk az osztály működésén, anélkül tehetjük meg, hogy a program többi részét módosítanánk, ha a korábban létrehozott tagfüggvények kívülről ugyanúgy viselkednek.

Végül, létrehozhatunk újabb osztályokat is, amelyek már korábban létrehozott osztályoktól örökölnék, kiterjeszthetik vagy módosíthatják a már meglévő tagfüggvényeket. Ez teszi igazán hatékonná az objektumközpontú programozást, melynek kulcsszavai az egységbezárás, az öröklés és a kód-újrahasznosítás.

Öröklés

Ahhoz, hogy olyan osztályt hozzunk létre, amely szolgáltatásait szülőjétől öröklí, egy kicsit módosítanunk kell az osztály meghatározását. A 8.6. példaprogram ismét a fejezet első felében levő témakörből mutat egy példát.

8.6. program Másik osztálytól öröklő osztály

```
1: <html>
2: <head>
3: <title>8.6. program Másik osztálytól öröklő osztály
  </title>
4: </head>
5: <body>
6: <?php
7: class elso_osztaly
8:     {
9:         var $nev = "Krisztián";
10:        function elso_osztaly( $n )
11:            {
12:                $this->nev = $n;
13:            }
14:        function koszon()
15:            {
16:                print "Üdvözlöm! $this->nev vagyok.<BR>";
17:            }
18:    }
19:
20: class masodik_osztaly extends elso_osztaly
21:     {
22:
23:     }
24:
25: $proba = new masodik_osztaly("Krisztián fia");
26: $proba->koszon(); // kiírja, hogy "Üdvözlöm! Krisztián
                      fia vagyok."
27: ?>
28: </body>
29: </html>
```

Figyeljük meg, hogyan származtattunk az `első_osztaly`-tól egy új osztályt! Az `extends` kulcsszót az osztály meghatározásában kell használnunk. Ezzel azt adjuk meg, hogy a kulcsszó után meghatározott osztály minden tulajdonságát és tagfüggvényét örökölni szeretnénk. Az összes `masodik_osztaly` típusú objektum rendelkezni fog egy `koszon()` nevű tagfüggvénnyel és egy `$nev` nevű tulajdonsággal, mint ahogyan minden `első_osztaly` típusú objektum is rendelkezik ezekkel.

Ha ez nem lenne elég, nézzük tovább a 8.6. példaprogramot és keressünk szokatlan dolgokat! Például figyeljük meg, hogy még konstruktort sem adtunk meg a `masodik_osztaly` osztálynak. Akkor hogy lehet az, hogy az objektum `$nev` tulajdonsága az alapértelmezett "Krisztián" karakterláncról arra a "Krisztián fia" karakterláncra változott, amit a konstruktor létrehozásakor átadtunk? Mivel nem hoztunk létre új konstruktort, a szülő objektumtípus (`első_osztaly`) konstruktora hívódott meg.



Ha egy osztályból egy másikat származtatunk, amelynek nincsen saját konstruktora, akkor a szülő osztály konstruktora hívódik meg, amikor egy gyermekobjektumot hozunk létre. Ez a PHP 4-es változatának újdonása.

A szülő tagfüggvényeinek felülírása

A `masodik_osztaly` típusú objektumok most pontosan úgy viselkednek, mint az `első_osztaly` típusúak. Az objektumközpontú világban a gyermek osztályban felül lehet írni a szülő osztály tagfüggvényeit, ezáltal lehetővé válik, hogy a gyermek objektumok bizonyos tagfüggvényei a szülőkéktől eltérően viselkedjenek (többalakúság, polimorfizmus), más tagfüggvények viszont érintetlenül hagyva ugyanúgy jelenjenek meg a gyermek osztályban. A 8.7. példában lecseréljük a `masodik_osztaly` `koszon()` tagfüggvényét.

8.7. program Egy felülírt tagfüggvény

```
1: <html>
2: <head>
3: <title>8.7. program Egy felülírt tagfüggvény</title>
4: </head>
5: <body>
6: <?php
```

8.7. program (folytatás)

```
7: class elso_osztaly
8:     {
9:         var $nev = "Krisztián";
10:        function elso_osztaly( $n )
11:            {
12:                $this->nev = $n;
13:            }
14:        function koszon()
15:            {
16:                print "Üdvözlöm! $this->nev vagyok.<BR>";
17:            }
18:    }
19:
20: class masodik_osztaly extends elso_osztaly
21:     {
22:        function koszon()
23:            {
24:                print "Azért se mondom meg a nevem!<BR>";
25:            }
26:    }
27:
28: $proba = new masodik_osztaly("Krisztián fia");
29: $proba->koszon(); // kiírja, hogy "Azért se mondom
                    meg a nevem!"
30: ?>
31: </body>
32: </html>
```

A gyermek `koszon()` tagfüggvénye hívódik meg, mert előnyt élvez a szülő hasonló nevű tagfüggvényével szemben.

A felülírt tagfüggvény meghívása

Előfordulhat, hogy a szülő osztály szolgáltatásait szeretnénk használni, de újakat is be szeretnénk építeni. Az objektumközpontú programozásban ez megvalósítható. A 8.8. példaprogramban a `masodik_osztaly koszon()` tagfüggvénye meghívja az `elso_osztaly` tagfüggvényét, amelyet éppen felülír.

8.8. program Felülírt tagfüggvény meghívása

```
1: <html>
2: <head>
3: <title>8.8. program Felülírt tagfüggvény
   meghívása</title>
4: </head>
5: <body>
6: <?php
7: class elso_osztaly
8:     {
9:         var $nev = "Krisztián";
10:        function elso_osztaly( $n )
11:            {
12:                $this->nev = $n;
13:            }
14:        function koszon()
15:            {
16:                print "Üdvözlöm! $this->nev vagyok.<BR>";
17:            }
18:    }
19:
20: class masodik_osztaly extends elso_osztaly
21:     {
22:        function koszon()
23:            {
24:                print "Azért se mondom meg a nevem! - ";
25:                elso_osztaly::koszon();
26:            }
27:    }
28:
29: $proba = new masodik_osztaly("Krisztián fia");
30: $proba->koszon(); // kiírja, hogy "Azért se mondom
   meg a nevem! - Üdvözlöm! Krisztián fia vagyok."
31: ?>
32: </body>
33: </html>
```

A

```
szuloOsztalyNeve::tagfuggvenyNeve()
```

sémát használva bármely felülírt tagfüggvényt meghívhatunk. A séma új, a PHP 3-as változatában még hibát okozott volna.

Egy példa az öröklésre

Már láttuk, hogyan tud egy osztály egy másiktól örökölni, annak tagfüggvényeit felülírni és képességeit kibővíteni. Most a tanultaknak hasznát is vehetjük. Készítsünk egy osztályt, amely a 8.5. példaprogramban szereplő `Tablázat` osztálytól örököl! Az új osztály neve `HTMLTablázat` lesz. Ezt az osztály azért szükséges, hogy kiküszöbölje a `Tablázat` `kiir()` tagfüggvénynek szépséghibáit és a böngésző lehetőségeit kihasználva szép kimenetet eredményezzen.

A HTMLTablázat saját tulajdonságai

A `HTMLTablázat` arra szolgál, hogy a már korábban létrehozott `Tablázat` típusú objektumot szabványos HTML formában jeleníthessük meg. Példánkban a táblázat `cellPadding` és a cellák `bgColor` tulajdonságai módosíthatók, a valós programokban azonban természetesen több tulajdonságot szeretnénk majd beállítani.

```
class HTMLTablázat extends Tablázat
{
    var $hatterSzin;
    var $cellaMargo = 2;
}
```

Egy új osztályt hoztunk létre, amely a `Tablázat` osztálytól fog örökölni. Két új tulajdonságot adtunk meg, az egyiknek a kezdőértékét is meghatároztuk.

A konstruktor

Már láthattuk, hogy a szülő konstruktora hívódik meg, ha a gyermekosztályban nem hozunk létre konstruktort. Most azonban azt szeretnénk, hogy konstruktorunk többre legyen képes, mint amennyit a `Tablázat` osztály konstruktora tett, ezért azt felül kell írunk.

```
function HTMLTablázat( $oszlopNevek, $hatter="#ffffff" )
{
    Tablázat::Tablázat( $oszlopNevek );
    $this->hatterSzin=$hatter;
}
```

A `HTMLTablázat` paramétereiben az oszlopok neveinek tömbjét, illetve egy karakterláncot vár. A karakterlánc lesz a HTML táblázat `bgColor` tulajdonsága. Megadtunk egy kezdeti értéket is, így ha nem adunk át második paramétert a konstruktornak, a háttér színe fehér lesz. A konstruktor meghívja a `Tablázat` osztály konstruktorát a kapott oszlopnév-tömbbel. A lustaság erény a programozás terén: hagyjuk, hogy a `Tablázat` konstruktora tegye a dolgát és a továbbiakban

nem foglalkozunk a táblázat kezelésével (ha egyszer már megírtuk, akkor használjuk is...). A konstruktor másik dolga a `HTMLTablázat` `hatterSzin` tulajdonságának beállítása.



Ha a gyermek osztály rendelkezik konstruktorral, akkor a szülő osztály konstruktora már nem kerül automatikusan meghívásra. Ezért ha szükséges, a gyermek osztályból kell meghívni a szülő konstruktorát.

A `cellaMargoAllit()` tagfüggvény

A származtatott osztály saját, új tagfüggvényeket is létrehozhat.

A `cellaMargoAllit()` tagfüggvény segítségével a felhasználó a táblázat szövege és a táblázat közötti üres rés nagyságát állíthatja be. Persze megtehetné ezt a `cellaMargo` tulajdonság közvetlen elérésével is, de ez nem túl jó ötlet. Alapszabály, hogy az objektumot használó személy érdekében legjobb tagfüggvényeket létrehozni erre a célra. Az osztályok bonyolultabb leszármazottaiban a `cellaMargoAllit()` tagfüggvény esetleg más tulajdonságokat is kénytelen módosítani, hogy megváltoztathassa a margo méretét. Sajnos nincs elegáns mód ennek a kikényszerítésére.

```
function cellaMargoAllit( $margo )
{
    $this->cellaMargo = $margo;
}
```

A `kiir()` tagfüggvény

A `kiir()` tagfüggvény felülírja a `Tablázat` osztály azonos nevű tagfüggvényét. Ez a függvény a szülő osztály logikájával azonos módon írja ki a táblázatot, de a `HTML table` elemét használja a táblázat formázására.

```
function kiir()
{
    print "<table cellPadding=\"{$this->cellaMargo}\"
        border=1>\n";
    print "<tr>\n";
    foreach ( $this->oszlopNevek as $oszlopNev )
        print "<th bgColor=\"{$this
            ->hatterSzin}\">{$oszlopNev}</th>";
    print "</tr>\n";
```



```
foreach ( $this->tablazatSorok as $sor => $cellak )
{
    print "<tr>\n";
    foreach ( $cellak as $cella )
        print "<td bgColor=\"\$this
            ->hatterSzin\">$cella</td>\n";
    print "</tr>\n";
}
print "</table>";
}
```

Ha a Tablázat osztálybeli változat világos, az itteni `kiir()` tagfüggvény is elég áttekinthető kell, hogy legyen. Először az `$oszlopNevek` elemeit írjuk ki, majd a `$tablazatSorok` sorait. A formázást a HTML `table` elemének `cellPadding` és `bgColor` tulajdonságai segítségével szabályozhatjuk.

A Tablázat és a HTMLTablázat osztályok a maguk teljességében

A 8.9. példaprogramban a Tablázat és a HTMLTablázat példákat egy helyen láthatjuk. Létrehozunk egy HTMLTablázat típusú objektumot, megváltoztatjuk `cellaMargo` tulajdonságát, írunk bele néhány sort, majd meghívjuk a `kiir()` tagfüggvényt. A valóságban az adatok minden bizonnyal közvetlenül egy adatbázisból származnának.

8.9. program A Tablázat és a HTMLTablázat osztályok

```
1: <html>
2: <head>
3: <title>8.9. program A Tablázat és a HTMLTablázat
   osztályok</title>
4: </head>
5: <body>
6: <?php
7: class Tablázat
8:     {
9:         var $tablazatSorok = array();
10:        var $oszlopNevek = array();
11:        var $oszlopszam;
```

8.9. program (folytatás)

```
12:     function Tablázat( $oszlopNevek )
13:     {
14:         $this->oszlopNevek = $oszlopNevek;
15:         $this->oszlopszam = count ( $oszlopNevek );
16:     }
17:
18:     function ujSor( $sor )
19:     {
20:         if ( count ($sor) != $this->oszlopszam )
21:             return false;
22:         array_push($this->tablazatSorok, $sor);
23:         return true;
24:     }
25:
26:     function ujNevesSor( $asszoc_sor )
27:     {
28:         if ( count ($asszoc_sor) != $this->oszlopszam )
29:             return false;
30:         $sor = array();
31:         foreach ( $this->oszlopNevek as $oszlopNev )
32:         {
33:             if ( ! isset( $asszoc_sor[$oszlopNev] ) )
34:                 $asszoc_sor[$oszlopNev] = "";
35:             $sor[] = $asszoc_sor[$oszlopNev];
36:         }
37:         array_push($this->tablazatSorok, $sor);
38:     }
39:
40:     function kiir()
41:     {
42:         print "<pre>";
43:         foreach ( $this->oszlopNevek as $oszlopNev )
44:             print "<B>$oszlopNev</B> ";
45:         print "\n";
46:         foreach ( $this->tablazatSorok as $y )
47:         {
48:             foreach ( $y as $xcella )
49:                 print "$xcella ";
50:             print "\n";
51:         }
```

8.9. program (folytatás)

```
52:         print "</pre>";
53:     }
54: }
55:
56: class HTMLTablázat extends Tablázat
57: {
58:     var $hatterSzin;
59:     var $cellaMargo = 2;
60:     function HTMLTablázat( $oszlopNevek,
        $hatter="#ffffff" )
61:     {
62:         Tablázat::Tablázat($oszlopNevek);
63:         $this->hatterSzin=$hatter;
64:     }
65:
66:     function cellaMargoAllit( $margo )
67:     {
68:         $this->cellaMargo = $margo;
69:     }
70:     function kiir()
71:     {
72:         print "<table cellPadding=\""$this->cellaMargo\""
            border=1>\n";
73:         print "<tr>\n";
74:         foreach ( $this->oszlopNevek as $oszlopNev )
75:             print "<th bgColor=\""$this->hatterSzin\""
                >$oszlopNev</th>";
76:         print "</tr>\n";
77:         foreach ( $this->tablázatSorok as $sor => $cellak )
78:             {
79:                 print "<tr>\n";
80:                 foreach ( $cellak as $cella )
81:                     print "<td bgColor=\""$this->hatterSzin\""
                        >$cella</td>\n";
82:                 print "</tr>\n";
83:             }
84:         print "</table>";
85:     }
86: }
87:
```

8.9. program (folytatás)

```

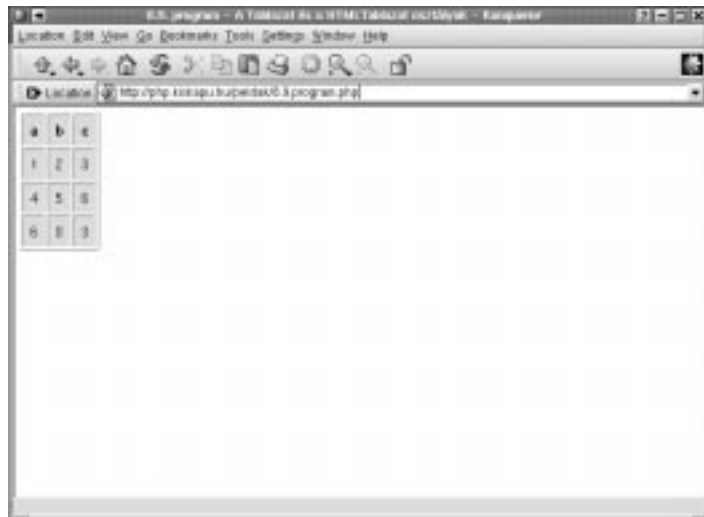
88: $proba = new HTMLTablazat( array("a","b","c"),
    "#00FF00");
89: $proba->cellaMargoAllit( 7 );
90: $proba->ujSor( array(1,2,3));
91: $proba->ujSor( array(4,5,6));
92: $proba->ujNevesSor( array ( "b"=>0, "a"=>6, "c"=>3 ));
93: $proba->kiir();
94: ?>
95: </body>
96: </html>

```

A 8.9. példaprogram kimenetét a 8.2. ábrán láthatjuk.

8.2. ábra

*Egy HTMLTablazat
típusú objektum
kimenete*



Miért alkalmazzunk öröklést?

Miért vágtuk ketté a feladatot egy Tablazat és egy HTMLTablazat részre? Biztos, hogy időt és fáradságot takaríthattunk volna meg, ha a HTML táblázat képességeit beépítjük a Tablazat osztályba? A válasz kulcsa a rugalmasság kérdése.

Képzeld el, hogy egy ügyfél azt a megbízást adja nekünk, hogy hozzunk létre egy osztályt, amely olyan táblázatok kezelésére képes, ahol a táblázat oszlopainak neve van. Ha egyetlen részből álló osztályt hozunk létre, amelybe minden szolgáltatást (a HTML megjelenítés minden apró részletével) beleépítünk, első látásra minden szépnek és jónak tűnhet. De ha visszajön az ügyfél, hogy szeretné, ha a program ízlésesen formázott szöveges kimenetet eredményezne, valószínűleg újabb tagfüggvényt kellene megpróbálnunk felvenni, melyek megoldják a problémát.

Egy-két héten belül ügyfelünk ráébred, hogy szeretné a programot elektronikus levelek küldésére használni és ha már úgyis fejlesztünk a programon, a cég belső hálózata az XML nyelvet használja, nem lehetne beépíteni a támogatást ehhez is? Ennél a pontnál, ha minden szolgáltatást egyetlen osztályba építünk bele, a program kezd ormótlanul nagy lenni, esetleg a teljes átírását fontolgatjuk.

Játsszuk el ezt a forgatókönyvet a `Tablázat` és a `HTMLTablázat` osztályainkkal! Alaposan sikerült elkülöníteni az adatok feldolgozását és megjelenítését. Ha ügyfelünk azzal a kéréssel fordul hozzánk, hogy szeretné a táblázatot fájlba menteni, csak egy új osztályt kell származtatnunk a `Tablázat` osztályból. Nevezzük ezt `TablázatFajl` osztálynak. Eddigi osztályainkhoz hozzá sem kell nyúlnunk. Ez igaz a `TablázatLevel` és az `XMLTablázat` osztályokra is. A 8.3. ábra az osztályok közötti kapcsolatokat (függőségeket, leszármazásokat) mutatja.

8.3. ábra

*Kapcsolatok
a `Tablázat` osztály és
gyermekai között*



Sőt, tudjuk, hogy minden, a `Tablázat` osztályból származtatott osztálynak van egy `kiir()` tagfüggvénye, így azokat egy tömbbe is gyűjthetjük. Azután végigszaladunk a tömbön, meghívjuk minden elem `kiir()` tagfüggvényét és nem kell azzal foglalkoznunk, hogy az adott elem a `Tablázat` melyik leszármazottja, a megfelelő kiíró függvény kerül meghívásra. Egy egyszerű `Tablázat` osztályból származtatott típusú objektumok tömbje segítségével elektronikus leveleket írhatunk, illetve HTML, XML vagy szöveges állományokat állíthatunk elő a `kiir()` függvény meghívásával.

Összefoglalás

Sajnos nincs rá mód, hogy az objektumközpontú programozást minden szempontból megvizsgáljuk egy rövidke óra alatt, de remélem, hogy a fontosabb lehetőségeket sikerült bemutatni.

Azt, hogy milyen mértékben használunk osztályokat a programokban, nekünk kell eldöntenünk. Az objektumközpontú megközelítést intenzíven használó programok általában némileg több erőforrást igényelnek, mint a hagyományosak, viszont a rugalmasság és átláthatóság jelentősen nő.

Ebben az órában arról tanultunk, hogyan hozhatunk létre osztályokat és belőlük objektumpéldányokat. Megtanultunk tulajdonságokat és tagfüggvényeket létrehozni és elérni. Végül azt is megtanultuk, hogyan hozhatunk létre új osztályokat más osztályokból az öröklés és a tagfüggvény-felülírás segítségével.

Kérdések és válaszok

Ebben az órában néhány barátságtalan fogalommal találkoztunk. Valóban szükséges az objektumközpontúságot megérteni ahhoz, hogy jó PHP programozó válhasson az emberből?

Erre a rövid válasz: nem. A legtöbb PHP programozó alig vagy egyáltalán nem ír objektumközpontú programot. Az objektumközpontú megközelítés nem tesz számunkra olyan dolgokat lehetővé, amelyek eddig elérhetetlenek voltak. A dolog lényege a programok szervezésében, az egyszer megírt programrészek újrahasznosításában és a könnyű fejlesztetőségben rejlik. Még ha el is határoznánk, hogy soha nem fogunk objektumközpontú programot írni, előfordulhat, hogy bele kell javítanunk mások által írt programokba, amelyekben osztályok vannak, ezért értenünk kell a szemlélet alapjait. Ez az óra ebben is segítséget nyújthat.

Nem értem a `$this` változó szerepét.

Egy osztályon belül szükséges az osztály tagfüggvényeit vagy tulajdonságait elérni. Ekkor a `$this` változót kell használni, amely egy mutató az adott objektumra, amelynek valamely tagfüggvényét meghívtuk. Mivel a `$this` változó mutató (hivatkozás), összetevőinek eléréséhez a `->` műveletjelet kell használni.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan hoznánk létre egy `uresOsztaly` nevű osztályt, amelynek nincsenek tagfüggvényei és tulajdonságai?
2. Adott egy `uresOsztaly` nevű osztály. Hogyan hoznánk létre egy példányát?
3. Hogyan lehet az osztályba egy tulajdonságot felvenni?
4. Hogyan kell a konstruktor nevét megválasztani?
5. Hogyan lehet egy osztályban konstruktort létrehozni?

6. Hogyan lehet létrehozni közöséges tagfüggvényeket?
7. Hogyan lehet osztályon belülről az osztály tulajdonságaihoz és tagfüggvénye-
ihez hozzáférni?
7. Hogyan lehet osztályon kívülről az osztály tulajdonságaihoz és tagfüggvénye-
ihez hozzáférni?
8. Mit kell tennünk, ha azt szeretnénk, hogy egy osztály más osztálytól
örököljön?

Feladatok

1. Hozzuk létre a `Szamolo` nevű osztályt, amely két egész számot tárol.
Írjunk hozzá egy `kiszamol()` nevű tagfüggvényt, amely kiírja a két számot
a böngészőbe!
2. Hozzuk létre az `Osszead` osztályt, amely a `Szamolo` osztálytól örököl.
Írjuk át ennek `kiszamol()` tagfüggvényét úgy, hogy az a két tulajdonság
összegét írja ki a böngészőbe!
3. Az előző feladathoz hasonlóan módon készítsük el a `Kivon` osztályt!



III. RÉSZ

Munka a PHP-vel

- 9. óra Űrlapok
- 10. óra Fájlok használata
- 11. óra A DBM függvények használata
- 12. óra Adatbázisok kezelése – MySQL
- 13. óra Kapcsolat a külvilággal
- 14. óra Dinamikus képek kezelése
- 15. óra Dátumok kezelése
- 16. óra Az adatok kezelése
- 17. óra Karakterláncok kezelése
- 18. óra A szabályos kifejezések használata
- 19. óra Állapotok tárolása sütikkel és GET típusú lekérdezésekkel
- 20. óra Állapotok tárolása munkamenet-függvényekkel
- 21. óra Munka kiszolgálói környezetben
- 22. óra Hibakeresés



9. ÓRA

Űrlapok

A könyv eddigi példáiban egy nagyon fontos dolgot nem láttunk. Létrehoztunk változókat és tömböket, készítettünk és meghívtunk különböző függvényeket, különféle objektumokkal dolgoztunk, eddigi ismereteink azonban értelmetlenek, amíg a felhasználó által megadott adatokat kezelni nem tudjuk. Ebben az órában ezt a témakört tekintjük át.

A Világhálón alapvetően a HTML űrlapokon keresztül áramlik az információ a felhasználó és a kiszolgáló között. A PHP-t úgy tervezték, hogy a kitöltött HTML űrlapokat könnyen fel tudja dolgozni.

Ebben az órában a következőket tanuljuk meg:

- Hogyan férjük hozzá a környezeti változókhoz és hogyan használjuk azokat?
- Hogyan férjük hozzá az űrlapmezőkben megadott információkhoz?
- Hogyan dolgozzunk az űrlapok több elem kiválasztását engedélyező elemeivel?

- Hogyan készítsünk olyan kódot, amely egyszerre tartalmazza a HTML űrlapot és az ezt kezelő PHP programot?
- Hogyan mentjük az állapotot rejtett mezőkkel?
- Hogyan irányítsuk a felhasználót új oldalra?
- Hogyan készítsünk fájlfeltöltő HTML űrlapot és hogyan írjunk olyan PHP programot, amely kezeli ezt?

Globális és környezeti változók

Mielőtt elkészítenénk első igazi űrlapunkat, kis kitérőt kell tennünk, hogy újra áttekintsük a globális változókat. A globális változókkal először a függvényekről szóló hatodik órában találkoztunk. A globális változók azok a változók, amelyeket a program legfelső szintjén, azaz a függvényeken kívül vezettünk be. Minden függvény számára elérhető a beépített `$GLOBALS` nevű tömb. A `$GLOBALS` tömb használatát láthatjuk a 9.1. példában, amelyben egy ciklussal kiírjuk programunk összes globális változóját.

9.1. program A `$GLOBALS` tömb elemeinek kiírása

```
1: <html>
2: <head>
3: <title>9.1. program A $GLOBALS tömb elemeinek
   kiírása</title>
4: </head>
5: <body>
6: <?php
7: $user1 = "Judit";
8: $user2 = "István";
9: $user3 = "János";
10: foreach ( $GLOBALS as $kulcs=>$ertek )
11:     {
12:         print "\$GLOBALS[\"$kulcs\"] == $ertek<br>";
13:     }
14: ?>
15: </body>
16: </html>
```

Három változót vezettünk be és a \$GLOBALS tömb elemein végiglépkedve kiírtuk a változók neveit és értékeit a böngészőben. Láthatjuk az általunk megadott változókat, de a program ezeken kívül másokat is kiírt, a PHP ugyanis automatikusan bevezet néhány globális változót, amelyek a kiszolgáló és az ügyfél környezetét írják le. Ezeket a változókat környezeti változóknak nevezzük. A kiszolgálótól, a rendszertől és a beállításoktól függően különféle környezeti változók létezhetnek, ezek ismerete rendkívül fontos. A 9.1. táblázatban a leggyakoribb környezeti változókat soroltuk fel. A környezeti változók értéke közvetlenül vagy a \$GLOBALS tömb részeként érhető el.

9.1. táblázat Környezeti változók

<i>Változó</i>	<i>Tartalma</i>	<i>Példa</i>
\$HTTP_USER_AGENT	A böngésző neve és változatszáma	Mozilla/4.6 (X11;I; Linux2.2.6-15apmac ppc)
\$REMOTE_ADDR	Az ügyfél IP címe	158.152.55.35
\$REQUESTED_METHOD	A kérelem módja (GET vagy POST)	POST
\$QUERY_STRING	A GET kérelmeknél az URL-hez kapcsolt kódolt adat	nev=janos&cim= ismeretlen
\$REQUEST_URI	A kérelem teljes címe a lekérdező karaktersorozattal	/php-konyv/urlapok/ 9.14.program.php?nev=janos
\$HTTP_REFERER	Az oldal címe, amelyről a kérelem érkezett	http://www.proba.hu/ egy_oldal.html

A PHP létrehoz más környezeti változókat is.

Például a \$GLOBALS ["PHP_SELF"] az éppen futó program elérési útját adja meg. A szerző rendszerén az érték a következő volt:

```
/php-konyv/urlapok/9.1.program.php
```

A változó értéke közvetlenül is elérhető, `$PHP_SELF` néven. Ebben az órában még sokszor fogjuk használni ezt a változót. A HTML oldalt leíró és az űrlapot elemző PHP kódot gyakran tároljuk egy állományban. Az oldal nevének tárolásához a `$PHP_SELF` változó értékét a HTML FORM elemének ACTION paraméteréhez rendeljük.

A `$GLOBALS` tömb ezenkívül még sok másra is használható.

Adatok bekérése a felhasználótól

Jelenleg a HTML és PHP kódot külön állományban tároljuk. A 9.2. példában egy egyszerű HTML űrlap kódját láthatjuk.

9.2. program Egy egyszerű HTML űrlap

```
1: <html>
2: <head>
3: <title>9.2. program Egy egyszerű HTML űrlap</title>
4: </head>
5: <body>
6: <form action="9.3.program.php" method="GET">
7: <input type="text" name="felhasznalo">
8: <br>
9: <textarea name="cim" rows="5" cols="40">
10: </textarea>
11: <br>
12: <input type="submit" value="rendben">
13: </form>
14: </body>
15: </html>
```

Egy HTML űrlapot hoztunk létre, amely egy "felhasznalo" nevű szövegmezőt, egy "cim" nevű szövegterületet, és egy gombot tartalmaz. Könyvünk nem foglalkozik részletesen a HTML ismertetésével, ha mégis nehéznek találjuk a példákat, olvassunk el egy, a HTML nyelvvel foglalkozó kötetet vagy számítógépes leírást. A FORM elem ACTION paramétere a `9.3.program.php` fájlra mutat, amely feldolgozza az űrlapon megadott adatokat. Mivel az ACTION csak a fájl nevét adja meg, a HTML és PHP kódot tartalmazó fájloknak egy könyvtárban kell lenniük.

A 9.3. példaprogram kiírja a felhasználó által megadott információkat.

9.3. program A 9.2. példa űrlapjának feldolgozása

```

1: <html>
2: <head>
3: <title>9.3. program A 9.2. példa űrlapjának
   feldolgozása</title>
4: </head>
5: <body>
6: <?php
7: print "Üdvözlét <b>$felhasznalo</b><P>\n\n";
8: print "A címe:<P>\n\n<b>$cim</b>";
9: ?>
10: </body>
11: </html>

```

Ez a könyv első olyan programja, amelyet nem hivatkozáson keresztül hívunk meg és nem közvetlenül a böngészőbe írjuk be a címét. A 9.3. példában található kódot az 9.3.program.php fájlban tároljuk. A fájlban lévő kódot akkor hívjuk meg, amikor a felhasználó kitölti a 9.2. példában található űrlapot. A program a \$felhasznalo és a \$cim változók értékét írja ki, amelyek a HTML űrlap "felhasznalo" szövegmezőjének és "cim" területének értékét tartalmazzák. Látható, hogy a PHP 4 segítségével milyen egyszerűen kezelhetők az űrlapok. Bármilyen megadott információ globális változóként hozzáférhető és a változók neve megegyezik a megfelelő HTML elem nevével.

Több elem kiválasztása a SELECT elemmel

Előző példánkban olyan HTML űrlap szerepelt, amely egy elemhez egyetlen érték hozzárendelését engedélyezte. Most megtanuljuk a SELECT elem kezelését, melynek segítségével több elem kiválasztását engedélyezhetjük. Ha a SELECT elemnek egyszerű nevet adunk:

```
<select name="termek" multiple>
```

akkor a feldolgozó program csak az egyik kiválasztott elemhez fér hozzá. Ha minden elemet el szeretnénk érni a programból, az elem neve után írunk üres szögletes zárójeleket. Ezt láthatjuk a 9.4. példában.

9.4. program SELECT elemet tartalmazó HTML űrlap

```
1: <html>
2: <head>
3: <title>9.4. program SELECT elemet tartalmazó HTML
   űrlap</title>
4: </head>
5: <body>
6: <form action="9.5.program.php" method="POST">
7: <input type="text" name="felhasznalo">
8: <br>
9: <textarea name="cim" rows="5" cols="40">
10: </textarea>
11: <br>
12: <select name="termek[]" multiple>
13: <option>Ultrahangos csavarhúzó
14: <option>Tricorder
15: <option>ORAC AI
16: <option>HAL 2000
17: </select>
18: <br>
19: <input type="submit" value="rendben">
20: </form>
21: </body>
22: </html>
```

Az űrlapot feldolgozó programban a SELECT elemben kiválasztott adatok a \$termekek tömbben találhatóak. Ezt mutatjuk be a 9.5. példában

9.5. program A 9.4. példában látott űrlap feldolgozása

```
1: <html>
2: <head>
3: <title>9.5. program A 9.4. példában látott űrlap
   feldolgozása</title>
4: </head>
5: <body>
6: <?php
7: print "Üdvözlét <b>$felhasznalo</b><P>\n\n";
8: print "A címe:<P>\n\n<b>$cim</b>";
9: print "A következő termékeket választotta:<p>\n\n";
10: print "<ul>\n\n";
```


9.5. program (folytatás)

```

11: foreach ( $termekek as $termek )
12:     {
13:         print "<li>$termek<br>\n";
14:     }
15: print "</ul>";
16: ?>
17: </body>
18: </html>
    
```

Nem csak a `SELECT` elem engedélyezi több elem kiválasztását. Ha több jelölőnégyzetnek ugyanazt a nevet adjuk, ahol a felhasználó több elemet is kiválaszthat, csak az utolsó kiválasztott értéket kapjuk meg, de ha a nevet üres szögletes zárójel követik, az adott nevű elemeket a PHP tömbként kezeli. A 9.4. példa `SELECT` elemét jelölőnégyzetekre cserélhetjük és ugyanazt a hatást érjük el:

```

<input type="checkbox" name="termekek[]" value="Ultrahangos
    ➔ csavarhúzó">Ultrahangos csavarhúzó<br>
<input type="checkbox" name="termekek[]"
    ➔ value="Tricorder">Tricorder<br>
<input type="checkbox" name="termekek[]" value=
    ➔ "ORAC AI">ORAC AI<br>
<input type="checkbox" name="termekek[]" value=
    ➔ "HAL 2000">HAL 2000<br>
    
```

Az űrlap minden mezőjének hozzárendelése egy tömbhöz

Az eddig megtanult módszerek mindaddig jól működnek, amíg programunk tudja, milyen mezőkkel dolgozik. Gyakran azonban olyan programra van szükségünk, amely az űrlap változásaihoz alkalmazkodik vagy egyszerre többféle űrlapot képes kezelni, anélkül, hogy keverné a mezők neveit. A PHP 4 globális változói erre a problémára is megoldást nyújtanak. Attól függően, hogy a kitöltött űrlap `GET` vagy `POST` metódust használt, elérhetővé válnak a `$HTTP_GET_VARS` vagy a `$HTTP_POST_VARS` változók. Ezek az asszociatív tömbök tartalmazzák az űrlap mezőinek neveit és a hozzájuk rendelt értékeket. A 9.6. példa azt mutatja be, hogyan listázható ki az űrlap minden mezője egy `GET` kérés esetén.

9.6. program A \$HTTP_GET_VARS tömb használata

```
1: <html>
2: <head>
3: <title>9.6. program A $HTTP_GET_VARS tömb
   használata</title>
4: </head>
5: <body>
6: <?php
7: foreach ( $HTTP_GET_VARS as $kulcs => $ertek )
8:     {
9:         print "$kulcs == $ertek<BR>\n";
10:    }
11: ?>
12: </body>
13: </html>
```

A fenti kód a GET kérelemn keresztül érkezett paraméterek neveit és értékeit írja ki. Természetesen itt még nem kezeltük azt az esetet, amikor valamelyik átadott paraméter tömb. Ha a 9.4. példaprogrammal olyan HTML űrlapról érkező adatok feldolgozását végeztetjük, amelyben több elem kiválasztását engedélyező SELECT mező szerepel, valami hasonlót olvashatunk:

```
felhasznalo == Kiss Iván
cim == Budapest, Magyarország
termekek == Array
```

A termekek tömböt tartalmazza a \$HTTP_GET_VARS tömb, de kódunk még nem tudja ezt kezelni. A 9.7. példaprogram ellenőrzi, hogy a paraméter típusa tömb-e és ha igen, kiírja a tömb elemeit.

9.7. program A \$HTTP_GET_VARS tömb használata, ha tömböt tartalmaz

```
1: <html>
2: <head>
3: <title>9.7. program A $HTTP_GET_VARS tömb
   használata, ha tömböt tartalmaz</title>
4: </head>
5: <body>
6: <?php
```

9.7. program (folytatás)

```

7: foreach ( $HTTP_GET_VARS as $kulcs => $ertek )
8:     {
9:         if ( gettype( $ertek ) == "array" )
10:            {
11:                print "$kulcs == <br>\n";
12:                foreach ( $ertek as $tombelem )
13:                    print ".....$tombelem<br>";
14:            }
15:        else
16:            {
17:                print "$kulcs == $ertek<br>\n";
18:            }
19:        }
20: ?>
21: </body>
22: </html>

```

Mielőtt a `foreach` segítségével a `$HTTP_GET_VARS` tömb elemein végiglépke-
nénk, a `gettype()` függvénnyel ellenőrizzük, hogy a következő elem tömb-e.
Ha az elem tömb, egy második `foreach` segítségével végiglépke-
dünk az elemein és kiírjuk azokat a böngészőbe.

Különbségek a GET és a POST továbbítás között

A program akkor rugalmas, ha el tudja dönteni, hogy `$HTTP_POST_VARS` vagy `$HTTP_GET_VARS` tömbből olvassa-e ki az elemeket. A továbbítás típusát (GET vagy POST) a legtöbb rendszerben a `$REQUEST_METHOD` környezeti változó tartalmazza, értéke értelemszerűen "get" vagy "post". Érdekes tudni, hogy a `$HTTP_POST_VARS` tömb csak POST továbbítási mód esetén tartalmaz elemeket.

A 9.8. program mindig a megfelelő tömbből olvassa ki a paramétereket.

9.8. program A GET és POST kérelmek kezelése

```
1: <html>
2: <head>
3: <title>9.8. program A GET és POST kérelmek
   kezelése</title>
5: </head>
6: <body>
7: <?php
8: $PARAMETEREK = ( count( $HTTP_POST_VARS ) )
9:     ? $HTTP_POST_VARS : $HTTP_GET_VARS;
10: foreach ( $PARAMETEREK as $kulcs => $ertekek )
11: {
12:     if ( gettype( $ertekek ) == "array" )
13:     {
14:         print "$kulcs == <br>\n";
15:         foreach ( $ertekek as $tombelem )
16:             print ".....$tombelem<br>";
17:     }
18:     else
19:     {
20:         print "$kulcs == $ertekek<br>\n";
21:     }
22: }
23: ?>
24: </body>
25: </html>
```

A feltételes műveletjellel beállítjuk a `$PARAMETEREK` változó értékét. A `count()` függvénnnyel ellenőrizzük, hogy a `$HTTP_POST_VARS` változó tartalmaz-e elemeket. A `count()` visszatérési értéke a tömb elemeinek száma: pozitív, ha a paraméterében megadott változó tartalmaz elemeket és 0 (`false`), ha nem.

Ha a `$HTTP_POST_VARS` tartalmaz elemeket, akkor a `$PARAMETEREK` változót egyenlővé tesszük vele, egyébként a `$PARAMETEREK` a `$HTTP_GET_VARS` értékét veszi fel. Most már kiírathatjuk a `$PARAMETEREK` tömb tartalmát a korábban látott módon.

PHP és HTML kód összekapcsolása egy oldalon

Néhány esetben szükség lehet rá, hogy az űrlapot leíró HTML kódot és az űrlapot kezelő PHP kódot egyetlen fájlban tároljuk. Ez akkor lehet hasznos, amikor a felhasználó számára többször adjuk át ugyanazt az űrlapot. Természetesen kódunk sokkal rugalmasabb, ha dinamikusan írjuk meg, de ekkor elveszítjük a PHP használatának előnyét. A HTML és PHP kódot ne keverjük a közös fájlban belül, mert így a forrás nehezebben lesz olvasható és módosítható. Ahol lehetséges, készítsünk HTML kódból meghívható PHP függvényeket, amelyeket később fel tudunk használni.

A következő példákban olyan oldalt fejlesztünk, amely az űrlapról beérkező egész számról megmondja, hogy kisebb vagy nagyobb-e egy előre megadott egész értéknél.

A 9.9. példában a fenti feladat megoldását tartalmazó HTML kódot láthatjuk. A kód egy egyszerű szövegmezőt és némi PHP kódot tartalmaz.

9.9. program Saját magát meghívó HTML kód

```
1: <html>
2: <head>
3: <title>9.9. program Saját magát meghívó
   HTML kód</title>
4: </head>
5: <body>
6: <form action="<?php print $PHP_SELF?>" method="POST">
7: Ide írja a tippjét: <input type="text" name="tipp">
8: </form>
9: </body>
10: </html>
```

A fenti űrlap saját magát hívja meg, mert a FORM elem ACTION paraméterének a \$PHP_SELF értéket adtunk. Vegyük észre, hogy nem készítettünk gombot, amely elküldi a beírt számot. Az újabb böngészők a szövegmező kitöltése és az ENTER lenyomása után automatikusan elküldik a megadott adatot, viszont néhány régebbi böngésző ezt nem teszi meg.

A 9.9. példában lévő program nem dinamikus, hiszen nem ír ki semmit, ami a felhasználótól függ. A 9.10. példában további PHP kódot építünk az oldalba. Először meg kell adnunk a számot, amit a felhasználónak ki kell találnia. A teljes változatban valószínűleg véletlenszámot állítanánk elő, most azonban egyszerűen megadjuk, mondjuk a 42-t. Ezután megnézzük, hogy az űrlapot kitöltötték-e már, különben

olyan változóval számolnánk, amely még nem létezik. A \$tipp változó létezésének ellenőrzésével megtudhatjuk, hogy az űrlapot kitöltötték-e már. Amikor az űrlap elküldi a "tipp" paramétert, a \$tipp változó globális változóként hozzáférhető lesz a programban. A \$tipp változó létezése esetén egészen biztosak lehetünk benne, hogy az űrlapot a felhasználó kitöltötte, így elvégezhetjük a további vizsgálatokat.

9.10. program Számkitalálós PHP program

```
1: <?php
2: $kitalalando_szam = 42;
3: $uzenet = "";
4: if ( ! isset( $tipp ) )
5:     {
6:         $uzenet = "Üdvözlöm a számkitalálós játékban!";
7:     }
8: elseif ( $tipp > $kitalalando_szam )
9:     {
10:         $uzenet = "A(z) $tipp túl nagy, próbáljon
            egy kisebbet!";
11:     }
12: elseif ( $tipp < $kitalalando_szam )
13:     {
14:         $uzenet = "A(z) $tipp túl kicsi, próbáljon
            egy nagyobbbat!";
15:     }
16: else // egyenlők kell, hogy legyenek
17:     {
18:         $uzenet = "Telitalálat!";
19:     }
20: ?>
21: <html>
22: <head>
23: <title>9.10. program Számkitalálós PHP program</title>
24: </head>
25: <body>
26: <h1>
27: <?php print $uzenet ?>
28: </h1>
29: <form action="<?php print $PHP_SELF?>" method="POST">
30: Ide írja a tippjét: <input type="text" name="tipp">
31: </form>
32: </body>
33: </html>
```

A program törzsében egy `if` szerkezettel vizsgáljuk a `$tipp` változót és adunk értéket az `$uzenet` változónak. Ha a `$tipp` változó még nem létezik, a felhasználó csak most lépett az oldalra, ezért üdvözljük. Különben megvizsgáljuk az előre megadott számot és a `$tipp` értékét, és ez alapján rendeljük az `$uzenet` változóhoz a megfelelő szöveget. Végül a HTML oldal törzsében csak ki kell írunk az `$uzenet` értékét. A program persze még továbbfejleszthető. A PHP és a HTML kódot szétválasztva tartva egy grafikus könnyen módosíthatja az oldalt, a programozó közreműködése nélkül.

Állapot mentése rejtett mezőkkel

A 9.10. példában található program nem tudja, hogy a felhasználó hányszor tip-pelt, egy rejtett mező bevezetésével azonban ez is számon tartható. A rejtett mező ugyanúgy működik, mint a szövegmező, de a felhasználó nem látja, hacsak meg nem nézi a HTML forráskódot. A 9.11. példában található programban bevezetünk egy rejtett mezőt, amely a találgatások számát tartalmazza.

9.11. program Állapot mentése rejtett mezővel

```

1: <?php
2: $kitalalando_szam = 42;
3: $uzenet = "";
4: $probalkozasok = ( isset( $probalkozasok ) ) ?
   ++$probalkozasok : 0;
5: if ( ! isset( $tipp ) )
6:     {
7:         $uzenet = "Üdvözlöm a számkitalálós játékban!";
8:     }
9: elseif ( $tipp > $kitalalando_szam )
10:    {
11:        $uzenet = "A(z) $tipp túl nagy, próbáljon
        egy kisebbet!";
12:    }
13: elseif ( $tipp < $kitalalando_szam )
14:    {
15:        $uzenet = "A(z) $tipp túl kicsi, próbáljon
        egy nagyobbbat!";
16:    }
17: else // egyenlők kell, hogy legyenek
18:    {
19:        $uzenet = " Telitalálat!";
20:    }
    
```

9.11. program (folytatás)

```
21: $tipp = (int) $tipp;
22: ?>
23: <html>
24: <head>
25: <title>9.11. program Állapot mentése rejtett
    mezővel</title>
26: </head>
27: <body>
28: <h1>
29: <?php print $uzenet ?>
30: </h1>
31: Tippelés sorszáma: <?php print $probalkozasok?>
32: <form action="<?php print $PHP_SELF?>" method="POST">
33: Ide írja a tippjét:
34: <input type="text" name="tipp" value="<?php
    print $tipp?>">
35: <input type="hidden" name="probalkozasok"
    value="<?php print $probalkozasok?>">
36: </form>
37: </body>
38: </html>
```

A rejtett mező neve "probalkozasok". A PHP segítségével kiírjuk a "probalkozasok" és a "tipp" mező értékét, hogy a felhasználó tudja, hány-szor próbálkozott és mit tippelt utoljára. Ez az eljárás akkor lehet hasznos, amikor a kitöltött űrlapot elemezzük. Ha az űrlapot rosszul töltötték ki, a felhasználó tud-ni fogja, mit rontott el. A beadott tipp értékét egész számmá alakítjuk a kérdőív-be írás előtt.



Egy kifejezés értékének kiírásához a böngészőben a `print()` vagy az `echo()` utasításokat használhatjuk. PHP módban ezt egyszerűbben is megtehetjük. Ha egyenlőségjelet (`=`) teszünk a PHP blokkot nyitó elem után, a böngésző az azt következő kifejezés értékét kiírja, így a

```
<? print $proba?>
```

helyett írhatjuk a következőt is:

```
<?=$proba?>
```


A PHP kód törzsében egy feltétellel megvizsgáljuk, hogy kell-e növelni a \$probalkozasok változó értékét. Ha a változó létezik, akkor növeljük az értékét, különben 0-ra állítjuk. A HTML kód törzsében folyamatosan kiírjuk, hányszor próbálkozott a felhasználó.



Ne bízunk meg feltétel nélkül a rejtett mezőkben. Nem tudhatjuk, hogy honnan származik az értékük. Nem mondjuk, hogy ne használjuk őket, csak azt, hogy nagy körültekintéssel tegyünk. A felhasználó a forrás megváltoztatásával könnyedén csalhat a programban. A rejtett mezők használata nem biztonságos.

A felhasználó átirányítása

Programunknak van egy nagy hátulütője. Az űrlap mindig újratöltődik, akár kitalálta a felhasználó a számot, akár nem. A HTML nyelvben sajnos elkerülhetetlen az egész oldal újratöltése, a felhasználót azonban átirányíthatjuk egy gratuláló oldalra, ha kitalálta a számot.

Amikor az ügyfélprogram elkezd a kapcsolatot a kiszolgálóval, elküld egy fejléct, amely különböző információkat tartalmaz az azt követő dokumentumról. A PHP ezt automatikusan megteszi, de a `header()` függvénnyel mi is beállíthatjuk a fejléc egyes elemeit. Ha a `header()` függvényt szeretnénk használni, biztosnak kell benne lennünk, hogy a böngészőnek nem írtunk ki semmit, különben a függvényhívás előtt a PHP elküldi a saját fejlécét. Ez mindenféle kiírás, még sortörés és szóköz karakter esetén is bekövetkezik. A `header()` függvény használatakor semmi nem lehet a függvényhívás előtt, így ellenőriznünk kell a felhasználandó külső állományokat is. A 9.12. példában egy jellemző PHP fejléct láthatunk.

9.12. példa Egy jellemző PHP fejléc

```
1: HEAD /php-konyv/urlapok/9.1.program.php HTTP/1.0
2: HTTP/1.1 200 OK
3: Date: Sun, 09 Jan 2000 18:37:45 GMT
4: Server: Apache/1.3.9 (Unix) PHP/4.0
5: Connection: close
6: Content-Type: text/html
```



A fejléct a Telnet programmal írathatjuk ki. Kapcsolódjunk a 80-as kapun a webkiszolgálóhoz és gépeljük be a következőt:

```
HEAD /utvonal/fajl.html HTTP/1.0
```

Az ügyfélprogram ekkor kiírja a fejléct.

Egy "Location" fejléc elküldésével a böngészőt egy másik lapra irányíthatjuk:

```
header( "Location: http://www.kiskapu.hu/" );
```

Tegyük fel, hogy készítettünk egy "gratulacio.html" oldalt, ahova átirányítjuk a felhasználót, amikor kitalálta a megadott számot. A megoldást a 9.13. példában találjuk.

9.13. program Fejléc küldése a header() függvénnyel

```
1: <?php
2: $kitalalando_szam = 42;
3: $uzenet = "";
4: $probalkozasok = ( isset( $probalkozasok ) ) ?
   ++$probalkozasok : 0;
5: if ( ! isset( $tipp ) )
6:     {
7:         $uzenet = "Üdvözlöm a számkitalálós játékban!";
8:     }
9: elseif ( $tipp > $kitalalando_szam )
10:    {
11:        $uzenet = "A(z) $tipp túl nagy, próbáljon
   egy kisebbet!";
12:    }
13: elseif ( $tipp < $kitalalando_szam )
14:    {
15:        $uzenet = "A(z) $tipp túl kicsi, próbáljon
   egy nagyobbbat!";
16:    }
17: else // egyenlők kell, hogy legyenek
18:     {
19:         header( "Location: gratulacio.html" );
20:         exit;
21:     }
22: $tipp = (int) $tipp;
```

9.13. program (folytatás)

```

23: ?>
24: <html>
25: <head>
26: <title>9.13. program Fejléc küldése a header()
    függvénnnyel</title>
27: </head>
28: <body>
29: <h1>
30: <?php print $uzenet ?>
31: </h1>
32: Típpelés sorszáma: <?php print $probalkozasok?>
33: <form action="<?php print $PHP_SELF?>" method="POST">
34: Ide írja a tippjét:
35: <input type="text" name="tipp" value="<?php print
    $tipp?>">
36: <input type="hidden" name="probalkozasok"
37:     value="<?php print $probalkozasok ?>">
38: </form>
39: </body>
40: </html>

```

Az if szerkezet else ágában a böngészőt átirányítjuk a "gratulacio.html" oldalra. A header() függvény meghívása után az oldalnak exit utasítással kell végződnie, hogy befejezze az űrlap vizsgálatát.

Fájlfeltöltő űrlapok és programok

Megtanultuk, hogyan kérhetünk be adatokat a felhasználotól, de a Netscape Navigator 2-től és az Internet Explorer 4-től kezdve lehetőségünk van fájlok feltöltésére is. Ebben a részben azzal foglalkozunk, hogyan lehet a fájlok feltöltését kezelni a PHP 4 segítségével.

Először létre kell hoznunk egy HTML űrlapot, amely tartalmazza a fájlfeltöltő mezőt, melynek egyik paramétere kötelezően a következő:

```
ENCTYPE="multipart/form-data"
```

A feltöltő mező előtt meg kell adnunk egy rejtett mezőt is. Ennek neve MAX_FILE_SIZE kell, hogy legyen, és a fogadandó fájl lehetséges legnagyobb méretét adja meg, bájtban. Nem lehet nagyobb, mint a php.ini fájlban megadott

upload_max_filesize értéke, amely alapértelmezés szerint 2 megabájt. Miután kitöltöttük a MAX_FILE_SIZE mezőt, a fájlt egy egyszerű INPUT elemmel tölthetjük fel, melynek TYPE paramétere "file". Bármilyen nevet adhatunk neki. A 9.14. példában a fájlt feltöltő HTML kódot láthatjuk.

9.14. program Fájlfeltöltő űrlap

```

1: <html>
2: <head>
3: <title>9.14. program Fájlfeltöltő űrlap</title>
4: </head>
5: <body>
6: <form enctype="multipart/form-data"
   action="<?print $PHP_SELF?>" method="POST">
7: <input type="hidden" name="MAX_FILE_SIZE"
   value="51200">
8: <input type="file" name="fajl"><br>
9: <input type="submit" value="feltöltés!">
10: </form>
11: </body>
12: </html>

```

Vegyük észre, hogy a program meghívja az oldalt, amely tartalmazza, hogy PHP kóddal kezeljük a fájlt. A legnagyobb fájl méretet 50 kilobájtban adjuk meg és "fajl"-nak nevezzük el.

Amikor a fájlt feltöltöttük, az egy ideiglenes könyvtárban tárolódik (ez alapbeállításban a /tmp, ha UNIX rendszerről van szó). A fájl elérési útját a betöltő mező nevével megegyező globális változó tartalmazza (példánkban a \$fajl). A fájlról a különböző információkat a PHP globális változókban tárolja. Nevük a fájl elérési útját tartalmazó változónév, kiegészítve a "name", "size" és "type" utótagokkal. A változók jelentését a 9.2. táblázat tartalmazza.

9.2. táblázat Feltöltött fájlhoz kapcsolódó globális változók

Változónév	Tartalma	Példa
\$fajl	A fájl ideiglenes elérési útja	/tmp/php5Pq3fU
\$fajl_name	A feltöltött fájl neve	test.gif
\$fajl_size	A feltöltött fájl mérete bájtban	6835
\$fajl_type	A feltöltött fájl típusa	image/gif

A PHP 4 a feltöltött fájlok adatainak tárolására beépített tömb típusú változókat használ. Ha egy vagy több fájlt töltünk fel egy űrlapon keresztül, a fájlok adatai a \$HTTP_POST_FILES tömbben tárolódnak. A tömb indexei a feltöltést meghatározó mezők nevei. A 9.3. táblázatban láthatjuk a tömb elemeinek nevét és értékét.

9.3. táblázat Feltöltött fájlhoz kapcsolódó globális változók

<i>Elem</i>	<i>Tartalma</i>	<i>Példa</i>
\$HTTP_POST_FILES["fajl"]["name"]	A feltöltött fájl neve	test.gif
\$HTTP_POST_FILES["fajl"]["size"]	A feltöltött fájl mérete bájtban	6835
\$HTTP_POST_FILES["fajl"]["type"]	A feltöltött fájl típusa	image/gif

A 9.15. példában látható program kiírja a rendelkezésre álló információkat a feltöltött fájlról, és ha az GIF formátumú, megjeleníti.

9.15. program Fájlfeltöltő program

```

1: <html>
2: <head>
3: <title>9.15. program - Fájlfeltöltő program</title>
4: </head>
5: <?php
6: $feltoltes_konyvtar = "/home/httpd/htdocs/feltoltesek";
7: $feltoltes_url = "http://php.kiskapu.hu/feltoltesek";
8: if ( isset( $fajl ) )
9:     {
10:    print "elérési út: $fajl<br>\n";
11:    print "név: $fajl_name<br>\n";
12:    print "méret: $fajl_size bájt<br>\n";
13:    print "típus: $fajl_type<p>\n\n";
14:    if ( $fajl_type == "image/gif" )
15:        {
16:    copy ( $fajl, "$feltoltes_konyvtar/$fajl_name") or die
        ("Nem lehet másolni");
17:
18:        print "<img
src=\"$feltoltes_url/$fajl_name\"><p>\n\n";
19:        }
20:    }

```

9.15. program (folytatás)

```
21: ?>
22: <body>
23: <form enctype="multipart/form-data" action="<?php print
    $PHP_SELF?>" method="POST">
24: <input type="hidden" name="MAX_FILE_SIZE"
    value="51200">
25: <input type="file" name="fajl"><br>
26: <input type="submit" value="feltöltés!">
27: </form>
28: </body>
```

A 9.15. példában először ellenőrizzük, hogy a `$fajl` globális változó létezik-e. Ha létezik, feltehetjük, hogy a feltöltés sikeres volt (legalábbis e példa erejéig).



Mindig csak megbízható forrásból fogadjunk fájlt és ellenőrizzük, hogy programunk számára megfelelő formátumú-e. A rosszindulatú látogatók olyan űrlapot készíthetnek, amelyben megegyező nevű elemek találhatók, de teljesen más tartalommal, így programunk nem a várt adatokat kapja meg. Lehet, hogy ez paranoiának tűnik, de jegyezzük meg, hogy a kiszolgálóoldali programozásban a paranoia jó dolog. Soha ne bízunk külső forrásból érkező adatokban, még akkor sem, ha a külső forrás egy olyan űrlap, amit éppen mi készítettünk.

Miután feltöltöttük a fájlt, a böngészőben megjelenítjük a `$fajl`, `$fajl_name`, `$fajl_size` és `$fajl_type` változók tartalmát, amelyek a fájlról információkat tartalmaznak.

Ezután ellenőrizzük a `$fajl_type` változót. Ha értéke `"image/gif"`, akkor az érkezett állományt GIF képként megjeleníthetjük. A típust a fájlkiterjesztés alapján állapítja meg a rendszer, ezért érdemes ellenőrizni, hogy a fájl valóban GIF formátumú-e. Ennek módjáról a tizenhetedik, karakterláncokkal foglalkozó órában tanulunk.

A `copy()` függvénnyel a feltöltött fájlt a kiszolgáló másik könyvtárába másolhatjuk. A `copy()` függvénynek két paramétert kell megadni: az első a másolandó fájl elérési útja, a második a fájl új elérési útja. A fájl eredeti elérési útja a `$fajl` változóban található. Az új elérési út számára létrehozunk egy `$feltoltes_konyvtar` nevű változót, amihez hozzáfűzzük a `$fajl_name` értéket. Ezután a `copy()` függvénnyel átmásoljuk a fájlt. UNIX rendszerben a kiszolgálóprogramok a `'nobody'` felhasználá-

lónév alatt futnak, ezért mielőtt másolatot készítenénk a fájlról, ellenőriznünk kell, hogy a művelet engedélyezett-e. Az `or` művelettel a `die()` függvényt használjuk, ha a másolás sikertelen. Ezt a módszert a tizedik, a fájlokkal foglalkozó órában részletesebben áttekintjük.

Másolás után az eredeti fájlt nem kell törölnünk, a PHP ezt megteszi helyettünk. Ha nem másoljuk vagy helyezzük át a feltöltött állományt, a fájl elvész, mivel az a PHP kód végrehajtása után törlődik az ideiglenes könyvtárból.

Amikor létrehozzuk a `$feltoltes_konyvtar` változót, létrehozzuk a `$feltoltes_url` változót is, a célkönyvtár URL címének tárolására. A lemásolt fájlt HTML kép elemként jelenítjük meg.

Összefoglalás

Eddig olyan dolgokat tanultunk, amelyek segítségével igazi interaktív környezetet hozhatunk létre. Van azonban még néhány tanulnivaló. A felhasználóról képesek vagyunk információkat szerezni, de mit tegyünk ezekkel? Például fájlba írhatjuk. Ez lesz a következő óra témája.

Ebben az órában megtanultuk, hogyan használjuk a `$GLOBALS` tömböt, az űrlapok adatait, a feltöltött fájlokat és a globális változókat. Megtanultuk, hogyan küldhetünk fejléctet az ügyfélnek a böngésző átirányítása érdekében. Megnéztük, hogyan listázzuk ki az űrlapról kapott adatokat és rejtett mezőket használtunk, adatok átadására a programtól a böngészőnek.

Kérdések és válaszok

Létre lehet hozni tömböt olyan elemek tárolására is, amelyek nem választómezővel vagy jelölőnégyzetekkel megadottak?

Igen. Minden elem, amely nevének végén üres szögletes zárójel van, tömbként tárolódik, így az elemek csoportba rendezhetők.

A `header()` függvény nagyon hasznosnak tűnik. Tanulunk még a HTTP fejlécekről?

Magáról a HTTP-ről még beszélünk a tizenharmadik órában.

Az elemek nevének automatikus változónévvé alakítása veszélyesnek tűnik. Kikapcsolható valahol ez a lehetőség?

Igen a `php.ini` fájl `register_globals` elemét kell `off`-ra állítani a lehetőség kikapcsolásához.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik környezeti változó tartalmazza a felhasználó IP címét?
2. Melyik környezeti változó tartalmaz a böngészőről információkat?
3. Hogyan nevezzük el az űrlap mezőjét, hogy a rá hivatkozó globális változó `$urlap_tomb` nevű tömb legyen?
4. Melyik beépített tömb tartalmazza a GET kérelmen keresztül érkező változókat?
5. Melyik beépített tömb tartalmazza a POST kérelmen keresztül érkező változókat?
6. Melyik függvényt használjuk a böngésző átirányítására? Milyen karakter-sorozatot kell átadnunk a függvénynek?
7. Hogyan lehet korlátozni egy űrlapról feltölthető fájl méretét?
8. Hogyan korlátozható az összes űrlapról vagy programból feltölthető fájl mérete?

Feladatok

1. Készítsünk számológép-programot, amelyben a felhasználó megadhat két számot és a rajtuk végrehajtandó műveletet (összeadás, kivonás, szorzás vagy osztás – ezeket ismerje a program).
2. Készítsünk programot, amely rejtett mező használatával megmondja, hogy a felhasználó hányszor használta az első feladatban szereplő számológépet.



10. ÓRA

Fájlok használata

A programozási nyelvek egyik legfontosabb lehetősége, hogy fájlokat lehet létrehozni, olvasni, írni velük. A PHP-nek szintén megvannak ezek a tulajdonságai. Ebben a fejezetben ezeket a lehetőségeket fogjuk áttekinteni:

- Hogyan ágyazzunk be fájlokat a dokumentumokba?
- Hogyan vizsgáljuk a fájlokat és könyvtárakat?
- Hogyan nyissunk meg egy fájlt?
- Hogyan olvassunk adatokat a fájlokból?
- Hogyan írjunk egy fájlba vagy fűzzünk hozzá?
- Hogyan zárjunk le egy fájlt?
- Hogyan dolgozzunk könyvtárakkal?

Fájlok beágyazása az `include()` függvénnyel

Az `include()` függvény lehetőséget ad arra, hogy fájlt ágyazzunk be a PHP dokumentumokba. A fájlban szereplő PHP kód úgy hajtódik végre, mintha a fődokumentum része lenne, ami hasznos, ha egy többoldalas programban külső kódokat szeretnénk beágyazni.

Eddig, ha olyan kódot írtunk, amelyet többször is fel akartunk használni, minden egyes helyre be kellett másolnunk azt. Ha hibát találtunk benne vagy lehetőségeit tovább akartuk bővíteni, mindenütt át kellett írunk. Ezekre a problémákra megoldás az `include()` függvény. Gyakran használt függvényeinket külső fájlba menthetjük és az `include()` függvénnyel futási időben tölthetjük be programunkba. Az `include()` függvény paramétere a fájl elérési útja. A 10.1. példában látható, hogyan ágyazható be egy fájl a PHP kódba az `include()` függvény segítségével.

10.1. program Az `include()` használata

```
1: <html>
2: <head>
3: <title>10.1. program Az include() használata</title>
4: </head>
5: <body>
6: <?php
7: include("10.2.program.php");
8: ?>
9: </body>
10: </html>
```

10.2. program A 10.1. példában szereplő beillesztett fájl

```
1: Engem ágyaztak be!!
```

A 10.1. példában található kódba az `10.2.program.php` fájl van beágyazva. Amikor lefuttatjuk a 10.1. példában található kódot, az "Engem ágyaztak be!!" szöveg jelenik meg, azaz szöveget ágyaztunk be a PHP kódba. Nem okoz ez gondot? A beágyazott fájl tartalma alapértelmezésben HTML formátumban jelenik meg. Ha végrehajtandó PHP kódot szeretnénk beágyazni, akkor azt az eredeti fájlban PHP címkék közé kell zárni. A 10.3. és 10.4. példában a beágyazott fájlban van a megjelenítő kód.

10.3. program Az include() függvény használata PHP kód végrehajtására más fájlban

```
1: <html>
2: <head>
3: <title>10.3. program Az include() függvény használata
   PHP kód végrehajtására más fájlban</title>
4: </head>
5: <body>
6: <?php
7: include("10.4.program.php");
8: ?>
9: </body>
10: </html>
```

10.4. program PHP kódot tartalmazó beágyazott fájl

```
1: <?php
2: print "Engem ágyasztak be!!<br>"
3: print "De most már össze tudok adni ... 4 + 4 = ".(4+4);
4: ?>
```

A PHP 4-ben a beágyazott fájlokban szereplő kódnak ugyanúgy lehet visszatérési értéket adni, mint a függvényeknek. A visszatérési értéket a fájl vége előtt egy return utasítás után kell meghatározni. A 10.5. és 10.6. példákban olyan fájlokat ágyazunk be a dokumentumba, amelyeknek visszatérési értékük van.

10.5. program PHP kód végrehajtása és visszatérési érték megadása az include() függvény használatával

```
1: <html>
2: <head>
3: <title>10.5. program PHP kód végrehajtása és
   visszatérési érték megadása az include()
   függvény használatával</title>
4: </head>
5: <body>
6: <?php
7: $eredmeny = include("10.6.program.php");
8: print "A visszatérési érték $eredmeny";
9: ?>
10: </body>
11: </html>
```

10.6. program Visszatérési értékkel rendelkező beágyazott fájl

```
1: <?php
2: $vissza = ( 4 + 4 );
3: return $vissza;
4: ?>
5: Ez a HTML soha nem jelenik meg, mert a visszatérés
   után írtuk!
```



PHP 3 használatakor a beágyazott fájlban csak akkor lehet visszatérési értéket adni a return utasítással, ha függvényben szerepel. A 10.6. példa a PHP 3-ban hibát okoz.

Ha az `include()` függvényt alkalmazzuk, megadhatunk feltételeket is. A megadott fájlt csak akkor ágyazza be az `include()`, ha a fájlra teljesülnek a feltételek. A következő kódban található beágyazás soha nem fog végrehajtódni:

```
$proba = false;
if ( $proba )
{
    include( "egy_fajl.txt" ); //nem ágyazza be
}
```

Ha az `include()` függvényt ciklusban használjuk, az `include()`-ban megadott fájl tartalma minden ismétlésnél bemásolódik és a fájlban lévő kód minden hívásnál végrehajtódik. A 10.7. példában található kódban az `include()` függvény egy `for` ciklus belsejében szerepel. Az `include()` paramétere minden ismétlésnél más fájlra mutat.

10.7. program Az include() használata cikluson belül

```
1: <html>
2: <head>
3: <title>10.7. program Az include() használata ciklu-
   son belül</title>
4: </head>
5: <body>
6: <?php
7: for ( $x = 1; $x<=3; $x++ )
8:     {
9:         $incfajl = "incfajl$x.txt";
```

10.7. program (folytatás)

```
10:     print "Megpróbálom beágyazni az $incfajl -t";
11:     include( "$incfajl" );
12:     print "<p>";
13:     }
14: ?>
15: </body>
16: </html>
```

10

A 10.7. programban található kód futás közben három különböző fájlt ágyaz be, ezek az "incfajl1.txt", "incfajl2.txt" és "incfajl3.txt". Mindegyik fájl tartalmazza a nevét és azt a szöveget, hogy beágyazták:

```
Megpróbálom beágyazni az incfajl1.txt -t
Ez a fájl az incfajl1.txt
Megpróbálom beágyazni az incfajl2.txt -t
Ez a fájl az incfajl2.txt
Megpróbálom beágyazni az incfajl3.txt -t
Ez a fájl az incfajl3.txt
```



Rendelkezésre áll egy másik PHP függvény is, a `require()`, amely hasonlóan működik az `include()`-hoz. Ciklusban azonban a `require()` nem használható úgy, mint az `include()`, ugyanis a `require()` a program futásának kezdetekor helyettesítődik be. Még egy fontos különbség a két függvény között, hogy a `require()`-nak nem lehet visszatérési értéke a PHP 4-esben.

Fájlok vizsgálata

Mielőtt elkezdünk ismerkedni a fájlokkal és könyvtárakkal, fussunk át néhány velük kapcsolatos tudnivalót. A PHP 4 igen sok lehetőséget ad arra, hogy a rendszerünkben található fájlokon különböző műveleteket hajtsunk végre. A következő bekezdésekben ezeket a függvényeket tekintjük át.

Fájl létezésének ellenőrzése a `file_exists()` függvénnyel

Fájl létezését a `file_exists()` függvénnyel vizsgálhatjuk. A `file_exists()` paramétere egy karakterlánc, amely a kérdéses fájl elérési útját és nevét tartalmazza. A függvény visszatérési értéke `true`, ha a fájl létezik, egyébként `false`.

```
if ( file_exists("proba.txt") )
    print("A fájl létezik!");
```

Fájl vagy könyvtár?

Az `is_file()` függvény azt dönti el, hogy paramétere fájl-e. A paraméterben meg kell adnunk egy elérési utat. Ha a megadott elérési út fájl jelöl, a függvény visszatérési értéke `true`, egyébként `false` lesz.

```
if ( is_file( "proba.txt" ) )
    print("a proba.txt egy fájl");
```

Ellenőrizhetjük azt is, hogy az elérési út könyvtárat jelöl-e. Az ellenőrzést az `is_dir()` függvény végzi. Ha a függvény paraméterében megadott elérési út könyvtárat határoz meg, a visszatérési érték `true`, egyébként `false` lesz.

```
if ( is_dir( "/tmp" ) )
    print "a /tmp egy könyvtár";
```

Fájl állapotának lekérdezése

Miután megtudtuk, hogy egy fájl létezik és valóban fájl, különféle dolgokat végezhethünk vele. Általában írunk bele, olvassuk vagy végrehajtjuk. A PHP-ben különböző függvények állnak rendelkezésre ezen lehetőségek lekérdezésére.

Az `is_readable()` függvénnyel megtudhatjuk, hogy az adott fájl olvasható-e számunkra. UNIX rendszerben lehet, hogy látjuk a fájlt a könyvtárszerkezetben, de nincs jogosultságunk annak olvasására. Ha az `is_readable()` paraméterében elérési úttal megadott fájl olvasható, visszatérési értéke `true`, egyébként `false` lesz.

```
if ( is_readable( "proba.txt" ) )
    print "a proba.txt olvasható";
```

Az `is_writeable()` függvénnyel megtudhatjuk, hogy az adott fájl írható-e számunkra. Ha az `is_writeable()` paraméterében elérési úttal megadott fájl írható, visszatérési értéke `true`, egyébként `false` lesz.

```
if ( is_writeable( "proba.txt" ) )
    print "a proba.txt írható";
```

Az `is_executable()` függvénnyel azt tudhatjuk meg, hogy az adott fájl futtatható-e. A visszatérési érték jogosultságunktól és a fájl kiterjesztésétől függ. Ha az `is_executable()` paraméterében elérési úttal megadott fájl futtatható, a visszatérési érték `true`, egyébként `false` lesz.

```
if ( is_executable( "proba.txt" )
    print "a proba.txt futtatható";
```

Fájl méretének lekérdezése a filesize() függvénnyel

A filesize() függvény a paraméterében elérési úttal megadott fájl bájtban számított méretével tér vissza. A végeredmény false, ha bármilyen hiba történik.

```
print "A proba.txt mérete: ";
print filesize( "proba.txt" );
```

Különféle fájlinformációk

Szükségünk lehet rá, hogy tudjuk, mikor hozták létre a fájlt vagy mikor használták utoljára. A PHP-vel ezeket az információkat is megszerezhetjük.

A fájl utolsó megnyitásának dátumát a fileatime() függvény segítségével kaphatjuk meg. A függvény paraméterében meg kell adni a fájl elérési útját. Visszatérési értéként az utolsó megnyitás dátumát kapjuk meg. A megnyitás jelenthet írást vagy olvasást is. Ezen függvények visszatérési értéke UNIX időbélyeg formátumú, azaz mindig az 1970 január elseje óta eltelt másodpercek száma. A következő példában ezt a számot a date() függvény segítségével olvasható formára alakítjuk. A date() függvény leírása a tizenötödik órában szerepel.

```
$atime = fileatime( "proba.txt" );
print "A proba.txt legutolsó megnyitásának dátuma:";
print date("Y.m.d. H:i", $atime);
//Egy minta végeredmény: 2000.01.23. 14:26
```

A filemtime() függvénnyel a fájl utolsó módosításának idejét kaphatjuk meg. A függvény paramétere a fájl elérési útja, visszatérési értéke pedig UNIX időbélyeg formátumú. A módosítás azt jelenti, hogy a fájl tartalma valamilyen módon megváltozott.

```
$mtime = filemtime( "proba.txt" );
print "A proba.txt utolsó módosításának dátuma:";
print date("Y.m.d. H:i", $mtime);
//Egy minta végeredmény: 2000.01.23. 14:26
```

A filectime() a fájl utolsó változásának időpontját adja meg. UNIX rendszerben a változás alatt azt értjük, amikor a fájl tartalma valamilyen módon módosul vagy a jogosultságok, illetve a tulajdonos megváltozik. Más rendszerekben a filectime() visszatérési értéke a fájl létrehozásának dátuma.

```
$ctime = filectime( "proba.txt" );  
print "A proba.txt utolsó változásának dátuma:";  
print date("Y.m.d. H:i", $ctime);  
//Egy minta végeredmény: 2000.01.23. 14:26
```

Több fájl tulajdonságot egyszerre megadó függvény

A 10.8. példában olyan függvényt hozunk létre, amely az előzőekben látott lehetőségeket fogja össze egyetlen függvényben.

10.8. program Egyszerre több fájl tulajdonságot megadó függvény

```
1: <html>  
2: <head>  
3: <title>10.8. program Egyszerre több fájl tulajdonságot  
   megadó függvény</title>  
4: </head>  
5: <body>  
6: <?php  
7: $fajl = "proba.txt";  
8: fileInformaciok( $fajl );  
9: function fileInformaciok( $f )  
10: {  
11:     if ( ! file_exists( $f ) )  
12:     {  
13:         print "$f nem létezik <BR>";  
14:         return;  
15:     }  
16:     print "$f ".(is_file( $f )?"":"nem ")."fájl<br>"  
17:     print "$f ".(is_dir( $f )?"":"nem ").  
       "könyvtár<br>"  
18:     print "$f ".(is_readable( $f )?"":"nem ").  
       "olvasható<br>"  
19:     print "$f ".(is_writeable( $f )?"":"nem ").  
       "írható<br>"  
20:     print "$f ".(is_executable( $f )?"":"nem ").  
       "futtatható<br>"  
21:     print "$f ".(filesize( $f ))."bájt méretű<br>"  
22:     print "$f utolsó megnyitásának dátuma: ".  
       date( "Y.m.d. H:i", fileatime( $f ) ).  
       "<br>";  
23:     print "$f utolsó módosításának dátuma: ".  
       date( "Y.m.d. H:i", filemtime( $f ) ).  
       "<br>";
```


10.8. program (folytatás)

```
24:         print "$f utolsó változásának dátuma: ".  
            date( "Y.m.d. H:i", filectime( $f ) ).  
            "<br>";  
25:     }  
26:  
27: ?>  
28: </body>  
29: </html>
```

10

Az egyszerűbb leírás kedvéért : műveleteket használtunk. Nézzük meg ezek közül az egyiket:

```
print "$f ".(is_file( $f )?"":"nem ")."fájl<br>"
```

A műveletjel bal oldalára került az `is_file()` függvény. Ha ennek eredménye `true`, akkor az operátor visszatérési értéke üres karakterlánc, egyébként a "nem " szöveg. A visszatérési érték után a szövegösszefűzés jelet követő karakterlánc is kiíródik. Az előbbi kifejezés kevésbé tömör formában a következőképpen írható le:

```
$igen_vagy_nem = is_file( $f ) ? "" : "nem ";  
print "$f $igen_vagy_nem"."fájl<br>"
```

Az `if` vezérlési szerkezettel még világosabb kódot írhatunk, ekkor azonban a program mérete jelentősen nő.

```
if ( is_file( $f ) )  
    print "$f fájl<br>";  
else  
    print "$f nem fájl<br>";
```

Mivel az előző példák ugyanannak a feladatnak a megoldásai, szabadon választhatjuk ki a nekünk tetszőt.

Fájlok létrehozása és törlése

Ha egy fájl nem létezik, a `touch()` függvény segítségével hozhatjuk létre. A függvény paraméterében meg kell adni egy elérési utat. Ezen az elérési úton próbál meg létrehozni a `touch()` egy üres fájlt. Ha a fájl már létezik, tartalma nem változik meg, de a módosítás dátuma a `touch()` függvény végrehajtási idejére módosul.

```
touch("sajat_fajl.txt");
```

Létező fájlt törölni az `unlink()` függvénnyel lehet. Az `unlink()` paramétere a fájl elérési útja:

```
unlink("sajat_fajl.txt");
```

A létrehozás, törlés, írás, olvasás, módosítás csak akkor lehetséges egy fájlra, ha a megfelelő jogosultságokkal rendelkezünk.

Fájl megnyitása írásra, olvasásra, hozzáfűzésre

Mielőtt elkezdünk dolgozni egy fájlal, meg kell nyitnunk írásra vagy olvasásra. Ezt az `fopen()` függvénnyel tehetjük meg. A függvény paraméterében meg kell adni a fájl elérési útját és a megnyitási módot. A legtöbbször használt módok az olvasás ("r"), írás ("w"), hozzáfűzés ("a"). A függvény visszatérési értéke egy egész szám. Ez az egész szám az úgynevezett fájlazonosító, amelyet változóként tárolhatunk. Ezzel hivatkozhatunk később a fájlra. Fájl olvasásra való megnyitásához a következőt kell beírnunk:

```
$fa = fopen( "proba.txt", 'r' );
```

Az írásra való megnyitáshoz a következőt:

```
$fa = fopen( "proba.txt", 'w' );
```

Ha a fájlt hozzáfűzésre akarjuk megnyitni, tehát nem kívánjuk felülírni a tartalmát, csak a végéhez szeretnénk fűzni valamit, a következőt kell tennünk:

```
$fa = fopen( "proba.txt", 'a' );
```

Az `fopen()` `false` értékkel tér vissza, ha valamiért nem tudta megnyitni a fájlt, ezért érdemes mindig ellenőrizni, hogy sikeres volt-e a megnyitás. Ezt például az `if` vezérlési szerkezettel tehetjük meg:

```
if ( $fa = fopen( "proba.txt", "w" ) )  
{  
    // $fa-val csinálunk valamit  
}
```

Esetleg használhatunk logikai műveletet, hogy megszakítsuk a végrehajtást, ha a fájl nem létezik:

```
( $fa = fopen( "proba.txt", "w" ) ) or die  
    ➤ ("A fájl sajnos nem nyitható meg!");
```

Ha az `fopen()` `true` értékkel tér vissza, a `die()` nem hajtódik végre, különben a kifejezés jobb oldalán a `die()` kiírja a paraméterében szereplő karakterláncot és megszakítja a program futását.

Amikor befejeztük a munkát egy fájjal, mindig be kell zárunk azt. Ezt az `fclose()` függvénnyel tehetjük meg, amelynek paraméterében azt a fájlazonosítót kell megadnunk, amelyet egy sikeres `fopen()` végrehajtása során kaptunk:

```
fclose( $fa );
```

10

Olvasás fájlból

A PHP-ben egy fájlból különböző függvények segítségével bájtontként, soronként vagy karakterenként olvashatunk.

Sorok olvasása fájlból az `fgets()` és

`feof()` függvényekkel

Miután megnyitottunk egy fájlt, beolvashatjuk a tartalmát sorról sorra, az `fgets()` függvénnyel. A függvény paraméterében meg kell adni az olvasandó fájl azonosítóját (amit az `fopen()` függvény ad a megnyitáskor), továbbá második paraméterként kell adnunk egy egész számot is, amely meghatározza, hogy legfeljebb hány bájtot olvasson ki a PHP, amíg sorvége vagy fájlvége jelet nem talál. Az `fgets()` függvény addig olvas a fájlból, amíg újsor karakterhez (`"\n"`) nem ér, a megadott bájtnyi adatot ki nem olvassa vagy a fájl végét el nem éri.

```
$sor = fgets( $fa, 1024 ); // ahol az $fa az fopen() által  
    ➤ visszaadott fájlazonosító
```

Tudnunk kell tehát, mikor érünk a fájl végére. Ezt az `feof()` függvény adja meg, melynek visszatérési értéke `true`, ha a fájl végére értünk, egyébként `false`. A függvény paraméterében egy fájlazonosítót kell megadni.

```
feof( $fa ); // ahol az $fa az fopen() által visszaadott  
    ➤ fájlazonosító
```

A 10.9. példában láthatjuk, hogyan kell egy fájlt sorról sorra beolvasni.

10.9. program Fájl megnyitása és sorról sorra olvasása

```
1: <html>
2: <head>
3: <title>10.9. program Fájl megnyitása és sorról sorra
   olvasása</title>
4: </head>
5: <body>
6: <?php
7: $fajlnev = "proba.txt";
8: $fa = fopen( $fajlnev, "r" ) or die("$fajlnev nem
   nyitható meg");
9: while ( ! feof( $fa ) )
10: {
11:     $sor = fgets( $fa, 1024 );
12:     print "$sor<br>";
13: }
14: fclose($fa);
15: ?>
16: </body>
17: </html>
```

Az `fopen()` függvénnyel próbáljuk olvasásra megnyitni a fájlt. Ha a kísérlet sikertelen, a `die()` függvénnyel megszakítjuk a program futását. Ez legtöbbször akkor történik meg, ha a fájl nem létezik vagy (UNIX rendszerben) nincs megfelelő jogosultságunk a megnyitásra. Az olvasás a `while` ciklusban történik. A `while` ciklus minden ismétlés alkalmával megnézi az `feof()` függvénnyel, hogy az olvasás elért-e a fájl végéhez, tehát a ciklus addig olvas, amíg el nem éri a fájl végét. Az `fgets()` minden ismétlésnél kiolvas egy sort vagy 1024 bájtot. A kiolvasott karaktersorozatot a `$sor` változóba mentjük és kiírjuk a böngészőbe. Minden kiírásnál megadunk egy `
` címkét, hogy a szöveg olvashatóbb legyen, végül bezárjuk a megnyitott állományt.

Tetszőleges mennyiségű adat olvasása fájlból

A fájlokból nem csak soronként, hanem előre meghatározott méretű darabokat is olvashatunk. Az `fread()` függvény paraméterében meg kell adni egy fájlazonosítót és az egyszerre kiolvasandó adatok mennyiségét, bájtban. A függvény visszatérési értéke a megadott mennyiségű adat lesz, kivéve, ha elérte a fájl végét.

```
$reszlet = fread( $fa, 16 );
```

A 10.10. példaprogram 16 bájtanként olvassa be a fájlt.

10.10. program Fájlb beolvasása az fread() függvénnnyel

```
1: <html>
2: <head>
3: <title>10.10. program Fájlb beolvasása az fread()
   függvénnnyel</title>
4: </head>
5: <body>
6: <?php
7: $fajlnev = "proba.txt";
8: $fa = fopen( $fajlnev, "r" ) or die("$fajlnev nem
   nyitható meg");
9: while ( ! feof( $fa ) )
10: {
11:     $reszlet = fread( $fa, 16 );
12:     print "$reszlet<br>";
13: }
14: fclose($fa);
15: ?>
16: </body>
17: </html>
```

Bár az fread() függvénnnek megadjuk, mennyi adatot olvasson ki, azt nem tudjuk meghatározni, hogy honnan kezdje az olvasást. Erre az fseek() ad lehetőséget, ezzel a függvénnnyel állítható be az olvasás helye. Paraméterként egy fájlazonosítót és egy egész számot kell megadnunk, amely a fájl elejétől bájtban mérve meghatározza az új olvasási helyet:

```
fseek( $fa, 64 );
```

A 10.11. példában az fseek() és fread() függvényekkel a fájl tartalmának második felét jelenítjük meg a böngészőben.

10.11. program Az fseek() használata

```
1: <html>
2: <head>
3: <title>10.11. program Az fseek() használata</title>
4: </head>
```

10.11. program (folytatás)

```
5: <body>
6: <?php
7: $fajlnev = "proba.txt";
8: $fa = fopen( $fajlnev, "r" ) or die("$fajlnev nem
    nyitható meg");
9: $fajlmeret = filesize($fajlnev);
10: $felmeret = (int)( $fajlmeret / 2 );
11: print "A fájl felének mérete: $felmeret <br>\n"
12: fseek( $fa, $felmeret );
13: $reszlet = fread( $fa, ($fajlmeret - $felmeret) );
14: print $reszlet;
15: fclose($fa);
16: ?>
17: </body>
18: </html>
```

A felezőpontot úgy számoltuk ki, hogy a `filesize()` függvénnyel lekértük a fájl méretét és elosztottuk kettővel. Ezt az értéket használtuk az `fseek()` második paramétereként. Végül az `fread()` segítségével kiolvastuk a fájl második felét és kiírtuk a böngészőbe.

Fájl karakterenkénti olvasása az `fgetc()` függvénnyel

Az `fgetc()` függvény hasonlít az `fgets()` függvényhez, de minden alkalommal, amikor meghívjuk, csak egyetlen karaktert olvas ki a fájlból. Mivel egy karakter mindig egy bájt méretű, más paramétert nem kell megadnunk, csak a már megszokott fájlazonosítót:

```
$karakter = fgetc( $fa );
```

A 10.12. példaprogram egy ciklus segítségével karakterenként kiolvassa a "proba.txt" tartalmát és minden karaktert külön sorba ír ki a böngészőbe.

10.12. program Az `fgetc()` használata

```
1: <html>
2: <head>
3: <title>10.12. program Az fgetc() használata</title>
4: </head>
5: <body>
```

10.12. program (folytatás)

```
6: <?php
7: $fajlnev = "program.txt";
8: $fa = fopen( $fajlnev, "r" ) or die("$fajlnev nem
  nyitható meg");
9: while ( ! feof( $fa ) )
10: {
11:     $karakter = fgetc( $fa );
12:     print "$karakter<br>";
13: }
14: fclose($fa);
15: ?>
16: </body>
17: </html>
```

10

Fájlba írás és hozzáfűzés

A fájlba írás és a hozzáfűzés nagyon hasonlítanak egymáshoz. Az egyetlen különbség az `fopen()` hívásában rejlik. Amikor írásra nyitjuk meg a fájlt, akkor az `fopen()` második paramétereként a "w" karaktert kell megadnunk:

```
$fa = fopen( "proba.txt", "w" );
```

Minden írási próbálkozás a fájl elején történik. Ha a fájl még nem létezne, a rendszer létrehozza azt. Ha a fájl létezik, tartalma felülíródik a megadott adatokkal.

Ha a fájlt hozzáfűzésre nyitjuk meg, az `fopen()` második paramétereként az "a" karaktert kell megadnunk:

```
$fa = fopen( "proba.txt", "a" );
```

Hozzáfűzésnél minden adat a fájl végére íródik, megtartva az előző tartalmat.

Fájlba írás az `fwrite()` és `fputs()` függvényekkel

Az `fwrite()` függvény paramétereként egy fájlazonosítót és egy karakterláncot kell megadni. A karakterlánc a megadott fájlba íródik. Az `fputs()` ugyanígy működik.

```
fwrite( $fa, "hello világ" );
fputs( $fa, "hello világ" );
```

A 10.13. példában először az `fwrite()` függvény használatával egy fájlba írunk, majd az `fputs()` függvénnyel adatokat fűzünk hozzá.

10.13. program Fájlba írás és hozzáfűzés

```
1: <html>
2: <head>
3: <title>10.11. program Fájlba írás és
   hozzáfűzés</title>
4: </head>
5: <body>
6: <?php
7: $fajlnev = "proba.txt";
8: print "$fajlnev fájlba írás";
9: $fa = fopen( $fajlnev, "w" ) or die("$fajlnev nem
   nyitható meg");
10: fwrite ( $fa, "Hello világ\n");
11: fclose( $fa );
12: print "$fajlnev fájlhoz hozzáfűzés";
13: $fa = fopen( $fajlnev, "a" ) or die("$fajlnev nem
   nyitható meg");
14: fputs ( $fa, "És más dolgok");
15: fclose( $fa );
16: ?>
17: </body>
18: </html>
```

Fájlok zárolása az `flock()` függvénnyel

Az eddig megtanultak igen jól használhatók, ha programjaink csak egyetlen felhasználót szolgálnak ki. A valóságban azonban egy alkalmazás oldalait általában többen is el szeretnék érni egyidőben. Képzeljük el, mi történne, ha egyszerre két felhasználó írna ugyanabba a fájlba. A fájl használhatatlanná válna.

Az `flock()` függvény használatával kizárhatjuk ezt a lehetőséget. Az `flock()` függvénnyel zárolt fájlt nem olvashatja vagy írhatja más folyamat, amíg a zárolás érvényben van. Az `flock()` függvény paramétereként egy fájlazonosítót és egy egész számot kell megadni. Ez utóbbi a zárolás típusát határozza meg. A lehetséges zárolásokat a 10.1. táblázat mutatja.

10.1. táblázat Az flock() függvény második paraméterének lehetséges értékei

<i>Egész</i>	<i>Lezárás típusa</i>	<i>Leírás</i>
1	Megosztott	Más folyamatok olvashatják a fájlt, de nem írhatnak bele (akkor használjuk, amikor olvassuk a fájlt)
2	Kizáró	Más folyamatok nem olvashatják és nem írhatják a fájlt (akkor használjuk, amikor írunk a fájlba)
3	Felszabadítás	A fájl zárolásának megszüntetése

10

Az flock() függvényt a fájl megnyitása után alkalmazzuk zárolásra és az fclose() előtt oldjuk fel vele a zárat.

```
$fa = fopen( "proba.txt", "a" );
flock( $fa, 2 ); // kizáró lefoglalás
// fájlba írás
flock( $fa, 3 ); // zárolás feloldása
fclose( $fa );
```



Az flock() a PHP zárolási megoldása. Csak azok a programok veszik figyelembe ezt a fajta zárolást, ahol ugyanezt a módszert alkalmazzuk, ezért előfordulhat, hogy a nem-PHP programok megnyitják a fájlunkat, akár írás közben is.

Munka könyvtárakkal

Az előzőekben áttekintettük, hogyan kezelhetjük a fájlokat, most megnézzük, hogyan hozhatunk létre, törölhetünk és olvashatunk könyvtárakat a PHP-ben.

Könyvtár létrehozása az mkdir() függvénnyel

Könyvtárat az mkdir() függvénnyel hozhatunk létre. Az mkdir() paraméterében meg kell adni egy karakterláncot, amely a létrehozandó könyvtár elérési útja, valamint egy oktális (nyolcas számrendszerű) számot, amely a jogosultságokat határozza meg. Az oktális számok elé mindig 0-t kell írni. A jogosultság megadásának csak UNIX rendszerekben van hatása. A jogosultsági mód három 0 és 7 közé eső számot tartalmaz, amelyek sorban a tulajdonos, a csoport, és mindenki más

jogait adják meg. A függvény visszatérési értéke `true`, ha a könyvtárat sikerült létrehozni, egyébként `false`. A könyvtár létrehozása általában akkor nem sikeres, ha a megadott elérési úton nincs jogosultságunk könyvtárat létrehozni, azaz nincs jogosultságunk írásra.

```
mkdir( "proba_konyvtar", 0777 ); // teljes  
➔ írás/olvasás/végrehajtás jogok
```

Könyvtár törlése az `rmdir()` függvénnyel

A rendszerből könyvtárat az `rmdir()` függvénnyel törölhetünk. A sikeres törléshez megfelelő jogosultsággal kell rendelkezünk és a könyvtárnak üresnek kell lennie. A függvény paraméterében a törlendő könyvtár elérési útját kell megadni.

```
rmdir( "proba_konyvtar" );
```

Könyvtár megnyitása olvasásra

Mielőtt be tudnánk olvasni egy könyvtár tartalmát, szükségünk van egy könyvtárazonosítóra. Ezt az azonosítót az `opendir()` függvény adja meg. A függvény paraméterében annak a könyvtárnak az elérési útját kell átadni, amelyet olvasni szeretnénk. A függvény visszatérési értéke a könyvtár azonosítója, kivéve, ha a könyvtár nem létezik vagy nincs jogosultságunk az olvasására. Ezekben az esetekben a visszatérési érték `false`.

```
$kvt = opendir( "proba_konyvtar" );
```

Könyvtár tartalmának olvasása

Ahogy az `fgets()` függvénnyel fájlból olvastunk, ugyanúgy használhatjuk a `readdir()` függvényt, hogy fájl vagy könyvtárnevet olvassunk ki egy megnyitott könyvtárból. A `readdir()` paraméterében meg kell adni az olvasandó könyvtár azonosítóját. A függvény visszatérési értéke a könyvtár következő elemének neve. Ha a könyvtár végére értünk, a visszatérési érték `false`. A `readdir()` csak az elem nevét és nem annak elérési útját adja meg. A 10.4. példában a `readdir()` függvény használati módját láthatjuk.

10.14. program Könyvtár tartalmának kiírása

```
1: <html>  
2: <head>  
3: <title>10.14. program Könyvtár tartalmának  
   kiírása</title>  
4: </head>
```

10.14. program (folytatás)

```
5: <body>
6: <?php
7: $kvtnev = "proba_konyvtar";
8: $kvt = opendir( $kvtnev );
9: while ( gettype( $fajl = readdir( $kvt ) ) != boolean )
10: {
11:     if ( is_dir( "$kvtnev/$fajl" ) )
12:         print "(D)";
13:     print "$fajl<br>";
14: }
15: closedir( $kvt );
16: ?>
17: </body>
18: </html>
```

10

A könyvtárat megnyitjuk az `opendir()` függvénnyel, majd egy `while` ciklussal végiglépkedünk annak összes elemén. A `while` ciklus feltételes kifejezésében meghívjuk a `readdir()` függvényt és a visszaadott értéket hozzárendeljük a `$fajl` változóhoz. A ciklus törzsében az `is_dir()` függvénnyel vizsgáljuk, hogy a `$kvtnev` és a `$fajl` változókból készített elérési út könyvtárat jelöl-e. Ha igen, neve elé teszünk egy "(D)" jelet. Így kiírjuk a böngészőbe a könyvtár tartalmát.

A `while` ciklus feltételes kifejezésének megírásakor igen elővigyázatosak voltunk. Sok PHP programozó a következőt használta volna:

```
while ( $fajl = readdir( $kvt ) )
{
    print "$fajl<br>\n";
}
```

Itt a `readdir()` visszatérési értékét vizsgáljuk. Minden "0"-tól különböző karakterlánc `true`-ként viselkedik, tehát ebből nem lehet probléma. Mi történik, ha könyvtárunk négy fájlt tartalmaz, melyek nevei "0", "1", "2", "3". Az előbbi kód a következő végeredményt adja:

```
.
..
```

Amikor a ciklus eléri a "0" nevű fájlt, a `readdir()` `false` értéknek veszi és a ciklus leáll. A 10.14. példában a `readdir()` függvény visszatérési értékének típusát vizsgáljuk és ezzel oldjuk meg a problémát.

Összefoglalás

Ebben az órában megtanultuk, hogyan ágyazhatunk be a dokumentumokba külső fájlban tárolt PHP kódot. Áttekintettük a fájlok különböző tulajdonságainak ellenőrzését, megnéztünk, hogyan olvashatunk fájlokat sorról sorra, karakterenként, vagy meghatározott részletekben. Megtanultuk, hogyan írhatunk fájlba és hogyan fűzhetünk karakterláncokat hozzá. Végül áttekintettük, hogyan hozhatunk létre, törölhetünk, vagy listázhatunk ki könyvtárakat.

Kérdések és válaszok

Az `include()` függvény lassítja a programok futását?

Mivel a beágyazott fájl meg kell nyitni és a tartalmát be kell olvasni, ezért azt kell mondjuk, hogy igen, a lassulás azonban nem számottevő.

Mindig le kell állítani a program futását, ha egy fájl nem sikerült megnyitni írásra vagy olvasásra?

Ha a program futásához elengedhetetlen a fájl, akkor a `die()` függvénnyel érdemes leállítani. Más esetben a program futását nem fontos megszakítani, de érdemes figyelmeztetni a felhasználót a hibára vagy feljegyezni a sikertelen megnyitási kísérletet egy naplóállományba. A huszonkettedik órában többet olvashatunk erről a megoldásról.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvénnyel adható programunkhoz külső kódot tartalmazó fájl?
2. Melyik függvénnyel tudjuk meg, hogy rendszerünkben megtalálható-e egy fájl?
3. Hogyan kapható meg egy fájl mérete?
4. Melyik függvénnyel nyitható meg egy fájl írásra vagy olvasásra?
5. Melyik függvénnyel olvashatunk ki egy sort egy fájlból?
6. Honnan tudhatjuk meg, hogy elértük a fájl végét?

7. Melyik függvényt használjuk, hogy egy sort írjunk egy fájlba?
8. Hogyan nyitunk meg egy könyvtárt olvasásra?
9. Melyik függvényt használjuk, hogy egy könyvtár elemeinek nevét megkapjuk, miután megnyitottuk azt?

Feladatok

1. Készítsünk egy oldalt, amely bekéri a felhasználó vezeték- és keresztnévét. Készítsünk programot, amely ezen adatokat fájlba menti.
2. Készítsünk programot, amely az előbbi feladatban mentett adatokat kiolvassa a fájlból. Írjuk ki azokat a böngészőbe (minden sor végén használjunk `
` címkét). Írjuk ki, hány sort tartalmaz a fájl, illetve a fájl méretét.



11. ÓRA

A DBM függvények használata

Ha nem is férünk hozzá valamilyen SQL adatbáziskezelőhöz (mint a MySQL vagy az Oracle), majdnem biztos, hogy valamilyen DBM-szerű adatbázisrendszer rendelkezésünkre áll. Ha mégsem, a PHP akkor is képes utánozni számunkra egy ilyen rendszer működését. A DBM függvények segítségével lényegében név-érték párokat tárolhatunk és kezelhetünk.

Noha ezek a függvények nem biztosítják számunkra egy SQL adatbázis erejét, rugalmasak és könnyen használhatók. Mivel a formátum igen elterjedt, az e függvényekre épülő kód általában hordozható, noha maguk a DBM adatokat tartalmazó állományok nem azok.

Ebben az órában a következő témákkal foglalkozunk:

- Megtanuljuk a DBM adatbázisok kezelését.
- Adatokkal töltünk fel egy adatbázist.
- Visszanyerjük adatainkat egy adatbázisból.
- Elemeket módosítunk.
- Megtanuljuk, hogyan tároljunk bonyolultabb adatokat egy DBM adatbázisban.

DBM adatbázis megnyitása

A DBM adatbázisokat a `dbmopen()` függvénnyel nyithatjuk meg, amelynek két paramétert kell átadnunk: a DBM fájl elérési útvonalát és a megnyitás módjára vonatkozó kapcsolókat. A függvény egy különleges DBM azonosítóval tér vissza, amelyet aztán a különböző egyéb DBM függvényekkel az adatbázis elérésére és módosítására használhatunk fel. Mivel a `dbmopen()` egy fájlt nyit meg írásra vagy olvasásra, a PHP-nek joga kell, hogy legyen az adatbázist tartalmazó könyvtár elérésére.

A `dbmopen()` függvénynek a 11.1. táblázatban felsorolt kapcsolókkal adhatjuk meg, milyen műveleteket kívánunk végrehajtani az adatbázison.

11.1. táblázat A `dbmopen()` kapcsolói

<i>Kapcsoló</i>	<i>Jelentés</i>
r	Az adatbázist csak olvasásra nyitja meg.
w	Az adatbázist írásra és olvasásra nyitja meg.
c	Létrehozza az adatbázist (ha létezik, akkor írásra/olvasásra nyitja meg).
n	Létrehozza az adatbázist (ha már létezik ilyen nevű, törli az előző változatot).

A következő kódrészlet megnyit egy adatbázist, ha pedig nem létezne a megadott néven, újat hoz létre:

```
$dbm = dbmopen( "./adat/termekek", "c" ) or
    ➔ die( "Nem lehet megnyitni a DBM adatbázist." );
```

Vegyük észre, hogy ha nem sikerülne az adatbázis megnyitása, a program futását a `die()` függvénnyel fejezzük be.

Ha befejeztük a munkát, zárjuk be az adatbázist a `dbmclose()` függvénnyel. Ez azért szükséges, mert a PHP automatikusan zárolja a megnyitott DBM adatbázist, hogy más folyamatok ne férhessenek hozzá a fájlhoz, mialatt a tartalmát olvassuk vagy írjuk. Ha az adatbázist nem zárjuk be, a várakozó folyamatok azután sem érhetik el az adatbázist, amikor már befejeztük a munkát. A `dbmclose()` függvény paramétere egy érvényes DBM azonosító:

```
dbmclose ( $dbm );
```


Adatok felvétele az adatbázisba

Új név-érték párt a `dbminsert()` függvénnyel vehetünk fel az adatbázisba. A függvénynek három paramétere van: egy érvényes DBM azonosító (amelyet a `dbmopen()` adott vissza), a kulcs és a tárolandó érték. A visszatérési érték 0, ha sikeres volt a művelet; 1, ha az elem már szerepel az adatbázisban; és -1 bármilyen más hiba esetén (például írási kísérlet egy csak olvasásra megnyitott adatbázisba). A `dbminsert()` már létező elemet nem ír felül.

A 11.1. programban létrehozuk és megnyitjuk a termékek nevű adatbázist és feltöltjük adatokkal.

11.1. program Adatok felvétele DBM adatbázisba.

```
1: <html>
2: <head>
3: <title>11.1. program Adatok felvétele DBM adatbázis-
   ba</title>
4: </head>
5: <body>
6: Termékek hozzáadása...
7:
8: <?php
9: $dbm = dbmopen( "../adat/termek", "c" ) or
   die( "Nem lehet megnyitni a DBM adatbázist." );
10:
11: dbminsert( $dbm, "Ultrahangos csavarhúzó", "23.20" );
12: dbminsert( $dbm, "Tricorder", "55.50" );
13: dbminsert( $dbm, "ORAC AI", "2200.50" );
14: dbminsert( $dbm, "HAL 2000", "4500.50" );
15:
16: dbmclose( $dbm );
17: ?>
18: </body>
19: </html>
```

Az adatbázisba illesztés során az összes érték karakterlánccá alakul, így a termékek árainak megadásakor idézőjeleket kell használnunk. Természetesen az adatbázisból kiolvasás után ezeket az értékeket lebegőpontos számokként is kezelhetjük, amennyiben szükséges. Vegyük észre, hogy nem csak egyszavas kulcsokat használhatunk.

Ha ezek után meghívjuk a `dbminsert()` függvényt egy olyan kulcsértékkel, amely már létezik az adatbázisban, a függvény az 1 értéket adja vissza és nem módosítja az adatbázist. Bizonyos körülmények között pontosan erre van szükség, de előfordulhat olyan eset is, amikor módosítani szeretnénk egy meglévő adatot, vagy ha nem található ilyen kulcs az adatbázisban, új adatot szeretnénk felvinni.

Adatok módosítása az adatbázisban

A DBM adatbázisban a bejegyzéseket a `dbmreplace()` függvénnyel módosíthatjuk. A függvény paraméterei: egy érvényes DBM azonosító, a kulcs neve és az új érték. A visszatérési érték a hibakód: 0, ha minden rendben volt és -1, ha valamilyen hiba lépett fel. A 11.2. példában az előző program egy új változata látható, amely a kulcsokat korábbi meglétüktől függetlenül felveszi az adatbázisba.

11.2. program Elemek felvétele vagy módosítása DBM adatbázisban

```
1: <html>
2: <head>
3: <title>11.2. program Elemek felvétele vagy módosítása
   DBM adatbázisban</title>
4: </head>
5: <body>
6: Termékek hozzáadása...
7: <?php
8: $dbm = dbmopen( "../adat/termekek", "c" )
9:         or die( "Nem lehet megnyitni a DMB adatbázis-
   zist." );
10: dbmreplace( $dbm, "Ultrahangos csavarhúzó", "25.20" );
11: dbmreplace( $dbm, "Tricorder", "56.50" );
12: dbmreplace( $dbm, "ORAC AI", "2209.50" );
13: dbmreplace( $dbm, "HAL 2000", "4535.50" );
14: dbmclose( $dbm );
15: ?>
16: </body>
17: </html>
```

A program működésének módosításához mindössze át kell írunk a `dbminsert()` függvényhívást `dbmreplace()`-re.

Adatok kiolvasása DBM adatbázisból

Egyetlen elemet a `dbmfetch()` függvény segítségével olvashatunk ki az adatbázisból. Ebben az esetben két paramétert kell átadnunk: egy létező DBM azonosítót és az elérni kívánt kulcs nevét. A függvény visszatérési értéke egy karakterlánc, a kulcshoz tartozó érték lesz. A "Tricorder" elem árát például a következő függvényhívással kérdezhetjük le:

```
$ar = dbmfetch( $dbm, "Tricorder" );
```

Ha "Tricorder" elem nem található az adatbázisban, a `dbmfetch()` egy üres karakterlánccal tér vissza.

Nem mindig ismerjük azonban az adatbázisban található kulcsokat. Mit tennénk például akkor, ha ki kellene írni a böngészőablakba az összes terméket és a hozzájuk tartozó árakat, anélkül, hogy „beleégetnénk” a programba a termékek nevét? A PHP biztosít egy módszert, amellyel az adatbázisban szereplő összes elemén végiglépkedhetünk.

Az adatbázis első kulcsát a `dbmfirstkey()` függvénnyel kérdezhetjük le. A függvény paramétere egy érvényes DBM azonosító, visszatérési értéke pedig a legelső kulcs. Természetesen ez nem feltétlenül egyezik meg az elsőként beillesztett adattal, ugyanis a DBM adatbáziskezelők gyakran saját rendezési eljárást használnak. Miután megkaptuk a legelső kulcsot, az összes rákövetkező elemet a `dbmnextkey()` függvény ismételt hívásával kérdezhetjük le. A függvény paraméterként szintén egy érvényes DBM azonosítót vár, visszatérési értéke pedig a következő kulcs a sorban. Ha ezeket a függvényeket együtt használjuk a `dbmfetch()`-cel, az adatbázis teljes tartalmát kiolvashatjuk.

A 11.3. példaprogram a termékek adatbázis teljes tartalmát kiírja a böngészőbe.

11.3. program DBM adatbázis összes bejegyzésének kiolvasása

```
1: <html>
2: <head>
3: <title>11.3. program DBM adatbázis összes bejegyzésének
4: kiolvasása</title>
5: </head>
6: <body>
7: A Hihetetlen Kütyük Boltja
8:   a következő izgalmas termékeket kínálja
9:   Önnek:
```

11.3. program (folytatás)

```

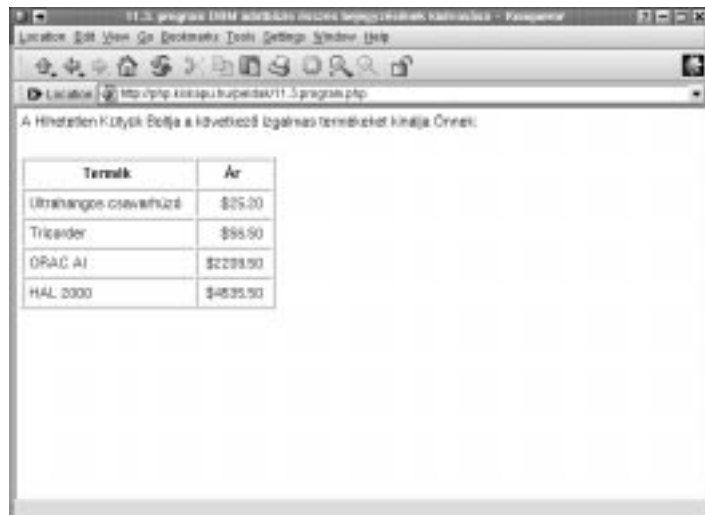
10: <p>
11: <table border="1" cellpadding="5">
12: <tr>
13: <td align="center"> <b>Termék</b> </td>
14: <td align="center"> <b>Ár</b> </td>
15: </tr>
16: <?php
17: $dbm = dbmopen( "../adat/termekek", "c" )
18:      or die( "Nem lehet megnyitni a DBM adatbázist."
              );
19: $kulcs = dbmfirstkey( $dbm );
20: while ( $kulcs != "" )
21:     {
22:         $ertek = dbmfetch( $dbm, $kulcs );
23:         print "<tr><td align = \"left\"> $kulcs </td>";
24:         print "<td align = \"right\"> \\$ertek
                </td></tr>";
25:         $kulcs = dbmnextkey( $dbm, $kulcs );
26:     }
27: dbmclose( $dbm );
28: ?>
29: </table>
30: </body>
31: </html>

```

A 11.1. ábrán a 11.3. program eredménye látható.

11.1. ábra

*A DBM adatbázis
összes bejegyzésének
lekérdezése.*



Termék	Ár
Útikalauz csavartúr	\$25.00
Tízper	\$66.90
ORAC AI	\$2288.90
HAL 2000	\$4835.90

Elemek meglétének lekérdezése

Mielőtt kiolvasnánk vagy módosítanánk egy elemet, hasznos lehet tudni, hogy létezik-e egyáltalán ilyen kulcsú elem az adatbázisban vagy sem. Erre a célra a `dbmexists()` függvény szolgál, amely paraméterként egy érvényes DBM azonosítót vár, illetve az ellenőrizendő elem nevét. A visszatérési érték `true`, ha az elem létezik.

```
if ( dbmexists( $dbm, "Tricorder" ) )  
    print dbmfetch( $dbm, "Tricorder" );
```

Elem törlése az adatbázisból

Az adatbázisból elemeket a `dbmdelete()` függvénnyel törölhetünk. A függvény bemenő paramétere egy érvényes DBM azonosító és a törlendő elem neve. Sikeres törlés esetén a visszatérési érték `true`, egyéb esetben (például ha az elem nem létezik) `false`.

```
dbmdelete( $dbm, "Tricorder" );
```

Összetett adatszerkezetek tárolása DBM adatbázisban

A DBM adatbázisban minden adat karaktersorozat formájában tárolódik, ezért az egész és lebegőpontos számokon, illetve karakterláncokon kívül minden egyéb adattípus elvész. Próbáljunk meg például egy tömböt tárolni:

```
$tomb = array( 1, 2, 3, 4 );  
$dbm = dbmopen( "./adat/proba", "c" ) or  
    ➤ die("Nem lehet megnyitni a DBM adatbázist.");  
dbminsert( $dbm, "tombproba", $tomb );  
print gettype( dbmfetch( $dbm, "tombproba" ) );  
// A kimenet: "string"
```

Itt létrehozunk egy tömböt és a `$tomb` változóba helyezzük. Ezután megnyitjuk az adatbázist és beszzúrjuk a tömböt `tombproba` néven, majd megvizsgáljuk a `dbmfetch()` függvény visszatérési típusát, amikor megpróbáljuk visszaolvasni a `tombproba` elemet – láthatjuk, hogy karakterláncot kaptunk vissza. Ha kiíratnánk volna a `tombproba` elem értékét, az `"Array"` karakterláncot kaptunk volna. Úgy látszik, ezzel el is úszott minden reményünk arra, hogy tömböket vagy más összetett adatszerkezetet tároljunk a DBM adatbázisban.

Szerencsére a PHP rendelkezik egy olyan lehetőséggel, amely segítségével a bonyolultabb szerkezeteket is egyszerű karaktersorozattá alakíthatjuk. Az így „kódolt” szerkezetet már tárolhatjuk későbbi használatra, DBM adatbázisban vagy akár fájlban is.

Az átalakítást a `serialize()` függvénnyel végezhetjük el, amelynek bemenő paramétere egy tetszőleges típusú változó, a visszaadott érték pedig egy karakterlánc:

```
$tomb = array( 1, 2, 3, 4 );
print serialize( $tomb );
// A kimenet: "a:4:{i:0;i:1;i:1;i:2;i:2;i:3;i:3;i:4;}"
```

A DBM adatbázisban ezt a karakterláncot tárolhatjuk. Ha vissza akarjuk állítani eredeti formájára, az `unserialize()` függvényt kell használnunk.

Ezzel a módszerrel lehetőségünk nyílik összetett adatszerkezetek tárolására a DBM adatbázisok által kínált egyszerű eszközökkel is. A 11.4. listában egy asszociatív tömb tartalmazza a termékekről rendelkezésre álló információkat – ezt alakítjuk át karakterláncná és helyezzük egy DBM adatbázisba.

11.4. program Összetett adatszerkezetek tárolása DBM adatbázisban

```
1: <html>
2: <head>
3: <title>11.4. program Összetett adatszerkezetek tárolása
   DBM adatbázisban</title>
4: </head>
5: <body>
6: Összetett adatok tárolása
7: <?php
8: $termekek = array(
9:     "Ultrahangos csavarhúzó" => array( "ar"=>"22.50",
10:         "szallitas"=>"12.50",
11:         "szin"=>"zöld" ),
12:     "Tricorder"
13:         => array( "ar"=>"55.50",
14:         "szallitas"=>"7.50",
15:         "szin"=>"vörös" ),
16:     "ORAC AI"
17:         => array( "ar"=>"2200.50",
18:         "szallitas"=>"34.50",
19:         "szin"=>"kék" ),
20:     "HAL 2000"
21:         => array( "ar"=>"4500.50",
22:         "szallitas"=>"18.50",
23:         "szin"=>"rózsaszín" )
24: );
```

11.4. program (folytatás)

```

22: $dbm = dbmopen( "./adat/ujtermek", "c" )
23:         or die("Nem lehet megnyitni a DBM adatbázist.");
24: foreach( $termek as $kulcs => $ertek )
25:         dbmreplace( $dbm, $kulcs, serialize( $ertek ) );
26: dbmclose( $dbm );
27: ?>
28: </body>
29: </html>

```

A listában egy többdimenziós tömböt építünk fel, amely kulcsként tartalmazza a termék nevét és három tömbelembe tárolja a színt, az árat és a szállítás költségét. Ezután egy ciklus segítségével feldolgozzuk az összes elemet: a termék nevét és a karakterláncból alakított tömböt átadjuk a `dbmreplace()` függvénynek. Ezután lezárjuk az adatbázist.

A 11.5. program az adatok visszatöltésére ad megoldást.

11.5. program Összetett adatszerkezetek visszaolvasása DBM adatbázisból

```

1: <html>
2: <head>
3: <title>11.5. program Összetett adatszerkezetek
   visszaolvasása
4:         DBM adatbázisból</title>
5: </head>
6: <body>
7: A Hihetetlen Kütyük Boltja
8:   a következő izgalmas termékeket kínálja
9:   Önnek:
10: <p>
11: <table border="1" cellpadding="5">
12: <tr>
13: <td align="center"> <b>Termék</b> </td>
14: <td align="center"> <b>Szín</b> </td>
15: <td align="center"> <b>Szállítási</b> </td>
16: <td align="center"> <b>Ár</b> </td>
17: </tr>
18: <?php

```

11.5. program (folytatás)

```

19: $dbm = dbmopen( "./adat/ujtermekek", "c" )
20:         or die("Nem lehet megnyitni
                a DBM adatbázist.");
21: $kulcs = dbmfirstkey( $dbm );
22: while ( $kulcs != "" )
23:     {
24:         $termektomb = unserialize( dbmfetch( $dbm,
                $kulcs ) );
25:         print "<tr><td align=\"left\"> $kulcs </td>";
26:         print '<td align="left" $termektomb["szin"]
                "</td>";
27:         print '<td align="right"$
                $termektomb["szallitas"] "</td>";
28:         print '<td align="right"$ $termektomb["ar"]
                "</td></tr>\n";
29:         $kulcs = dbmnextkey( $dbm, $kulcs );
30:     }
31: dbmclose( $dbm );
32: ?>
33: </table>
34: </body>
35: </html>

```

Ez a megoldás hasonló a 11.3. példában látottakhoz, de ebben az esetben több mezőt nyomtatunk ki. Megnyitjuk az adatbázist és a `dbmfirstkey()`, illetve a `dbmnextkey()` függvénnyel beolvassuk az összes adatot az adatbázisból. A ciklusban az `unserialize()` függvénnyel létrehozuk a termékeket tartalmazó tömböt. Így már egyszerű feladat kiírni az összes elemet a böngészőablakba. A 11.2. ábrán a 11.5. program kimenete látható.

11.2. ábra

Összetett adatszerkezetek visszaolvasása DBM adatbázisból



Termék	Szín	Szállítás	Ár
Utazáshoz szükséges	zöld	\$12.00	\$22.00
Tiszta víz	vörös	\$1.00	\$0.00
OPAC AI	fehér	\$3450	\$1138.00
HAL DORG	vöröses	\$1050	\$408.00

Egy példa

Már eleget tudunk ahhoz, hogy elkészítsünk egy működőképes programot az ebben az órában tanult módszerekkel. A feladat a következő: készítsünk egy karbantartó oldalt, ahol a webhely szerkesztője megváltoztathatja a 11.2. példa-program által létrehozott adatbázis termékeinek árát. Tegyük lehetővé a rendszer-gazda számára, hogy új elemekkel bővítse az adatbázist, illetve hogy a régi elemeket törölje. Az oldalt nem tesszük nyilvános kiszolgálón elérhetővé, ezért a biztonsági kérdésekkel most nem foglalkozunk.

Először is fel kell építenünk az űrlapot, amely az adatbázis elemeit tartalmazza. A termékek árának módosításához szükségünk lesz egy szövegmezőre és minden elemhez tartozni fog egy jelölőnégyzet is, melynek segítségével az adott elemet törlésre jelölhetjük ki. A lapon el kell helyeznünk két további szövegmezőt is, az új elemek felvételéhez. A 11.6. példában az oldalt elkészítő program olvasható.

11

11.6. program HTML űrlap készítése DBM adatbázis alapján

```
1: <?
2: $dbm = dbmopen( "../adat/termekek", "c" )
3:         or die("Nem lehet megnyitni
           a DBM adatbázist.");
4: ?>
5: <html>
6: <head>
7: <title>11.6. program HTML űrlap készítése
8: DBM adatbázis alapján </title>
9: </head>
10: <body>
11: <form method="POST">
12: <table border="1">
13: <tr>
14: <td>Törlés</td>
15: <td>Termék</td>
16: <td>Ár</td>
17: </tr>
18: <?php
19: $kulcs = dbmfirstkey( $dbm );
20: while ( $kulcs != "" )
21:     {
22:         $ar = dbmfetch( $dbm, $kulcs );
23:         print "<tr><td> <input type=\"checkbox\" \
                name=\"torles[]\" ";
```

11.6. program (folytatás)

```

24:     print "value=\"\$kulcs\"> </td>";
25:     print "<td> \$kulcs </td>";
26:     print "<td> <input type=\"text\"
           name=\"arak[\$kulcs]\" ";
27:     print "value=\"\$ar\"> </td></tr>";
28:     \$kulcs = dbmnextkey( \$dbm, \$kulcs );
29:     }
30: dbmclose( \$dbm );
31: ?>
32: <tr>
33: <td>&nbsp;</td>
34: <td><input type="text" name="uj_nev"></td>
35: <td><input type="text" name="uj_ar"></td>
36: </tr>
37: <tr>
38: <td colspan="3" align="right">
39: <input type="submit" value="Változtat">
40: </td>
41: </tr>
42: </table>
43: </form>
44: </body>
45: </html>

```

Szokás szerint az első lépés az adatbázis megnyitása. Ezután megkezdünk egy űrlapot és mivel nem adunk meg céloldalt, feldolgozó programként magát az oldalt jelöljük ki.

Elkészítjük a táblázat fejlécét, majd végigolvassuk az adatbázist a `dbmfirstkey()` és `dbmnextkey()` függvények segítségével, az értékeket a `dbmfetch()` függvénnyel olvasva ki.

A táblázat első mezője minden sorban egy jelölőnégyzetet tartalmaz. Vegyük észre, hogy mindegyik jelölőnégyzet neve `"torles[]"`. Ennek hatására a PHP létrehozza a `$torles` tömböt, amely az összes bejelölt elemet tartalmazza. A jelölőnégyzetekhez tartozó értékek, így a `$torles` tömb elemei is azok az azonosítók lesznek, amelyekkel a DBM adatbázis az adott terméket nyilvántartja (ezt az értéket a `$kulcs` változóban tároljuk). Így, miután a rendszergazda kitöltötte és elküldte az űrlapot, a program `$torles` tömbje azon adatbázis-elemek kulcsait fogja tárolni, amelyeket törölnünk kell.

Ezután kiírjuk a böngészőablakba az elem nevét és létrehozunk egy szöveges mezőt a termék árának. A mezőt ugyanolyan módon nevezzük el, mint az előzőt, ezúttal azonban a szögletes zárójelek közé beírjuk az azonosítót, amely alapján a DBM adatbázis az elemet tárolja. A PHP ennek hatására létrehozza az \$sarak tömböt, amelyben a kulcsok a termékek azonosítói.

Lezárjuk az adatbázist és visszatérünk HTML módba az új bejegyzés létrehozására szolgáló uj_nev és uj_ar mezők megjelenítéséhez. A 11.3. ábrán a 11.6. program kimenetét láthatjuk.

11.3. ábra

HTML űrlap készítése
DBM adatbázis
alapján

Törles Termék	Ar
Ultrasongas esavérháló	25.20
Tricorder	56.50
ORAC AI	2209.50
HAL 2000	4535.50

VÁROZOK

Miután elkészítettük az űrlapot, meg kell írunk a kódot, amely a felhasználó által megadott adatokat kezeli. A feladat nem olyan nehéz, mint amilyennek látszik. Először töröljük a kijelölt elemeket az adatbázisból, majd módosítjuk az árakat, végül felvesszük az új elemet az adatbázisba.

Miután a karbantartó kitöltötte és elküldte az űrlapot, a törlendő elemek listája a \$storles tömbben áll rendelkezésünkre. Mindössze annyi a dolgunk, hogy végigolgvassuk a tömböt, és az összes elemét töröljük az adatbázisból.

```
if ( isset ( $storles ) )
{
    foreach ( $storles as $kulcs => $sertek )
    {
        unset( $sarak[$sertek]);
        dbmdelete( $dbm, $sertek );
    }
}
```

Először is megvizsgáljuk, hogy létezik-e a `$torles` változó. Ha a felhasználó csak most érkezett az oldalra vagy nem jelölt ki törlése egyetlen terméket sem, a változó nem létezik. Ellenkező esetben egy ciklus segítségével végigolvassuk és minden elemére meghívjuk a `dbmdelete()` függvényt, amely eltávolítja a DBM adatbázisból a paraméterként megadott elemet. Hasonlóan járunk el az `$arak` tömb esetében is, itt azonban a PHP `unset()` függvényét kell használnunk a tömbelem törlésére. Az `$arak` tömb a felhasználótól érkezett, fellehetően részben módosított árakat tartalmazza. Ha nem törölnénk az `$arak` tömbből a megfelelő elemet, a következő kód újra beillesztené.

Az adatbázis elemeinek frissítése során két választási lehetőségünk van. Az első változatot akkor alkalmazzuk, ha az adatbázis karbantartását nem egyetlen szerkesztő fogja végezni, tehát feltételezhető, hogy a programot egyidőben több felhasználó is futtatni fogja – eszerint csak azokat az elemeket változtatjuk meg, amelyeket a felhasználó kijelölt. A másik változat, amely az összes elemet megváltoztatja, akkor alkalmazható, ha a program csak egyetlen példányban futhat:

```
if ( isset ( $arak ) )
{
    foreach ( $arak as $kulcs => $ertek )
        dbmreplace( $dbm, $kulcs, $ertek );
}
```

Először is ellenőriznünk kell az `$arak` tömb meglétét. Ebben a tömbben az adatbázis egy teljesen új változata lehet. Egy ciklus segítségével tehát végigolvassuk az összes elemet és egyesével frissítjük az adatbázisban.

Végül ellenőriznünk kell, hogy a felhasználó kezdeményezte-e új elem felvételét az adatbázisba:

```
if ( ! empty( $uj_nev ) && ! empty( $uj_ar ) )
    dbminsert( $dbm, "$uj_nev", "$uj_ar" );
```

Ahelyett, hogy az `$uj_nev` és `$uj_ar` meglétét ellenőriznénk, azt kell ellenőriznünk, hogy értékük nem „üres”-e. Ez apró, de lényeges különbség. Amikor a felhasználó elküldi az űrlapot, a változók mindenképpen létrejönnek, de ha a szövegmezők nem kaptak értéket, a megfelelő változók üres karakterláncot tartalmaznak. Mivel nem akarunk üres elemeket tárolni az adatbázisban, fontos ellenőrizni, hogy nem üresek-e a változók. Azért használjuk a `dbminsert()` függvényt a `dbmreplace()` helyett, hogy elkerüljük a már bevitt termékek véletlen felülírását.

A teljes kódot a 11.7. példa tartalmazza.

11.7. program A teljes adatbázis-karbantartó program

```
1: <?php
2: $dbm = dbmopen( "../adat/termekek", "c" )
3:         or die("Nem lehet megnyitni
           a DBM adatbázist.");
4:
5: if ( isset ( $storles ) )
6:     {
7:         foreach ( $storles as $kulcs => $ertek )
8:             {
9:                 unset( $arak[$ertek]);
10:                dbmdelete( $dbm, $ertek );
11:            }
12:     }
13:
14: if ( isset ( $arak ) )
15:     {
16:         foreach ( $arak as $kulcs => $ertek )
17:             dbmreplace( $dbm, $kulcs, $ertek );
18:     }
19:
20: if ( ! empty( $uj_nev ) && ! empty( $uj_ar ) )
21:     dbminsert( $dbm, "$uj_nev", "$uj_ar" );
22: ?>
23:
24: <html>
25: <head>
26: <title>11.7. program A teljes adatbázis-karbantartó
           program </title>
27: </head>
28: <body>
29:
30: <form method="POST">
31:
32: <table border="1">
33: <tr>
34: <td>Törlés</td>
35: <td>Termék</td>
36: <td>Ár</td>
37: </tr>
38:
39: <?php
```

11.7. program (folytatás)

```
40: $kulcs = dbmfirstkey( $dbm );
41: while ( $kulcs != "" )
42:     {
43:         $ar = dbmfetch( $dbm, $kulcs );
44:         print "<tr><td> <input type=\"checkbox\" \
            name=\"torles[]\" \" ";
45:         print "value=\"$kulcs\"> </td>";
46:         print "<td> $kulcs </td>";
47:         print "<td> <input type=\"text\" \
            name=\"arak[$kulcs]\" \" ";
48:         print "value=\"$ar\"> </td></tr>";
49:         $kulcs = dbmnextkey( $dbm, $kulcs );
50:     }
51:
52: dbmclose( $dbm );
53: ?>
54:
55: <tr>
56: <td>&nbsp;  </td>
57: <td><input type="text" name="uj_nev"></td>
58: <td><input type="text" name="uj_ar"></td>
59: </tr>
60:
61: <tr>
62: <td colspan="3" align="right">
63: <input type="submit" value="Változtat">
64: </td>
65: </tr>
66:
67: </table>
68: </form>
69:
70: </body>
71: </html>
```

Összefoglalás

Ebben az órában megtanultuk, hogyan használjuk a PHP hatékony DBM függvényeit adatok tárolására és visszaolvasására. Megtanultuk a `dbmopen()` használatát egy új DBM azonosító létrehozására. Ezt az azonosítót használtuk az összes többi DBM függvényénél is. Új adatokat adtunk az adatbázishoz a `dbminsert()` függvénnyel, módosítottunk meglevőket a `dbmreplace()`-szel és töröltünk a `dbmdelete()` használatával. Megtanultuk, hogyan használhatjuk a `dbmfetch()` függvényt az adatok visszaolvasására. A `serialize()` és `unserialize()` függvényekkel összetett adatszerkezeteket tároltunk DBM adatbázisban, végül egy gyakorlati példán keresztül megnéztük, hogyan is használhatók fel e módszerek a valós problémák megoldására.

11

Kérdések és válaszok

Mikor használjak DBM adatbázist SQL adatbázis helyett?

A DBM jó választás, ha kis mennyiségű és viszonylag egyszerű szerkezetű adatot szeretnénk tárolni (név-érték párokat). A DBM adatbázist használó programok rendelkeznek a hordozhatóság nagyon fontos előnyével, de ha nagyobb mennyiségű adatot kell tárolnunk, válasszunk inkább egy SQL adatbáziskezelőt, például a MySQL-t.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvényt használhatjuk egy DBM adatbázis megnyitásához?
2. Melyik függvénnyel szúrhatunk be új elemet egy DBM adatbázisba?
3. Melyik függvénnyel módosíthatunk egy elemet?
4. Hogyan érünk el egy elemet az adatbázisban, ha ismerjük a nevét?
5. Hogyan olvasnánk ki egy DBM adatbázisból a legelső elem nevét (nem az értékét)?
6. Hogyan érjük el a további neveket?
7. Hogyan törölünk egy elemet a DBM adatbázisból, ha ismerjük a nevét?

Feladatok

- 1 Hozzunk létre egy DBM adatbázist a felhasználók azonosítóinak és jelszavainak tárolására. Készítsünk egy programot, amellyel a felhasználók létrehozhatják a rájuk vonatkozó bejegyzést. Ne feledjük, hogy két azonos nevű elem nem kerülhet az adatbázisba.
- 2 Készítsünk egy bejelentkező programot, amely ellenőrzi a felhasználó azonosítóját és jelszavát. Ha a felhasználói bemenet egyezik valamelyik bejegyzéssel az adatbázisban, akkor üdvözljük a felhasználót valamilyen különleges üzenettel. Egyébként jelenítsük meg újra a bejelentkező űrlapot.



12. ÓRA

Adatbázisok kezelése – MySQL

A PHP nyelv egyik meghatározó tulajdonsága, hogy nagyon könnyen képes adatbázisokhoz csatlakozni és azokat kezelni. Ebben az órában elsősorban a MySQL adatbázisokkal foglalkozunk, de amint majd észre fogjuk venni, a PHP által támogatott összes adatbázist hasonló függvények segítségével érhetjük el. Miért esett a választás éppen a MySQL-re? Mert ingyenes, ugyanakkor nagyon hatékony eszköz, amely képes megfelelni a valós feladatok által támasztott igényeknek is. Az sem elhanyagolható szempont, hogy többféle rendszerhez is elérhető. A MySQL adatbáziskiszolgálót a `http://www.mysql.com` címről tölthetjük le. Ebben az órában a következő témákkal foglalkozunk:

- Megnézünk néhány SQL példát.
- Csatlakozunk egy MySQL adatbáziskiszolgálóhoz.
- Kiválasztunk egy adatbázist.
- Tanulunk a hibakezelésről.
- Adatokat viszünk fel egy táblába.

- Adatokat nyerünk ki egy táblából.
- Megváltoztatjuk egy adattábla tartalmát.
- Megjelenítjük az adatbázis szerkezetét.

(Nagyon) rövid bevezetés az SQL nyelvbe

ÚJDONSÁG

Az SQL jelentése *Structured Query Language*, vagyis strukturált lekérdező nyelv. Az SQL szabványosított nyelvezete segítségével a különböző típusú adatbázisokat azonos módon kezelhetjük. A legtöbb SQL termék saját bővítésekkel látja el a nyelvet, ahogy a legtöbb böngésző is saját HTML nyelvjárást „beszél”. Mindazonáltal SQL ismeretek birtokában nagyon sokféle adatbázist fogunk tudni használni, a legkülönbözőbb operációs rendszereken.

A könyvben – terjedelmi okok miatt – még bevezető szinten sem tárgyalhatjuk az SQL-t, de megpróbálunk megvilágítani egy-két dolgot a MySQL-lel és általában az SQL-lel kapcsolatban.

A MySQL kiszolgáló démonként fut a számítógépen, így a helyi vagy akár távoli gépek felhasználói bármikor csatlakozhatnak hozzá. Miután a csatlakozás megtörtént, ki kell választanunk a megfelelő adatbázist, ha van jogunk hozzá.

Egy adatbázison belül több adattáblánk is lehet. Minden tábla oszlopokból és sorokból áll. A sorok és oszlopok metszéspontjában tároljuk az adatokat. Minden oszlopba csak előre megadott típusú adatot tölthetünk, az INT típus például egész számot, míg a VARCHAR változó hosszúságú, de egy adott értéknél nem hosszabb karakterláncot jelent.

A kiválasztott adatbázisban a következő SQL utasítással hozhatunk létre új táblát:

```
CREATE TABLE entablam ( keresztnev VARCHAR(30),  
➡ vezeteknev VARCHAR(30), kor INT );
```

Ez a tábla három oszlopot tartalmaz. A keresztnev és vezeteknev oszlopokba legfeljebb 30 karaktert írhatunk, míg a kor oszlopba egy egész számot.

A táblába új sort az INSERT paranccsal vehetünk fel:

```
INSERT INTO entablam ( keresztnev, vezeteknev, kor )  
➡ VALUES ( 'János', 'Kovács', 36 );
```

A mezőneveket az első, zárójelek közti kifejezéssel adjuk meg, míg az értékeket a megfelelő sorrendben a második zárójelpár között soroljuk fel.

A táblából az összes adatot a `SELECT` paranccsal kaphatjuk meg:

```
SELECT * FROM entablam;
```

A „*” szokásos helyettesítő karakter, jelentése „az összes mező”. Ha nem az összes mező tartalmát akarjuk lekérdezni, írjuk be az érintett mezők neveit a csillag helyére:

```
SELECT kor, keresztnév FROM entablam;
```

Már létező bejegyzést az `UPDATE` paranccsal módosíthatunk.

```
UPDATE entablam SET keresztnév = 'Gábor';
```

Az utasítás az összes sorban "Gábor"-ra módosítja a keresztnév mező tartalmát. A `WHERE` záradékkal leszűkíthetjük a `SELECT` és `UPDATE` parancsok hatáskörét. Például:

```
SELECT * FROM entablam WHERE keresztnév = 'Gábor';
```

Ez az utasítás csak azokat a sorokat írja ki, amelyekben a keresztnév mező értéke "Gábor".

A következő példa csak azokban a sorokban változtatja "Gábor"-ra a keresztnév mező értékét, ahol a vezetéknév mező a "Szakács" karakterláncot tartalmazza.

```
UPDATE entablam SET keresztnév = "Gábor" WHERE vezetéknév =  
➡ = "Szakács";
```

Az SQL-ről további információkat találunk Ryan K. Stephens és szerzőtársai *Teach Yourself SQL in 21 Days* című könyvében.

Csatlakozás a kiszolgálóhoz

Mielőtt elkezdhetnénk dolgozni az adatbázissal, csatlakoznunk kell a kiszolgálóhoz. A PHP-ben erre a `mysql_connect()` függvény szolgál. A függvény három karakterláncot vár paraméterként: a gazdagép nevét, a felhasználó nevét és a jelszót. Ha ezek egyikét sem adjuk meg, a függvény feltételezi, hogy a kérés a `localhost`-ra (azaz a helyi gépre) vonatkozik és felhasználóként a PHP-t futtató felhasználót, jelszóként pedig egy üres karakterláncot ad át. Az alapértelmezés

a `php.ini` fájlban felülbírálnak, de egy próbálkozásra szánt kiszolgálót kivéve nem bölcs dolog ezzel próbálkozni, ezért a példákban mindig használni fogjuk a felhasználónevet és a jelszót. A `mysql_connect()` függvény siker esetén egy kapcsolatazonosítót ad vissza, amelyet egy változóba mentünk, hogy a későbbiekben folytathassuk a munkát az adatbáziskiszolgálóval.

Az alábbi kódrészlet a `mysql_connect()` függvény segítségével kapcsolódik a MySQL adatbáziskiszolgálóhoz.

```
$kapcsolat = mysql_connect( "localhost", "root", "jelszo" );  
if ( ! $kapcsolat )  
    die( "Nem lehet csatlakozni a MySQL kiszolgálóhoz!" );
```

Ha a PHP-t az Apache kiszolgáló moduljaként használjuk, a `mysql_pconnect()` függvényt is használhatjuk az előzőekben megadott paraméterekkel. Fontos különbség a két függvény között, hogy a `mysql_pconnect()`-tel megnyitott adatbáziskapcsolat nem szűnik meg a PHP program lefutásával vagy a `mysql_close()` függvény hatására (amely egy szokásos MySQL kiszolgálókapcsolat bontására szolgál), hanem továbbra is aktív marad és olyan programokra várakozik, amelyek a `mysql_pconnect()` függvényt hívják. Más szóval a `mysql_pconnect()` függvény használatával megtakaríthatjuk azt az időt, ami egy kapcsolat felépítéséhez szükséges és egy előzőleg lefutott program által hagyott azonosítót használhatunk.

Az adatbázis kiválasztása

Miután kialakítottuk a kapcsolatot a MySQL démonnal, ki kell választanunk, melyik adatbázissal szeretnénk dolgozni. Erre a célra a `mysql_select_db()` függvény szolgál, amelynek meg kell adnunk a kiválasztott adatbázis nevét és szükség szerint egy kapcsolatazonosító értéket. Ha ez utóbbit elhagyjuk, automatikusan a legutoljára létrehozott kapcsolat helyettesítődik be. A `mysql_select_db()` függvény igaz értéket ad vissza, ha az adatbázis létezik és jogunk van a használatára. A következő kódrészlet a `pelda` nevű adatbázist választja ki.

```
$adatbazis = "pelda";  
mysql_select_db( $adatbazis ) or die ( "Nem lehet  
➡ megnyitni a következő adatbázist: $adatbazis" );
```

Hibakezelés

Eddig ellenőriztük a MySQL függvények visszatérési értékét és hiba esetén a `die()` függvénnyel kiléptünk a programból. A hibakezeléshez azonban hasznosabb lenne, ha a hibaüzenetek valamivel több információt tartalmaznának. Ha valamilyen művelet nem sikerül, a MySQL beállít egy hibakódot és egy hibaüzenetet. A hibakódhoz a `mysql_errno()`, míg a hibaüzenethez a `mysql_error()` függvénnyel férhetünk hozzá. A 12.1 példa az eddigi kódrészleteinket teljes programmá kapcsolja össze, amely kapcsolódik az adatbáziskezelőhöz és kiválasztja az adatbázist. A hibaüzenetet a `mysql_error()` függvénnyel tesszük használhatóbbá.

12.1. program Kapcsolat megnyitása és az adatbázis kiválasztása

```
1: <html>
2: <head>
3: <title>12.1. program Kapcsolat megnyitása és
4: adatbázis kiválasztása</title>
5: </head>
6: <body>
7: <?php
8: $felhasznalo = "jozsi";
9: $jelszo = "bubosvocsok";
10: $adatbazis = "pelda";
11: $kapcsolat = mysql_connect( "localhost",
12:                             $felhasznalo, $jelszo );
12: if ( ! $kapcsolat )
13:     die( "Nem lehet kapcsolódni
14:         a MySQL kiszolgálóhoz!" );
14: print "Sikerült a kapcsolatfelvétel<P>";
15: mysql_select_db( $adatbazis )
16:     or die ( "Nem lehet megnyitni a $adatbázist: "
17:             .mysql_error() );
17: print "Sikeresen kiválasztott adatbázis: \"
18:         $adatbazis\"<P>";
18: mysql_close( $kapcsolat );
19: ?>
20: </body>
21: </html>
```

Ha az \$adatbazis változó értékét mondjuk "nincsmeg"-re módosítjuk, a program egy nem létező adatbázist próbál meg elérni. Ekkor a die() kimenete valami ilyesmi lesz:

```
Nem lehet megnyitni a nincsmeg adatbázist: Access denied
➔ for user: 'jozsi@localhost' to database 'nincsmeg'
```

Adatok hozzáadása táblához

Már sikerült hozzáférést szerezni az adatbázishoz, így ideje némi adatot is bevinni a táblákba. A következő példákhoz képzeljük el, hogy az általunk készítendő oldalon a látogatóknak lehetőségük lesz tartományneveket vásárolni.

A példa adatbázisban készítsük el az öt oszlopot tartalmazó tartományok táblát. Az azonosito mező lesz az elsődleges kulcs, amely automatikusan növel egy egész értéket, ahogy újabb bejegyzések kerülnek a táblába, a tartomany mező változó számú karaktert tartalmazhat (VARCHAR), a nem mező egyetlen karaktert, az email mező pedig a felhasználó elektronikus levélcímét. A táblát a következő SQL paranccsal készíthetjük el:

```
CREATE TABLE tartomanyok ( azonosito INT NOT NULL
➔ AUTO_INCREMENT,
                           PRIMARY KEY( azonosito ),
                           tartomany VARCHAR( 20 ),
                           nem CHAR( 1 ),
                           email VARCHAR( 20 ) );
```

Az adatok felviteléhez össze kell állítanunk és le kell futtatnunk egy SQL parancsot, erre a célra a PHP-ben a mysql_query() függvény szolgál. A függvény paraméterként egy SQL parancsot tartalmazó karakterláncot és szükség szerint egy kapcsolazonosítót vár. Ha ez utóbbit elhagyjuk, a függvény automatikusan az utoljára megnyitott kapcsolaton keresztül próbálja meg kiadni az SQL parancsot. Ha a program sikeresen lefutott, a mysql_query() pozitív értéket ad vissza, ha azonban formai hibát tartalmaz vagy nincs jogunk elérni a kívánt adatbázist, a visszatérési érték hamis lesz. Meg kell jegyeznünk, hogy egy sikeresen lefutott SQL program nem feltétlenül okoz változást az adatbázisban vagy tér vissza valamilyen eredménnyel. A 12.2 példában kibővítjük a korábbi programot és a mysql_query() függvénnyel kiadunk egy INSERT utasítást a példa adatbázis tartományok tábláján.

12.2. program Új sor hozzáadása táblához

```
1: <html>
2: <head>
3: <title>12.2. program Új sor hozzáadása
   táblához</title>
4: </head>
5: <body>
6: <?php
7: $felhasznalo = "jozsi";
8: $jelszo = "bubosvocskok";
9: $adatbazis = "pelda";
10: $kapcsolat = mysql_connect( "localhost",
   $felhasznalo, $jelszo );
11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
   a MySQL kiszolgálóhoz!" );
13: mysql_select_db( $adatbazis, $kapcsolat )
14:     or die ( "Nem lehet megnyitni a $adatbázist:
   ".mysql_error() );
15: $parancs = "INSERT INTO tartomanyok ( tartomany,
   nem, email )
16:     VALUES ( '123xyz.com', 'F',
   'okoska@tartomany.hu' )";
17: mysql_query( $parancs, $kapcsolat )
18: or die ( "Nem lehet adatot hozzáadni
   a \"tartomanyok\" táblához: "
19: .mysql_error() );
20: mysql_close( $kapcsolat );
21: ?>
22: </body>
23: </html>
```

Vegyük észre, hogy nem adtunk értéket az azonosító mezőnek. A mező értéke automatikusan növekszik az INSERT hatására.

Természetesen minden esetben, amikor a böngészőben újratöltjük a 12.2 példa-programot, ugyanaz a sor adódik hozzá a táblához. A 12.3 példa a felhasználó által bevitt adatot tölti be a táblába.

12.3.program A felhasználó által megadott adatok beszúrása a táblába

```
1: <html>
2: <head>
3: <title>12.3. program A felhasználó által megadott ada-
   tok beszúrása a táblába</title>
4: </head>
5: <body>
6: <?php
7: if ( isset( $startomany ) && isset( $nem ) &&
   isset( $email ) )
8:     {
9:         // Ne feledjük ellenőrizni a felhasználó által
   megadott adatokat!
10:        $dbhiba = "";
11:        $vissza = adatbazis_bovit( $startomany, $nem,
   $email, $dbhiba );
12:        if ( ! $vissza )
13:            print "Hiba: $dbhiba<BR>";
14:        else
15:            print "Köszönjük!";
16:        }
17:    else    {
18:        urlap_keszit();
19:    }
20:
21: function adatbazis_bovit( $startomany, $nem, $email,
   &$dbhiba )
22:     {
23:        $felhasznalo = "jozsi";
24:        $jelszo = "bubosvocsok";
25:        $adatbazis = "pelda";
26:        $kapcsolat = mysql_pconnect( "localhost",
   $felhasznalo, $jelszo );
27:        if ( ! $kapcsolat )
28:            {
29:                $dbhiba = "Nem lehet kapcsolódni
   a MySQL kiszolgálóhoz!";
30:                return false;
31:            }
32:        if ( ! mysql_select_db( $adatbazis, $kapcsolat ) )
33:            {
34:                $dbhiba = mysql_error();
```


12.3.program (folytatás)

```
35:         return false;
36:     }
37:     $parancs = "INSERT INTO tartomanyok ( tartomany,
        nem, email )
38:     VALUES ( '$tartomany', '$nem', '$email' )";
39:     if ( ! mysql_query( $parancs, $kapcsolat ) )
40:     {
41:         $dbhiba = mysql_error();
42:         return false;
43:     }
44:     return true;
45: }
46:
47: function urlap_keszit()
48: {
49:     global $PHP_SELF;
50:     print "<form action=\".$PHP_SELF\"
        method=\"POST\">\n";
51:     print "A kívánt tartomány<p>\n";
52:     print "<input type=\"text\" name=\"tartomany\"> ";
53:     print "Email cím<p>\n";
54:     print "<input type=\"text\" name=\"email\"> ";
55:     print "<select name=\"nem\">\n";
56:     print "\t<option value=\"N\"> Nő\n";
57:     print "\t<option value=\"F\"> Férfi\n";
58:     print "</select>\n";
59:     print "<input type=\"submit\"
        value=\"Elküld\">\n</form>\n";
60: }
61: ?>
62: </body>
63: </html>
```

A tömörség érdekében a 12.3. példából kihagytunk egy fontos részt. Megbízunk a felhasználókban: semmilyen formában nem ellenőriztük a felhasználó által bevitt adatokat. Ezt a feladatot a 17. órában tárgyalt karakterlánckezelő függvényekkel végezhetjük el. Meg kell jegyeznünk, hogy a beérkező adatok ellenőrzése mindig fontos feladat, itt csak azért maradt el, mivel így jobban összpontosíthattunk az óra témájára.

Ellenőrizzük a `$startomany`, `$nem` és `$email` változókat. Ha megvannak, nyugodtan feltehetjük, hogy a felhasználó adatokat küldött az űrlap kitöltésével, így meghívjuk az `adattbazis_bovit()` függvényt.

Az `adattbazis_bovit()`-nek négy paramétere van: a `$startomany`, a `$nem` és az `$email` változók, illetve a `$dbhiba` karakterlánc. Ez utóbbiba fogjuk betölteni az esetlegesen felbukkanó hibaüzeneteket, így hivatkozásként kell átadnunk. Ennek hatására ha a függvénytörzsön belül megváltoztatjuk a `$dbhiba` értékét, a másolat helyett tulajdonképpen az eredeti paraméter értékét módosítjuk.

Először megkísérlünk megnyitni egy kapcsolatot a MySQL kiszolgálóhoz. Ha ez nem sikerül, akkor hozzárendelünk egy hibaszöveget a `$dbhiba` változóhoz és `false` értéket visszaadva befejezzük a függvény futását. Ha a csatlakozás sikeres volt, kiválasztjuk a `tartomany` táblát tartalmazó adatbázist és összeállítunk egy SQL utasítást a felhasználó által küldött értékek felhasználásával, majd az utasítást átadjuk a `mysql_query()` függvénynek, amely elvégzi az adatbázisban a kívánt műveletet. Ha a `mysql_select_db()` vagy a `mysql_query()` függvény nem sikeres, a `$dbhiba` változóba betöltjük a `mysql_error()` függvény által visszaadott értéket és `false` értékkel térünk vissza. Minden egyéb esetben, vagyis ha a művelet sikeres volt, `true` értéket adunk vissza.

Miután az `adattbazis_bovit()` lefutott, megvizsgáljuk a visszatérési értéket. Ha `true`, az adatok bekerültek az adatbázisba, így üzenetet küldhetünk a felhasználónak. Egyébként ki kell írunk a böngészőbe a hibaüzenetet. Az `adattbazis_bovit()` függvény által visszaadott `$dbhiba` változó most már hasznos adatokat tartalmaz a hiba természetét illetően, így ezt a szöveget is belefoglaljuk a hibaüzenetbe.

Ha a kezdeti `if` kifejezés nem találja meg a `$startomany`, `$nem` vagy `$email` valamelyikét, feltehetjük, hogy a felhasználó nem küldött el semmilyen adatot, ezért meghívunk egy másik általunk írt függvényt, az `urlap_keszit()`-et, amely egy HTML űrlapot jelenít meg a böngészőben.

Automatikusan növekvő mező értékének lekérdezése

Az előző példákban úgy adtuk hozzá a sorokat a táblához, hogy nem foglalkoztunk az `azonosito` oszlop értékével, hiszen az adatok beszúrásával az folyamatosan növekedett, értékére pedig nem volt szükségünk. Ha mégis szükségünk lenne rá, egy SQL lekérdezéssel bármikor lekérdezhethetjük az adatbázisból. De mi van akkor, ha azonnal szükségünk van az értékre? Fölösleges külön lekérdezést végrehajtani, hiszen a PHP tartalmazza a `mysql_insert_id()` függvényt, amely a legutóbbi `INSERT` kifejezés során beállított automatikusan növelt mező értékét adja vissza. A `mysql_insert_id()` függvénynek szükség esetén átadhatunk

egy kapcsolatazonosító paramétert, amelyet elhagyva a függvény alapértelmezés szerint a legutoljára létrejött MySQL kapcsolat azonosítóját használja.

Így ha meg akarjuk mondani a felhasználónak, hogy milyen azonosító alatt rögzítettük az adatait, az adatok rögzítése után közvetlenül hívjuk meg a `mysql_insert_id()` függvényt.

```
$parancs = "INSERT INTO tartomanyok ( tartomany, nem,  
    ➤ email ) VALUES ( '$tartomany', '$nem', '$email' )";  
mysql_query( $parancs, $kapcsolat );  
$azonosito = mysql_insert_id();  
print "Köszönjük. Az Ön tranzakció-azonosítója:  
    ➤ $azonosito. Kérjük jegyezze meg ezt a kódot.";
```

Adatok lekérdezése

Miután adatainkat már képesek vagyunk az adatbázisban tárolni, megismerkedhetünk azokkal a módszerekkel, amelyek az adatok visszanyerésére szolgálnak. Mint bizonyára már nyilvánvaló, a `mysql_query()` függvény használatával ki kell adnunk egy `SELECT` utasítást. Az azonban már korántsem ilyen egyszerű, hogyan jutunk hozzá a lekérdezés eredményéhez. Nos, miután végrehajtottunk egy sikeres `SELECT`-et, a `mysql_query()` visszaad egy úgynevezett eredményazonosítót, melynek felhasználásával elérhetjük az eredménytáblát és információkat nyerhetünk róla.

12

Az eredménytábla sorainak száma

A `SELECT` utasítás eredményeként kapott tábla sorainak számát a `mysql_num_rows()` függvény segítségével kérdezhetjük le. A függvény paramétere a kérdéses eredmény azonosítója, visszatérési értéke pedig a sorok száma a táblában. A 12.4. példaprogramban lekérdezzük a `tartomany` tábla összes sorát és a `mysql_num_rows()` függvénnyel megkapjuk a teljes tábla méretét.

12.4. program Sorok száma a `SELECT` utasítás eredményében.

```
1: <html>  
2: <head>  
3: <title>12.4. program A mysql_num_rows() függvény  
    használata</title>  
4: </head>  
5: <body>  
6: <?php
```

12.4. program (folytatás)

```
7: $felhasznalo = "jozsi";
8: $jelszo = "bubosvocskok";
9: $adatbazis = "pelda";
10: $kapcsolat = mysql_connect( "localhost",
    $felhasznalo, $jelszo );
11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
        a MySQL kiszolgálóhoz!" );
13: mysql_select_db( $adatbazis, $kapcsolat )
14:     or die ( "Nem lehet megnyitni a $adatbazis adatbázis: ".mysql_error() );
15: $eredmeny = mysql_query( "SELECT * FROM tartomanyok" );
16: $sorok_szama = mysql_num_rows( $eredmeny );
17: print "Jelenleg $sorok_szama sor van a táblában<P>";
18: mysql_close( $kapcsolat );
19: ?>
20: </body>
21: </html>
```

A `mysql_query()` függvény egy eredményazonosítót ad vissza. Ezt átadjuk a `mysql_num_rows()` függvénynek, amely megadja a sorok számát.

Az eredménytábla elérése

Miután végrehajtottuk a `SELECT` lekérdezést és az eredményazonosítót tároltuk, egy ciklus segítségével férhetünk hozzá az eredménytábla soraihoz. A PHP egy belső mutató segítségével tartja nyilván, hogy melyik sort olvastuk utoljára. Ha kiolvastunk egy eredménysort, a program automatikusan a következőre ugrik.

A `mysql_fetch_row()` függvénnyel kiolvashatjuk a belső mutató által hivatkozott sort az eredménytáblából. A függvény paramétere egy eredményazonosító, visszatérési értéke pedig egy tömb, amely a sor összes mezőjét tartalmazza. Ha elértük az eredménykészlet végét, a `mysql_fetch_row()` függvény `false` értékkel tér vissza. A 12.5. példa a teljes `tartomany` táblát megjeleníti a böngészőben.

12.5. program Tábla összes sorának és oszlopának megjelenítése

```
1: <html>
2: <head>
3: <title>12.5. program Tábla összes sorának és
   oszlopának megjelenítése</title>
4: </head>
5: <body>
6: <?php
7: $felhasznalo = "jozsi";
8: $jelszo = "bubosvocskok";
9: $adatbazis = "pelda";
10: $kapcsolat = mysql_connect( "localhost",
   $felhasznalo, $jelszo );
11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
   a MySQL kiszolgálóhoz!" );
13: mysql_select_db( $db, $kapcsolat )
14:     or die ( "Nem lehet megnyitni a $adatbazis
   adatbázist: ".mysql_error() );
15: $eredmeny = mysql_query( "SELECT * FROM tartomanyok"
   );
16: $sorok_szama = mysql_num_rows( $eredmeny );
17: print "Jelenleg $sorok_szama sor van a táblában<P>";
18: print "<table border=1>\n";
19: while ( $egy_sor = mysql_fetch_row( $eredmeny ) )
20:     {
21:         print "<tr>\n";
22:         foreach ( $egy_sor as $mezo )
23:             print "\t<td>$mezo</td>\n";
24:         print "</tr>\n";
25:     }
26: print "</table>\n";
27: mysql_close( $kapcsolat );
28: ?>
29: </body>
30: </html>
```

Kapcsolódunk a kiszolgálóhoz és kiválasztjuk az adatbázist, majd a `mysql_query()` függvénnyel egy `SELECT` lekérdezést küldünk az adatbázis kiszolgálójához. Az eredményt az `$eredmeny` változóban tároljuk, amit az eredménysorok számának lekérdezéséhez használunk, ahogy korábban láttuk.

A `while` kifejezésben a `mysql_fetch_row()` függvény visszatérési értékét az `$egy_sor` változóba töltjük. Ne feledjük, hogy az értékadó kifejezés értéke megegyezik a jobb oldal értékével, ezért a kiértékelés során a kifejezés értéke mindig igaz lesz, amíg a `mysql_fetch_row()` függvény nem nulla értékkel tér vissza. A ciklusmagban kiolvassuk az `$egy_sor` változóban tárolt elemeket és egy táblázat celláiként jelenítjük meg azokat a böngészőben.

A mezőket név szerint is elérhetjük, még hozzá kétféle módon.

A `mysql_fetch_array()` függvény egy asszociatív tömböt ad vissza, ahol a kulcsok a mezők nevei. A következő kódrészletben módosítottuk a 12.5. példa `while` ciklusát, a `mysql_fetch_array()` függvény felhasználásával:

```
print "<table border=1>\n";
while ( $egy_sor = mysql_fetch_array( $eredmeny ) )
{
    print "<tr>\n";
    print
"<td>".$egy_sor["email"]."</td><td>".$egy_sor["tartomany"].
    ➡ "</td>\n";
    print "</tr>\n";
}
print "</table>\n";
```

A `mysql_fetch_object()` függvény segítségével egy objektum tulajdonságai-ként férhetünk hozzá a mezőkhöz. A mezőnevek lesznek a tulajdonságok nevei. A következő kódrészletben ismét módosítottuk a kérdéses részt, ezúttal a `mysql_fetch_object()` függvényt használtuk.

```
print "<table border=1>\n";
while ( $egy_sor = mysql_fetch_object( $eredmeny ) )
{
    print "<tr>\n";
    print "<td>".$egy_sor->email."</td><td>".$egy_sor->
    ➡ tartomany."</td>\n";
    print "</tr>\n";
}
print "</table>\n";
```

A `mysql_fetch_array()` és `mysql_fetch_object()` függvények lehetővé teszik számunkra, hogy a sorból kinyert információkat szűrjük és végrehajtásuk sem kerül sokkal több időbe, mint a `mysql_fetch_row()` függvényé.

Adatok frissítése

Az adatokat az UPDATE utasítással frissíthetjük, amelyet természetesen a `mysql_query()` függvénnyel kell átadnunk az adatbázis kiszolgálójának. Itt is igazak a korábban elmondottak, azaz egy sikeres UPDATE utasítás nem feltétlenül jelent változást az adatokban. A megváltozott sorok számát a `mysql_affected_rows()` függvénnyel kérdezhetjük le, amelynek szükség szerint egy kapcsolatazonosítót kell átadnunk. Ennek hiányában a függvény – mint azt már megszokhattuk – a legfrissebb kapcsolatra értelmezi a műveletet. A `mysql_affected_rows()` függvényt bármely olyan SQL lekérdezés után használhatjuk, amely feltehetően módosított egy vagy több sort az adattáblában.

A 12.6. példa egy olyan programot tartalmaz, amely lehetővé teszi az adatbázis karbantartója számára, hogy bármelyik sor `tartomany` mezőjét megváltoztassa.

12.6. program Sorok frissítése az adatbázisban

```
1: <html>
2: <head>
3: <title>12.6. program Sorok frissítése az adatbázisban
4: </title>
5: </head>
6: <body>
7: <?php
8: $felhasznalo = "jozsi";
9: $jelszo = "bubosvocok";
10: $adatbazis = "pelda";
11: $kapcsolat = mysql_connect( "localhost",
    $felhasznalo, $jelszo );
12: if ( ! $kapcsolat )
13:     die( "Nem lehet kapcsolódni
        a MySQL kiszolgálóhoz!" );
14: mysql_select_db( $adatbazis, $kapcsolat )
15:     or die ( "Nem lehet megnyitni a $adatbazis
        adatbázist: ".mysql_error() );
16: if ( isset( $tartomany ) && isset( $azonosito ) )
17:     {
18:         $parancs = "UPDATE tartomanyok SET tartomany =
            '$tartomany' WHERE
            azonosito=$azonosito";
19:         $eredmeny = mysql_query( $parancs );
```

12.6. program (folytatás)

```
20:      if ( ! $eredmeny )
21:          die ("Nem sikerült a módosítás: "
                .mysql_error());
22: print "<h1>A tábla módosítva, ".
        mysql_affected_rows() .
23:      " sor változott</h1><p>";
24:      }
25: ?>
26: <form action="<? print $PHP_SELF ?>" method="POST">
27: <select name="azonosito">
28: <?
29: $eredmeny = mysql_query( "SELECT tartomany, azonosito
                           FROM tartomanyok" );
30: while( $egy_sor = mysql_fetch_object( $eredmeny ) )
31:     {
32:         print "<option value=\"\$egy_sor->azonosito\"";
33:         if ( isset($azonosito) && $azonosito ==
               $egy_sor->azonosito )
34:             print " selected";
35:         print "> $egy_sor->tartomany\n";
36:     }
37: mysql_close( $kapcsolat );
38: ?>
39: </select>
40: <input type="text" name="tartomany">
41: <input type="submit" value="Frissítés">
42: </form>
43: </body>
44: </html>
```

A művelet a szokásos módon indul: kapcsolódunk az adatbázis kiszolgálójához és kiválasztjuk az adatbázist. Ezek után megvizsgáljuk a \$tartomany és az \$azonosito változók meglétét. Ha mindent rendben találunk, összeállítjuk az SQL utasítást, amely a \$tartomany változó értékének megfelelően frissíti azt a sort, ahol az azonosito mező tartalma megegyezik az \$azonosito változó értékével. Ha nem létező azonosito értékre hivatkozunk vagy a kérdéses sorban a \$tartomany változó értéke megegyezik a tartomany mező tartalmával, nem kapunk hibaüzenetet, de a mysql_affected_rows() függvény visszatérési értéke 0 lesz. A módosított sorok számát kiírjuk a böngészőbe.

A karbantartó számára készítünk egy HTML űrlapot, amelyen keresztül elvégezheti a változtatásokat. Ehhez ismét a `mysql_query()` függvényt kell használnunk: lekérdezzük az `azonosito` és `tartomany` oszlopok összes elemét és beillesztjük azokat egy HTML `SELECT` listába. A karbantartó ezáltal egy lenyíló menüből választhatja ki, melyik bejegyzést kívánja módosítani. Ha a karbantartó már elküldte egyszer az űrlapot, akkor az utoljára hivatkozott `azonosito` érték mellé a `SELECTED` módosítót is kitesszük, így a módosítások azonnal látszódni fognak a menüben.

Információk az adatbázisokról

Mindeztidáig egyetlen adatbázisra összpontosítottunk, megtanulva, hogyan lehet adatokkal feltölteni egy táblát és az adatokat lekérdezni. A PHP azonban számos olyan eszközzel szolgál, amelyek segítségével megállapíthatjuk, hány adatbázis hozzáférhető az adott kapcsolaton keresztül és milyen ezek szerkezete.

Az elérhető adatbázisok kiírása

A `mysql_list_dbs()` függvény segítségével listát kérhetünk az összes adatbázisról, melyek az adott kapcsolaton keresztül hozzáférhetők. A függvény paramétere szükség szerint egy kapcsolatazonosító, visszatérési értéke pedig egy eredményazonosító. Az adatbázisok neveit a `mysql_tablename()` függvénnyel kérdezhetjük le, amelynek paramétere ez az eredményazonosító és az adatbázis sorszáma, visszatérési értéke pedig az adatbázis neve. Az adatbázisok sorszámozása 0-val kezdődik. Az összes adatbázis számát a `mysql_list_dbs()` függvény után meghívott `mysql_num_rows()` függvény adja meg.

12

12.7. program Adott kapcsolaton keresztül elérhető adatbázisok

```
1: <html>
2: <head>
3: <title>12.7. program Adott kapcsolaton keresztül
   elérhető adatbázisok
4: </title>
5: </head>
6: <body>
7: <?php
8: $felhasznalo = "jozsi";
9: $jelszo = "bubosvocskok";
10: $kapcsolat = mysql_connect( "localhost",
    $felhasznalo, $jelszo );
```

12.7. program (folytatás)

```

11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
           a MySQL kiszolgálóhoz!" );
13: $adatbazis_lista = mysql_list_dbs( $kapcsolat );
14: $szam = mysql_num_rows( $adatbazis_lista );
15: for( $x = 0; $x < $szam; $x++ )
16:     print mysql_tablename( $adatbazis_lista,
                             $x ). "<br>";

17: mysql_close( $kapcsolat );
18: ?>
19: </body>
20: </html>

```

A `mysql_list_dbs()` függvény visszaad egy eredményazonosítót, amelyet paraméterként átadunk a `mysql_num_rows()` függvénynek. Az így megkapott adatbázisok számát a `$szam` változóba töltjük és felhasználjuk a `for` ciklusban.

Az `$x` indexet minden lépésben eggyel növeljük, 0-tól a talált adatbázisok számáig, majd az eredményazonosítóval együtt átadjuk a `mysql_tablename()` függvénynek az adatbázis nevének lekérdezéséhez. A 12.1 ábrán a 12.7 program eredménye látható egy böngészőablakban.

12.1. ábra

Az elérhető adatbázisok listája



A `mysql_list_dbs()` függvény által visszaadott eredményazonosítót a `mysql_query()` eredményéhez hasonlóan is felhasználhatnánk, vagyis a tényleges neveket a `mysql_fetch_row()` függvénnyel is lekérdezhethetnénk. Ekkor eredményül egy tömböt kapnánk, melynek első eleme tartalmazná az adatbázis nevét.

Adatbázistáblák listázása

A `mysql_list_tables()` függvénnyel egy adott adatbázis tábláinak nevét sorolathatjuk fel. A függvény paraméterei az adatbázis neve és szükség szerint a kapcsolat azonosítója. A visszatérési érték egy eredményazonosító, ha az adatbázis létezik és jogunk van hozzáférni. Az eredményazonosítót `mysql_tablename()` vagy `mysql_fetch_row()` hívásokhoz használhatjuk fel, a fent látott módon.

A következő kódrészlet a `mysql_list_tables()` függvény segítségével az adatbázis összes tábláját kiírja.

```
$eredmeny = mysql_list_tables( "pelda", $kapcsolat );  
while ( $tabla_sor = mysql_fetch_row( $eredmeny ) )  
    print $tabla_sor[0]. "<BR>\n";
```

12

Információk a mezőkről

A `SELECT` lekérdezés után az eredménytábla mezőinek számát a `mysql_num_fields()` függvénnyel kaphatjuk meg. A függvény paramétere egy eredményazonosító, visszatérési értéke pedig egy egész szám, amely a mezők számát adja meg.

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );  
$mezok_szama = mysql_num_fields( $eredmeny );
```

A mezőkhöz sorszámokat rendelünk, a számozás 0-val kezdődik. A sorszám és az eredményazonosító alapján számos információt lekérdezhethetünk a mezőről, beleértve a nevét, típusát, legnagyobb hosszát és egyéb jellemzőit is.

A mező nevének lekérdezésére a `mysql_field_name()` függvény szolgál.

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );  
$mezok_szama = mysql_num_fields( $eredmeny );  
for ( $x=0; $x<$mezok_szama; $x++ )  
    mysql_field_name( $eredmeny, $x ) . "<br>\n";
```

A mező legnagyobb hosszát a `mysql_field_len()` függvénnyel kaphatjuk meg.

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
    mysql_field_len( $eredmeny, $x ) . "<BR>\n";
```

A mező egyéb jellemzőit a `mysql_field_flags()` függvénnyel olvashatjuk ki:

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
    mysql_field_flags( $eredmeny, $x ) . "<BR>\n";
```

Lehetőségünk van még a mező típusának megállapítására,
a `mysql_field_type()` függvénnyel:

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
    mysql_field_type( $eredmeny, $x ) . "<BR>\n";
```

Az adatbázis szerkezete – összeáll a kép

A 12.8 példa egyesíti a fenti eljárásokat és megjelenít minden elérhető adatbázist, táblát és mezőt.

12.8 program Minden adatbázis, tábla és mező megjelenítése

```
1: <html>
2: <head>
3: <title>12.8. program Minden adatbázis, tábla és
   mező megjelenítése</title>
4: </head>
5: <body>
6: <?php
7: $felhasznalo = "root";
8: $jelszo = "titkos";
9: $kapcsolat = mysql_connect( "localhost",
   $felhasznalo, $jelszo );
```

12.8 program (folytatás)

```

10: if ( ! $kapcsolat )
11:     die( "Nem lehet kapcsolódni
           a MySQL kiszolgálóhoz!" );
12: $adatbázis_lista = mysql_list_dbs( $kapcsolat );
13: while ( $adatbázisok = mysql_fetch_row
           ( $adatbázis_lista ) )
14:     {
15:         print "<b>".$adatbázisok[0]."</b>\n";
16:         if ( !@mysql_select_db( $adatbázisok[0],
$kapcsolat ) )
17:             {
18:                 print "<dl><dd>Nem lehet kiválasztani – " .
mysql_error() ." </dl>";
19:                 continue;
20:             }
21:         $tablak = mysql_list_tables( $adatbázisok[0],
$kapcsolat );
22:         print "\t<dl><dd>\n";
23:         while ( $tabla_sor = mysql_fetch_row( $tablak )
)
24:             {
25:                 print "\t<b>$tabla_sor[0]</b>\n";
26:                 $eredmeny = mysql_query( "SELECT * from "
.$tabla_sor[0] );

27:                 $mezok_szama = mysql_num_fields( $eredmeny );
28:                 print "\t\t<dl><dd>\n";
29:                 for ( $x=0; $x<$mezok_szama; $x++ )
30:                     {
31:                         print "\t\t<i>";
32:                         print mysql_field_type( $eredmeny, $x );
33:                         print "</i> <i>";
34:                         print mysql_field_len( $eredmeny, $x );
35:                         print "</i> <b>";
36:                         print mysql_field_name( $eredmeny, $x );
37:                         print "</b> <i>";
38:                         print mysql_field_flags( $eredmeny, $x );
39:                         print "</i><br>\n";
40:                     }
41:                 print "\t\t</dl>\n";
42:             }

```

12.8 program (folytatás)

```
43:      print "\t</dl>\n";
44:      }
45: mysql_close( $kapcsolat );
46: ?>
47: </body>
48: </html>
```

Először is a szokásos módon a MySQL kiszolgálóhoz kapcsolódunk, majd meghívjuk a `mysql_list_dbs()` függvényt. A kapott eredményazonosítót átadjuk a `mysql_fetch_row()` függvénynek és az `$adatbazisok` tömb első elemébe betöltjük az aktuális adatbázis nevét.

Ezután megpróbáljuk kiválasztani az adatbázist a `mysql_select_db()` függvénnyel. Ha nincs hozzáférési jogunk, akkor megjelenítjük a hibaüzenetet a böngészőablakban, majd a `continue` utasítással továbblépünk a következő adatbázisra. Ha ki tudjuk választani az adatbázist, a műveletet a táblanevek kiírásával folytatjuk.

Az adatbázis nevét átadjuk a `mysql_list_tables()` függvénynek, tároljuk az új eredményazonosítót, majd a `mysql_fetch_row()` függvénnyel kiolvassuk a táblák neveit. Minden táblán a következő műveletsort hajtjuk végre: kiírjuk a tábla nevét, majd összeállítunk egy SQL utasítást, amely az összes oszlopot lekérdezi az adott táblában. A `mysql_query()` függvénnyel végrehajtjuk a lekérdezést, majd az eredményazonosító alapján a `mysql_num_fields()` függvénnyel meghatározzuk a mezők számát.

A `for` ciklusban minden mezőn elvégezzük a következő műveletsort: beállítjuk az `$x` változót, hogy az aktuális mező sorszámát tartalmazza (ez kezdetben 0). Ezután a sorszám ismeretében meghívjuk az előző részben tárgyalt összes ellenőrző függvényt, eredményüket pedig átlátható formában, táblázatba rendezve elküldjük a böngészőnek.

Ezeket a műveleteket minden elérhető adatbázison elvégezzük.

A program egy áttekinthető szerkezeti vázlatot jelenít meg a böngészőablakban, amely az adott kapcsolaton keresztül elérhető összes adatbázist, adattáblát és mezőt tartalmazza. A 12.2. ábra a program kimenetét mutatja.

12.2. ábra

Az összes elérhető adatbázis, tábla és mező listája



Összefoglalás

Ebben az órában a MySQL adatbáziskezelés alapjait tanultuk meg: az adatok tárolását és visszanyerését.

Megtanultuk, hogyan építsük fel a kapcsolatot a MySQL kiszolgálóval a `mysql_connect()` és `mysql_pconnect()` függvények segítségével.

Az adatbázist a `mysql_select_db()` függvénnyel választottuk ki. Ha a kiválasztás nem sikerült, lekérdeztük a hibát a `mysql_error()` függvénnyel.

SQL utasításokat adtunk ki a `mysql_query()` függvény segítségével és a függvény által visszaadott eredményazonosító segítségével elértük az adatokat vagy megtudtuk a módosított sorok számát.

A PHP MySQL függvényeinek segítségével kiírtattuk az elérhető adatbázisokat, táblákat és mezőket, és az egyes mezőkről is bővebb információkat szereztünk.

Kérdések és válaszok

Ez az óra szigorúan a MySQL-re vonatkozik. Hogyan ültethetjük át ezeket a fogalmakat más adatbázissal működő rendszerekre?

Az mSQL-kezelő függvények például szinte teljesen megegyeznek a MySQL függvényekkel, de más SQL kiszolgálóknak is megvannak a saját PHP függvényeik. SQL utasításokat minden adatbáziskiszolgálónak küldhetünk. Ha szabványos ANSI SQL-lel dolgozunk, nem lesz különösebb problémánk az adatbáziskiszolgálók közötti átálláskor.

Hogyan írhatunk olyan kódot, amelyet könnyű átültetni egy másik adatbázis kiszolgáló környezetbe?

Gyakran jó ötlet az adatbázis-lekérdező függvényeket egy egyszerű osztályba vagy könyvtárba csoportosítani. Így ha fel kell készíteni a kódot egy másik adatbáziskezelővel való együttműködésre, a kódnak csak kis részét kell átírni.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan kell csatlakozni a MySQL adatbáziskiszolgálóhoz?
2. Melyik függvény szolgál az adatbázis kiválasztására?
3. Melyik függvénnyel küldhetünk SQL utasítást a kiszolgálónak?
4. Mit csinál a `mysql_insert_id()` függvény?
5. Tegyük fel, hogy küldtünk egy `SELECT` lekérdezést a MySQL-nek. Nevezünk meg három függvényt, amelyekkel hozzáférhetünk az eredményhez.
6. Tegyük fel, hogy küldtünk egy `UPDATE` lekérdezést a MySQL-nek. Melyik függvénnyel állapíthatjuk meg, hány sort érintett a módosítás?
7. Melyik függvényt kellene használnunk, ha az adatbáziskapcsolaton keresztül elérhető adatbázisok nevét szeretnénk megtudni?
8. Melyik függvénnyel kérdezhetjük le az adatbázisban található táblák neveit?

Feladatok

1. Hozzunk létre egy adattáblát három mezővel: `email` (legfeljebb 70 karakter), `uzenet` (legfeljebb 250 karakter) és `datum` (a UNIX időbélyegzőt tartalmazó egész szám). Tegyük lehetővé a felhasználóknak, hogy feltöltsék az adatbázist.
2. Készítsünk programot, amely megjeleníti az 1. pontban elkészített tábla tartalmát.



13. ÓRA

Kapcsolat a külvilággal

Ezen az órán olyan függvényekkel ismerkedünk meg, amelyek a „külvilággal” való érintkezést teszik lehetővé. Az óra során a következőkről tanulunk:

- Környezeti változók – részletesebben
- A HTTP kapcsolat felépítése
- Távoli kiszolgálón levő dokumentumok elérése
- Saját HTTP kapcsolat létrehozása
- Kapcsolódás más hálózati szolgáltatásokhoz
- Levélküldés programból

Környezeti változók

Már találkoztunk néhány környezeti változóval; ezeket a PHP a kiszolgáló segítségével bocsátotta rendelkezésünkre. Néhány ilyen változóval több dolgot is megtudhatunk weboldalunk látogatóiról, arra azonban gondoljunk, hogy lehet, hogy ezek a változók a mi rendszerünkön nem elérhetők, esetleg a kiszolgálóprogram nem támogatja azokat, így használatuk előtt érdemes ezt ellenőrizni. A 13.1. táblázat ezek közül a változók közül mutat be néhányat.

13.1. táblázat Néhány hasznos környezeti változó

<i>Változó</i>	<i>Leírás</i>
\$HTTP_REFERER	A programot meghívó webhely címe.
\$HTTP_USER_AGENT	Információ a látogató böngészőjéről és rendszeréről.
\$REMOTE_ADDR	A látogató IP címe.
\$REMOTE_HOST	A látogató számítógépének neve.
\$QUERY_STRING	Az a (kódolt) karakterlánc, amely a webcímet kiegészíti (formátuma kulcs=ertek&masikkulcs=masikertek). E kulcsok és értékek elérhetővé kell, hogy váljanak programunk részére a megfelelő globális változókból is.
\$PATH_INFO	Az esetleg a webcímhez illesztett további információ.

A 13.1. példaprogram ezen változók értékét jeleníti meg a böngészőben.

13.1. program Néhány környezeti változó felsorolása

```

1: <html>
2: <head>
3: <title>13.1. program Néhány környezeti változó
   felsorolása</title>
4: </head>
5: <body>
6: <?php
7: $korny_valtozok = array( "HTTP_REFERER",
   "HTTP_USER_AGENT", "REMOTE_ADDR", "REMOTE_HOST",
   "QUERY_STRING", "PATH_INFO" );

```

13.1. program (folytatás)

```
8: foreach ( $korny_valtozok as $valtozo )  
9:     {  
10:         if ( isset( $$valtozo ) )  
11:             print "$valtozo: ${$valtozo}<br>";  
12:     }  
13: ?>  
14: </body>  
15: </html>
```

Figyeljük meg, hogyan alakítottuk a szöveggént tárolt változóneveket valódi változókká. Ezt a módszert a negyedik órán tanultuk.

A 13.1. ábrán a 13.1. kód kimenetét láthatjuk. Az ábrán látható adatokat úgy kaptuk, hogy a programot egy másik oldal hivatkozásán keresztül hívtuk meg. A programot meghívó hivatkozás így nézett ki:

```
<A HREF='13.1.program.php/tovabbi/utvonal?nev=ertek'>gyerünk</A>
```

Amint láthatjuk, a hivatkozás relatív útvonalat használ a 13.1.program.php meghívására.

13.1. ábra

*Néhány környezeti
változó kiírása
a böngészőben*



Az elérési út azon része, mely programunk nevét követi, (jelen esetben a /tovabbi/utvonal) a \$PATH_INFO változóban áll majd rendelkezésünkre.

A lekérdezés szövegét nem dinamikusan illesztettük a hivatkozásba (`nev=ertek`), de ettől függetlenül az a `$QUERY_STRING` változóban lesz elérhető. A lekérdező karakterláncokkal legtöbbször akkor találkozunk, ha egy `GET` metódust használó űrlap hívja meg a programot, de magunk is előállíthatunk ilyet, hogy segítségével adatokat továbbíthassunk lapról lapra. A lekérdező karakterlánc név–érték párokból áll, melyeket `ÉS` jel (`&`) választ el egymástól. Az adatpárok URL kódolású formában kerülnek a webcímbe, hogy a bennük szereplő, a webcímekben nem megengedett vagy más jelentő karakterek ne okozzanak hibát. Ezt úgy oldották meg, hogy a problémás karaktereket hexadecimális megfelelőjükre cserélik. Bár a teljes karakterlánc elérhető a `$QUERY_STRING` környezeti változóban, nagyon ritkán lehet szükségünk rá, pontosan a kódolt mivolta miatt. De azért is mellőzük a használatát, mert az összes név–érték pár rendelkezésünkre áll globális változók formájában is (jelen esetben például létrejön a `$nev` változó és az értéke `ertek` lesz).

A `$HTTP_REFERER` változó értéke akkor lehet hasznos számunkra, ha nyomon szeretnénk követni, hogy programunkat mely hivatkozáson keresztül érték el. Nagy körültekintéssel kell eljárunk azonban, mert ez és tulajdonképpen bármely másik környezeti változó is nagyon könnyen „meghamisítható”. (Az óra folyamán azt is megtudhatjuk, hogyan.) A nevet sajnálatos módon a kezdetekkor egy fejlesztő elírta (helyesen `HTTP_REFERER`-nek kellene neveznünk), módosítása a korábbi változatokkal való összeegyeztethetőség megőrzése érdekében nem történt meg. Ezenkívül nem minden böngésző adja meg ezt az információt, így használatát célszerű elkerülnünk.

A `$HTTP_USER_AGENT` változó szétbontásával rájöhethetünk, milyen operációs rendszert, illetve böngészőt használ a látogató, de tudnunk kell, hogy ezek az adatok is hamisíthatók. A változóban elérhető értéket akkor használhatjuk, ha a böngésző típusától vagy a változattól függően más és más HTML kódot vagy JavaScriptet szeretnénk elküldeni a látogatónak. A tizenhetedik és tizennyolcadik órában mindent megtanulunk arról, hogyan nyerhetjük ki a számunkra fontos adatokat ebből a karakterláncból.

A `$REMOTE_ADDR` változó a látogató IP címét tartalmazza, így a webhely látogatóinak azonosítására használható. Ne feledjük azonban, hogy a felhasználók IP címe többnyire nem állandó, hiszen az internetszolgáltatók általában dinamikusan osztják ki felhasználóiknak a címeket, így az minden csatlakozás során más lehet, tehát ugyanaz az IP cím különböző látogatókat jelenthet és a különböző IP címek is takarhatnak egyetlen felhasználót.

A `$REMOTE_HOST` változó nem biztos, hogy hozzáférhető; ez a kiszolgáló beállításaitól függ. Ha létezik, a felhasználó számítógépének nevét találhatjuk benne. A változó meglétéhez a kiszolgálónak az IP cím alapján minden kéréskor le kell kérdeznie a gép nevét, ami időigényes feladat, így a kiszolgáló ezen képességét

a hatékonyság kedvéért gyakran letiltják. Ha nem létezne ilyen változó, az információhoz a \$REMOTE_ADDR segítségével juthatunk hozzá; később azt is látjuk majd, hogyan.

A HTTP ügyfél-kiszolgáló kapcsolat rövid ismertetése

A kiszolgáló és az ügyfél közti adatforgalom részletes ismertetése túlmutat könyvünk keretein, többek között azért, mert a PHP ezen részletek szakszerű kezeléséről is gondoskodik helyettünk. A folyamatot azonban nem hiábavaló alapjaiban megismerni, mert szükségünk lehet rá, ha olyan programot szeretnénk írni, amely weboldalakat tölt le vagy webcímek állapotát ellenőrzi.

ÚJDONSÁG

A *HTTP* jelentése *hiperszöveg-átviteli protokoll*. Nem más, mint szabályok halmaza, melyek előírják, hogy az ügyfél milyen kérésekkel fordulhat a kiszolgálóhoz és az milyen válaszokat kell, hogy adjon. Mind az ügyfél, mind a kiszolgáló információt szolgáltat magáról és a küldött adatokról. Ezen információk leg többjét a PHP-ből környezeti változókon keresztül érhetjük el.

A kérés

Az ügyfél szigorú szabályok szerint kérhet adatokat a kiszolgálótól. A kérés legfeljebb három részből állhat:

- A kérés sora
- Fejléc
- Törzs

A legfontosabb a kérés sora. Itt határozzuk meg a kérés fajtáját (GET, HEAD vagy POST), a kért dokumentum címét és az átviteli protokoll változatszámát (HTTP/1.0 vagy HTTP/1.1). Egy jellemző kérés a következőképpen néz ki:

```
GET /egy_dokumentum.html HTTP/1.0
```

Ekkor az ügyfél egy GET kérést kezdeményez. Más szóval lekér egy dokumentumot, de adatokat nem küld. (Valójában kisebb mennyiségű adat küldésére a GET kérés alkalmazásakor is van lehetőség, ha azt lekérdező karakterlánc formájában hozzáadjuk az URL végéhez.) A HEAD módszer akkor alkalmazandó, ha nem magára a dokumentumra, csak annak tulajdonságaira vagyunk kíváncsiak, végül a POST kérés arra szolgál, hogy adatokat küldjünk az ügyféltől a kiszolgálónak. Ez legtöbbször HTML űrlapok esetében használatos.

A kifogástalan GET kéréshez egyetlen kérésor elküldése is elegendő. A kérés végét úgy tudathatjuk a kiszolgálóval, hogy egy üres sort küldünk neki.

A legtöbb ügyfél (általában böngészőprogram) a kérésoron kívül küld egy fejléc részt is, amely név–érték párokból áll. Az oldal lekérésével érkezett fejlécek legtöbbjéhez környezeti változók formájában hozzá is férhetünk programunkból. Az ügyfél fejlécének minden sora egy kettősponttal elválasztott névből és értékből áll. A 13.2 táblázat néhány lehetséges fejléc-nevet sorol fel.

13.2 táblázat Néhány fejléc-kulcs

<i>Név</i>	<i>Leírás</i>
Accept	Az ügyfél által kezelhető dokumentumtípusok listája.
Accept-Encoding	Az ügyfél számára elfogadható kódolási és tömörítési formátumok listája.
Accept-Charset	Az ügyfél által előnyben részesített nemzetközi karakterkészlet.
Accept-Language	Az ügyfél által előnyben részesített nyelv (a magyar nyelv esetében „hu”).
Host	Az ügyfél kérése által megcímzett gép neve. Egyes kiszolgálók csak látszólagos (virtuális) kiszolgálók, a beérkező kéréseket nem önálló számítógép kezeli. Ilyen esetekben komoly jelentősége van ennek az adatnak.
Referer	Azon dokumentum címe, melynek egyik hivatkozása alapján a kérés létrejött.
User-Agent	Az ügyfélprogram típusa és változatszáma.

A GET és a HEAD eljárásoknál a kérést a fejléc és az utána következő üres sor zárja, a POST típusnál azonban az üres sort az üzenet törzse követi. A törzs tartalmazza a kiszolgálónak szóló összes adatot, jobbra URL kódolású név–érték párok, a lekérdező karakterláncokhoz hasonlóan.

A 13.2 példában egy jellemző, mindennapos kérést mutatunk be, melyet egy Netscape 4.7 kezdeményezett.

13.2. példa Jellemző Netscape-kérés

```
GET / HTTP/1.0
Referer: http://www.linuxvilag.hu/index.html
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (X11; I; Linux 2.2.13 i686)
Host: www.kiskapu.hu
Accept: image/gif, image/x-xbitmap, image/jpeg,
        image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

A válasz

Miután a kiszolgáló fogadta az ügyfél kérését, választ küld. A válasz általában a következő három részből tevődik össze:

- Állapot
- Fejléc
- Törzs

Amint látjuk, a kérés és a válasz szerkezete igen hasonló. A fejléc bizonyos sorai tulajdonképpen ügyfél és kiszolgáló által küldve egyaránt megállják helyüket, nevezetesen azok, amelyek a törzsre vonatkoznak.

Az állapot sor a kiszolgáló által használt protokollt (HTTP/1.0 vagy HTTP/1.1) adja meg, illetve egy válaszkódot és egy azt magyarázó szöveget.

Számos válaszkód létezik, mindegyik a kérés sikerességéről vagy sikertelenségéről ad információt. A 13.3-as táblázat a leggyakoribb kódok jelentését tartalmazza.

13.3 táblázat Néhány válaszkód

<i>Kód</i>	<i>Szöveg</i>	<i>Leírás</i>
200	OK	A kérés sikeres volt és a törzsben megtalálható a válasz.
301	Moved Permanently	A kért adat nem található a kiszolgálón, de a fejlécben megtalálható annak új helye.
302	Moved Temporarily	A kért adatot ideiglenesen áthelyezték, de a fejléc elárulja, hogy hová.
404	Not Found	A kért adat nem található a megadott címen.
500	Internal Server Error	A kiszolgáló vagy egy CGI program komoly problémába ütközött a kérés teljesítése során.

Ennek megfelelően a jellegzetes válaszsor a következőképpen fest:

HTTP/1.1 200 OK

A fejléc rész a válaszfejléc soraiból áll, melyek formátuma a kérésfejléc soraihoz hasonló. A 13.4 táblázat a leggyakrabban alkalmazott fejlécelemekből szemezget.

13.4 táblázat A kiszolgáló válaszfejlécének leghétköznapiabb elemei

<i>Név</i>	<i>Leírás</i>
Date	Az aktuális dátum
Server	A kiszolgáló neve és változatszáma
Content-Type	A törzsben szereplő adat MIME típusa
Content-Length	A törzs mérete bájtban
Location	Egy alternatív dokumentum teljes elérési útja (kiszolgálóoldali átirányítás esetén)

Miután a kiszolgáló elküldte a fejléceket és a szokásos üres sort, elküldi a törzset (a kérésben tulajdonképpen igényelt dokumentumot). A 13.3 példa egy jellemző választ mutat be.

13.3. példa A kiszolgáló válasza

```
1: HTTP/1.1 200 OK
2: Date: Sun, 30 Jan 2000 18:02:20 GMT
3: Server: Apache/1.3.9 (Unix)
4: Connection: close
5: Content-Type: text/html
6:
7: <html>
8: <head>
9: <title>13.3. lista A kiszolgáló válasza</title>
10: </head>
11: <body>
12: Üdvözlét!
13: </body>
14: </html>
```

Dokumentum letöltése távoli címről

Bár a PHP kiszolgálóoldali nyelv, esetenként ügyfélként viselkedve adatokat kérhet egy távoli címről és a kapott adatokat programunk rendelkezésére bocsáthatja. Ha már jártasak vagyunk a kiszolgálón lévő fájlok kezelésében, nem okozhat komoly gondot a távoli helyen lévő adatok lekérése sem. Az a helyzet ugyanis, hogy a kettő között – formailag – semmi különbség nincs. Az `fopen()`-nel ugyanúgy megnyithatunk egy webcímet, ahogy azt egy fájl esetében tennénk. A 13.4. példaprogram egy távoli kiszolgálóhoz kapcsolódva adatokat kér le, majd megjeleníti azokat a böngészőben.

13

13.4. program Weboldal letöltése és megjelenítése az `fopen()`-nel

```
1: <html>
2: <head>
3: <title>13.4. program Weboldal letöltése és megjele-
   nítése az fopen()-nel</title>
4: </head>
5: <body>
6: <?php
7: $weboldal = "http://www.php.net/index.php";
8: $fajlmutato = fopen( $weboldal, "r" ) or
   die("A $weboldal nem nyitható meg");
```

13.4. program (folytatás)

```
9: while ( ! feof( $fajlmutato ))  
10:   print fgets( $fajlmutato, 1024 );  
11: ?>  
12: </body>  
13: </html>
```

Ha megnézzük ezt az oldalt, akkor a PHP honlapját kell látnunk, azzal a kis különbséggel, hogy a képek nem jelennek meg. Ennek oka, hogy az oldalon található IMG hivatkozások általában relatívak, így az oldal megjelenítésekor a böngésző a mi kiszolgálónkon keresi azokat. Ezen úgy segíthetünk, hogy a HEAD HTML elemet az alábbi sorral bővítjük:

```
<base href="http://www.php.net/index.php">
```

Az esetek többségében azonban nem a letöltött adatok megjelenítése, hanem a feldolgozás a feladat.

Az `fopen()` egy fájlmutatóval tér vissza, ha sikerült felépítenie a kapcsolatot és `false` (hamis) értékkel, ha nem jött létre a kapcsolat vagy nem találta a fájlt. A kapott fájlmutató a továbbiakban ugyanúgy használható az olvasáshoz, mint a helyi fájlok kezelése esetében. A PHP a távoli kiszolgálónak úgy mutatkozik be, mint egy PHP ügyfél. A szerző rendszerén a PHP a következő kérést küldi el:

```
GET /index.php HTTP/1.0  
Host: www.php.net  
User-Agent: PHP/4.0.3
```

Ez a megközelítés elég egyszerű, így az esetek többségében elegendő, ha ezt alkalmazzuk weblapok letöltéséhez. Szükségünk lehet viszont más hálózati szolgáltatásokhoz való kapcsolódásra, vagy épp csak többet szeretnénk megtudni a dokumentumról, elemezve annak fejlécét. Ekkor már más eszközökhöz kell folyamodnunk. Hogy ezt hogyan tehetjük meg? Nos, az óra folyamán lesz még erről szó.

Átalakítás IP címek és gépnevek között

Ha kiszolgálónk nem is bocsátja rendelkezésünkre a látogató gépének nevét a `$REMOTE_HOST` változóban, a látogató címéhez minden bizonnyal hozzájuthatunk a `$REMOTE_ADDR` változóból. A változóra alkalmazva a `gethostbyaddr()` függvényt megtudhatjuk a látogató gépének nevét. A függvény egy IP címet ábrázo-

ló karakterláncot vár paraméterként és az annak megfelelő gépnévvel tér vissza. Ha hiba lép fel a név előállítása során, kimenetként a beadott IP címet kapjuk meg változatlanul. A 13.5-ös program a `$REMOTE_HOST` hiánya esetén a `gethostbyaddr()` függvény segítségével előállítja a felhasználó gépének nevét.

13.5. program Gépnév lekérdezése a `gethostbyaddr()` segédletével

```
1: <html>
2: <head>
3: <title>13.5. program Gépnév lekérdezése
   a gethostbyaddr() segédletével</title>
4: </head>
5: <body>
6: <?php
7: if ( isset( $REMOTE_HOST ) )
8:     print "Üdvözljük $REMOTE_HOST<br>";
9: elseif ( isset ( $REMOTE_ADDR ) )
10:     print " Üdvözljük
      ".gethostbyaddr( $REMOTE_ADDR ). "<br>";
11: else
12:     print " Üdvözljük, akárki is Ön.<br>";
13: ?>
14: </body>
15: </html>
```

Ha a `$REMOTE_HOST` változó létezik, egyszerűen megjelenítjük annak értékét. Ha nem létezik, megpróbáljuk a `$REMOTE_ADDR` alapján a `gethostbyaddr()` függvénnyel előállítani a gépnevet. Ha minden kísérletünk csődöt mondott, általános üdvözlő szöveget jelenítünk meg.

Egy gép címének a név alapján történő meghatározásához a `gethostbyname()` függvényt használhatjuk. Ennek paramétere egy gépnév, visszatérési értéke pedig hiba esetén ugyanez a gépnév, sikeres átalakítás esetén viszont a megnevezett gép IP címe.

Hálózati kapcsolat létesítése

Eddig minden feladatunkat könnyen meg tudtuk oldani, mert a PHP a távoli weblapok lekérését közönséges fájlkezeléssé egyszerűsítette számunkra. Néha azonban nagyobb felügyeletet kell gyakorolnunk egy-egy hálózati kapcsolat felett, de legalábbis a szokásosnál több jellemzőjét kell ismernünk.

A hálózati kiszolgálók felé az `fsockopen()` függvénnyel nyithatunk meg egy kapcsolatot. Paramétere egy IP cím vagy gépnév, egy kapuszám és két változóhivatkozás. Változóhivatkozásokat úgy készítünk, hogy `ÉS (&)` jelet írunk a változó neve elé. Megadhatunk továbbá egy nem kötelező időkorlát-paramétert is, amely annak az időtartamnak a hossza (másodpercben), ameddig a függvény vár a kapcsolat létrejöttére. Ha sikerült megteremtenie a kapcsolatot, egy fájlmutatóval tér vissza, egyébként pedig `false` értéket ad.

A következő kódrészlet kapcsolatot nyit egy webkiszolgáló felé.

```
$fajlmutato = fsockopen( "www.kiskapu.hu", 80, &$hibakod,  
    ➡ &$hibaszoveg, 30 );
```

A webkiszolgálók általában a 80-as kapun várják a kéréseket.

Az első hivatkozott változó, a `$hibaszam` sikertelen művelet esetén egy hibakódot tartalmaz, a `$hibaszoveg` pedig bővebb magyarázattal szolgál a hibáról.

Miután visszakaptuk a kapcsolat fájlmutatóját, a kapcsolaton keresztül az `fputs()` és `fgets()` függvények segítségével írhatunk és olvashatunk is, ahogy ezt egy közönséges fájl esetében is tehetjük. Munkánk végeztével a kapcsolatot az `fclose()` függvénnyel bonthatjuk.

Most már elegendő tudás birtokában vagyunk ahhoz, hogy kapcsolatot létesíthessünk egy webkiszolgálóval. A 13.6-os programban megnyitunk egy HTTP kapcsolatot, lekérünk egy fájlt, majd egy változóban tároljuk azt.

13.6. program Weboldal lekérése az `fsockopen()` használatával

```
1: <html>  
2: <head>  
3: <title>13.6. program Weboldal lekérése az fsockopen()  
    használatával</title>  
4: </head>  
5: <body>  
6: <?php  
7: $kiszolgalo = "www.kiskapu.hu";  
8: $lap = "/index.html";  
9: $fajlmutato = fsockopen( "$kiszolgalo", 80,  
    &$hibaszam, &$hibaszoveg);  
10: if ( ! $fajlmutato )
```

13.6. program (folytatás)

```
11: die ( "Nem sikerült kapcsolódni a $kiszolgalo
    géphez:\nA hiba kódja: $hibaszam\nA hiba
    szövege: $hibaszoveg\n" );
12:
13: $lekeres = "GET $lap HTTP/1.0\r\n";
14: $lekeres .= "Host: $kiszolgalo\r\n";
15: $lekeres .= "Referer:
    http://www.linuxvilag.hu/index.html\r\n";
16: $lekeres .= "User-Agent: PHP browser\r\n\r\n";
17: $lap = array();
18: fputs ( $fajlmutato, $lekeres );
19: while ( ! feof( $fajlmutato ) )
20:     $lap[] = fgets( $fajlmutato, 1024 );
21: fclose( $fajlmutato );
22: print "A kiszolgáló ".(count($lap))."
    sort küldött el!";
23: ?>
24: </body>
25: </html>
```

Figyeljük meg, milyen kérésfejléct küldtünk a kiszolgálónak. A távoli gép webmestere a fejléc User-Agent sorában feltüntetett adatokat látni fogja a webkiszolgáló naplójában. A webmester számára ráadásul úgy fog tűnni, mintha a `http://www.linuxvilag.hu/index.html` címről mutatott volna egy hivatkozás a kért oldalra és mi ennek segítségével kértük volna le az oldalt. Emiatt programjainkban némi fenntartással kell kezelnünk az ezen fejlécekből előálló környezeti változók tartalmát és sokkal inkább segédadatoknak kell azokat tekintenünk, mintsem tényeknek.

Vannak esetek, amikor meg kell hamisítani a fejléceket. Szükségünk lehet például olyan adatokra, amelyeket a kiszolgáló csak Netscape böngészőnek küld el. Ezek elérésének egyetlen módja, ha a User-Agent fejlécsorban egy Netscape böngészőre jellemző karakterláncot küldünk. Ennek a webmesterek nem igazán örülnek, hiszen döntéseiket a kiszolgáló gyűjtötte statisztikák alapján kell meghozniuk. Hogy segíthessünk ebben nekik, lehetőleg ne hamisítsuk meg adatainkat, hacsak feltétlenül nem szükséges.

A 13.6-os program nem sokkal bővítette a PHP beépített lapkérő módszereit. A 13.7-es példa azonban már az `fsockopen()`-es módszert alkalmazza egy tömbben megadott weboldalak letöltéséhez és közben a kiszolgáló által visszaadott hibakódokat is ellenőrzi.

13.7. program Az állapotsor megjelenítése webkiszolgálók válaszaiból

```
1: <html>
2: <head>
3: <title>13.7. program Az állapotsor megjelenítése
   webkiszolgálók válaszaiból</title>
4: </head>
5: <body>
6: <?php
7: $ellenorizando = array (      "www.kiskapu.hu" =>
                                   "/index.html",
8:                                "www.virgin.com" =>
                                   "/nincsilyenlap.html",
9:                                "www.4332blah.com" =>
                                   "/nemletezogep.html"
10:                               );
11: foreach ( $ellenorizando as $kiszolgalo => $lap )
12: {
13:     print "Csatlakozási kísérlet a $kiszolgalo géphez
        ...<BR>\n";
14:     $fajlmutato = fsockopen( "$kiszolgalo", 80,
                               &$hibaszam, &$hibaszoveg, 10);
15:     if ( ! $fajlmutato )
16:     {
17:         print "A $kiszolgalo géphez nem sikerült
            csatlakozni:\n<br>A hiba kódja:
            $hibaszam\n<br>Szövege: $hibaszoveg\n";
18:         print "<br><hr><br>\n";
19:         continue;
20:     }
21:     print "A $lap oldal fejlécének letöltése ...<br>\n";
22:     fputs( $fajlmutato, "HEAD $lap HTTP/1.0\r\n\r\n" );
23:     print fgets( $fajlmutato, 1024 );
24:     print "<br><br><br>\n";
25:     fclose( $fajlmutato );
26: }
27: ?>
28: </body>
29: </html>
```

Először létrehozunk egy asszociatív tömböt az ellenőrzendő kiszolgálónevekből és az oldalak címeiből. Ezeket sorra vesszük a foreach utasítás segítségével. Minden elemnél az fsockopen() -nel kezdeményezünk egy kapcsolatot az adott címmel,

de a válaszra csak 10 másodpercig várunk. Ha ezen belül nem érkezne válasz, üzenetet jelenítünk meg a böngészőben, majd a `continue` utasítással a következő címre lépünk. Ha a kapcsolatteremtés sikeres volt, kérelmet is küldünk a kiszolgálónak. A kéréshez a `HEAD` eljárást használjuk, mert a lap tartalmára nem vagyunk kíváncsiak. Az `fgets()` segítségével beolvassuk az első sort, az állapotsort. A jelen példában nem elemezzük a fejléc többi részét, ezért lezárjuk a kapcsolatot az `fclose()` függvénnyel, majd továbblépünk a következő címre.

A 13.2-es ábrán a 13.7. program kimenete látható.

13.2. ábra

Kiszolgálók választ megjelölő program



NNTP kapcsolat létrehozása az `fsockopen()`-nel

Az `fsockopen()` függvény tetszőleges internetkiszolgálóval képes kapcsolatot teremteni. A 13.8. példában egy NNTP (Usenet) kiszolgálóhoz kapcsolódunk: kiválasztunk egy hírcsoportot, majd megjelenítjük első üzenetének fejlécét.

13.8. program Egyszerű NNTP kapcsolat az `fsockopen()` használatával

```
1: <html>
2: <head>
3: <title>13.8. program Egyszerű NNTP kapcsolat az
   fsockopen() használatával</title>
4: </head>
5: <body>
6: <?php
7: $kiszolgáló = "news"; // ezt saját hírkiszolgálónk
   címére kell beállítanunk
```

13.8. program (folytatás)

```
8: $csoport = "alt.test";
9: $sor = "";
10: print "<pre>\n";
11: print "– Csatlakozás a $kiszolgaló
      kiszolgálóhoz\n\n";
12: $fajlmutato = fsockopen( "$kiszolgaló", 119,
                          &$hibaszam, &$hibaszoveg, 10 );
13: if ( ! $fajlmutato )
14:     die("Csatlakozás a $kiszolgaló géphez
      sikertelen\n$hibaszam\n$hibaszoveg\n\n");
15: print "– Kapcsolat a $kiszolgaló géppel
      felvéve\n\n";
16: $sor = fgets( $fajlmutato, 1024 );
17: $allapot = explode( " ", $sor );
18: if ( $allapot[0] != 200 )
19:     {
20:         fputs( $fajlmutato, "close" );
21:         die("Hiba: $sor\n\n");
22:     }
23: print "$sor\n";
24: print "– A $csoport kiválasztása\n\n";
25: fputs( $fajlmutato, "group ".$csoport."\n" );
26: $sor = fgets( $fajlmutato, 1024 );
27: $allapot = explode( " ", $sor );
28: if ( $allapot[0] != 211 )
29:     {
30:         fputs( $fajlmutato, "close" );
31:         die("Hiba: $sor\n\n");
32:     }
33: print "$sor\n";
34: print "– Az első üzenet fejlécének lekérése\n\n";
35: fputs( $fajlmutato, "head\n" );
36: $sor = fgets( $fajlmutato, 1024 );
37: $allapot = explode( " ", $sor );
38: print "$sor\n";
39: if ( $allapot[0] != 221 )
40:     {
41:         fputs( $fajlmutato, "close" );
42:         die("Hiba: $sor\n\n");
43:     }
```

13.8. program (folytatás)

```
44: while ( ! ( strpos($sor, ".") === 0 ) )
45:     {
46:         $sor = fgets( $fajlmutato, 1024 );
47:         print $sor;
48:     }
49: fputs( $fajlmutato, "close\n" );
50: print "</pre>";
51: ?>
52: </body>
53: </html>
```

A 13.8. program egy kicsit többet mutat annál, hogyan nyissunk az `fsockopen()`-nel NNTP kapcsolatot. Valós alkalmazás esetén a visszakapott sorok elemzését célszerű egy függvénnyel végezni, egyrészt azért, hogy megszabaduljunk az ismétlésektől, másrészt azért, hogy több adatot is kinyerhessünk a válaszból. Mielőtt azonban újra feltalálnánk a kereket, célszerű, ha megismerkedünk a PHP IMAP függvényeivel, amelyekkel mindez a munka automatizálható.

A fenti programban a `$kiszorgalo` változó tárolja a hírkiszolgáló nevét, a `$csoport` pedig annak a csoportnak a nevét, melyhez kapcsolódni szeretnénk. Ha ki szeretnénk próbálni ezt a programot, javítsuk át a `$kiszorgalo` változó értékét az internetszolgáltatónk által megadott hírkiszolgáló névére, a `$csoport`-ot kedvenc csoportunkra. Az `fsockopen()`-nel a kiszolgáló 119-es kapujára csatlakozunk, mert ez a hírcsoport szolgáltatás szabványos kapuja. Ha nem kapunk vissza használható fájlmutatót, a `die()` függvény segítségével megjelenítünk egy hibaüzenetet a böngészőben és kilépünk a programból. Kapcsolódás esetén a kiszolgálótól kapunk kell egy nyugtázó üzenetet, amit az `fgets()` függvénnyel próbálunk fogadni. Ha minden zökkenőmentesen zajlott, a visszakapott szöveg a 200-as állapotkóddal kezdődik. Ennek ellenőrzéséhez az `explode()` függvénnyel a szöközők mentén szétdaraboljuk a `$sor` változót és a darabokat egy tömbbe helyezzük. Az `explode()` függvénnyel részletesebben a tizenhetedik órában foglalkozunk. Ha a kapott tömb első eleme (a nullás indexű) 200, akkor továbblépünk, egyébként pedig befejezzük a programot.

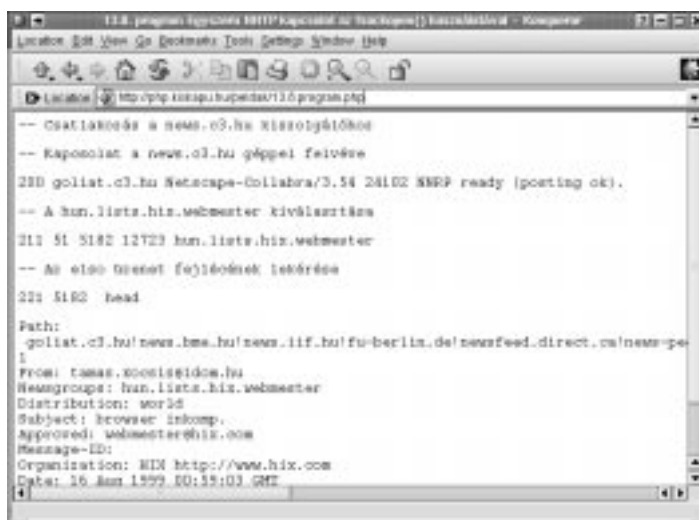
Ha minden az elvárásoknak megfelelően haladt, akkor a "group" paranccsal kiválasztjuk a kívánt hírcsoportot. Ha ez is sikerült, a kiszolgálónak egy 211-es állapotkóddal kezdődő szöveget kell válaszul küldenie. Ezt is ellenőrizzük és sikertelenség esetén kilépünk a programból.

A hírcsoport kiválasztása után küldünk egy "head" parancsot. Ennek hatására visszakapjuk a csoport első üzenetének fejlécét. Természetesen kérésünkre ez esetben is először egy nyugta érkezik, mely a 211-es állapotkódot kell, hogy tartalmazza. A fejléc csak ezt követően érkezik, számos hasznos információt tartalmazó sorral. A fejlécet záró sor egyetlen pontot tartalmaz. Ezt a sort egy while ciklussal keressük meg. Mindaddig, míg a kiszolgáló válaszsora nem ponttal kezdődik, további sorokat olvasunk be és mindegyiket megjelenítjük a böngészőben.

Végző lépésként bontjuk a kapcsolatot. A 13.3-as ábra a 13.8-as program egy lehetséges eredményét mutatja be.

13.3. ábra

*NNTP kapcsolat
megteremtése*



Levél küldése a mail() függvénnyel

A PHP leveszi a vállunkról az elektronikus levelezés gondját is. A mail() függvény három karakterláncot vár paraméterként. Az első a címzett, a második a levél tárgya, a harmadik pedig maga az üzenet. A mail() false értékkel tér vissza, ha problémába ütközik a levél küldése során. A következő kódrészlet egy rövid levél küldését példázza:

```
$cimzett = "valaki@tartomany.hu";
$targy = "üdvözlét";
$uzenet = "ez csak egy próba üzenet! ";
mail($cimzett, $targy, $uzenet) or
    ➔ print "A levél elküldése sikertelen";
```

Ha a PHP-t UNIX rendszeren futtatjuk, a Sendmailt fogja használni, más rendszereken a helyi vagy egy távoli SMTP kiszolgálót fog feladata elvégzéséhez igénybe venni. A kiszolgálót a `php.ini` fájl SMTP utasításával kell beállítani.

Nem kell a `mail()` kötelező paraméterei által előállított fejlécekre szorítkoznunk. A függvénynek van ugyanis egy elhagyható negyedik paramétere is, mellyel szabadon alakíthatjuk az elküldendő levél fejléceit. Az ebben felsorolt fejlécso-
rokat a CRLF (`\r\n`) karakterpárral kell elválasztanunk. Az alábbi példában egy From (Feladó) és egy X-Priority (Fontosság) mezővel bővítjük a fejléct. Ez utóbbit csak bizonyos levelezőrendszerek veszik figyelembe. A példa a következő:

```
$cimzett = "valaki@tartomany.hu";
$felado = "masvalaki@masiktartomany.hu";
$targy = "üdvözlét ";
$uzenet = "ez csak egy proba üzenet! ";
mail( $cimzett, $targy, $uzenet,
    ➤ "From: $felado\r\nX-Priority: 1 (Highest)" )
or print "A levél elküldése sikertelen";
```

Összefoglalás

Ezen az órán megtudhattuk, hogyan vethetjük be a környezeti változókat, ha több adatot szeretnénk megtudni látogatóinkról. Ha nem tudjuk a látogató gépének nevét, a `gethostbyaddr()` függvénnyel kideríthetjük.

Láthattuk miként zajlik egy HTTP kapcsolat megteremtése a kiszolgáló és az ügyfél között, megtanultuk, hogyan töltsünk le egy dokumentumot a webről az `fopen()` segítségével és hogyan építsük ki saját HTTP kapcsolatunkat az `fsockopen()` függvény felhasználásával. Az `fsockopen()`-nel más hálózati szolgáltatásokhoz is kapcsolódtunk, végül pedig elektronikus levelet küldtünk a programból a `mail()` függvény egyszerű meghívásával.

Kérdések és válaszok

A HTTP meglehetősen misztikusnak tűnik. Tényleg szükség van az ismeretére, ha jó PHP kódot szeretnénk írni?

Nem. Kitűnő programok készíthetők a kiszolgáló-ügyfél párbeszéd részletes ismerete nélkül is. Másfelől azonban szükség van ilyen alapvető ismeretekre, ha kicsit bonyolultabb feladatokat szeretnénk programozottan végrehajtani, például weboldalakat letölteni.

Ha én is könnyedén küldhetek hamis fejléceket, akkor mennyire bízhatok a mások fejlécei alapján létrehozott környezeti változókkal?

Az olyan környezeti változókkal, mint a `$HTTP_REFERER` vagy a `$HTTP_USER_AGENT`, nem szabad megbíznunk, amennyiben ezen információk pontos ismerete programunkban alapvető követelmény. Az ügyfélprogramok tekintélyes hányada azonban nem hazudik: ha a böngészőtípust és az egyéb felhasználói jellemzőket olyan céllal gyűjtjük, hogy az abból készített statisztikák elemzése alapján a felhasználókat jobban szolgálhassuk, nincs értelme foglalkoznunk azzal, hogy nem minden adat feltétlenül helytálló.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Mely környezeti változó árulja el nekünk annak a lapnak a címét, amely a programunkra hivatkozott?
2. Miért nem felel meg a `$REMOTE_ADDR` változó a látogató nyomon követésére?
3. Minek a rövidítése és mit jelent a HTTP?
4. A fejléc mely sorából tudhatja meg a kiszolgáló, hogy milyen ügyfélprogramtól érkezett a kérés?
5. Mit takar a kiszolgáló 404-es válaszkódja?
6. Anélkül, hogy külön kapcsolatot építenénk fel egy kiszolgálóval, mely függvénnyel tölthetünk le egy weboldalt róla?
7. Adott IP cím alapján hogyan deríthető ki a hozzá tartozó gép neve?
8. Melyik függvény használható hálózati kapcsolat létrehozására?
9. A PHP mely függvényével küldenénk elektronikus levelet?

Feladatok

1. Készítsünk egy programot, amely egy webcímet kér be (például `http://www.kiskapu.hu/`) a felhasználótól, majd egy HEAD kérést küldve felveszi a kapcsolatot azzal. Jelenítsük meg a kapott választ a böngészőben. Ne feledkezzünk meg arról, hogy a kapcsolat nem mindig jön létre.
2. Készítsünk olyan programot, amely a felhasználónak lehetővé teszi, hogy begépeljen némi szöveget, majd elektronikus levél formájában továbbítsa azt a mi e-mail címünkre. Áruljuk el a felhasználónak, hogy a környezeti változók mit is állítanak róla.



14. ÓRA

Dinamikus képek kezelése

Az ezen az órán vett függvények a GD nevű nyílt forráskódú programkönyvtáron alapulnak.

A GD könyvtár olyan eszközcsoport, melynek segítségével futásidőben képeket hozhatunk létre és kezelhetjük azokat. A GD-ről a <http://boute11.com/gd/> weboldal szolgál bővebb információval. Ha a könyvtárat telepítettük és a PHP-t is lefordítottuk, elkezdhetünk dinamikus képeket létrehozni a PHP grafikus függvényeivel. Sok rendszer még mindig a könyvtár egy régebbi változatát futtatja, amely a képeket GIF formátumban hozza létre. A könyvtár későbbi változatai licenszokok miatt nem támogatják a GIF formátumot. Ha rendszerünkre a könyvtár egy újabb változatát telepítettük, úgy is lefuttathatjuk a PHP-t, hogy a képalkotó függvények kimeneti formátuma PNG legyen, amelyet a legtöbb böngésző támogat.

A GD könyvtár segítségével menet közben hozhatunk létre bonyolult alakzatokat a PHP grafikus függvényeivel. Az óra során a következőket tanuljuk meg:

- Hogyan hozunk létre és jelenítsünk meg egy képet?
- Hogyan dolgozunk a színekkel?
- Hogyan rajzolunk alakzatokat, például köríveket, téglalapokat és sokszögeket?

- Hogyan töltünk ki színnel alakzatokat?
- Hogyan kezeljük a TrueType betűtípusokat?

Képek létrehozása és megjelenítése

Mielőtt elkezdhetnénk a képekkel foglalkozni, először szert kell tennünk egy képzazonosítóra. Ezt az `imagecreate()` függvény segítségével tehetjük meg. Ennek két paramétere van, az egyik a kép magasságára vonatkozik, a másik a szélességére. Ekkor egy képzazonosítót kapunk vissza, amelyet az órán tárgyalt majdnem mindegyik függvény esetében használni fogunk.

Miután megkaptuk az azonosítót, már majdnem készen állunk arra, hogy megjelenítsük első képünket a böngészőben. Ehhez az `imagegif()` függvényre van szükségünk, amelynek paramétere a képzazonosító. A 14.1. programban a kép létrehozásához és megjelenítéséhez használt függvényeket láthatjuk.

14.1. program Dinamikusan létrehozott kép

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: imagegif($kep);
5: ?>
```

Fontos, hogy a `Content-type` fejlécsort minden egyéb előtt küldtük a böngészőnek. Közölnünk kell a böngészővel, hogy egy képet küldünk, különben a program kimenetét HTML-ként kezeli. A programot a böngésző már közvetlenül vagy egy `IMG` elem részeként hívhatja meg.

```

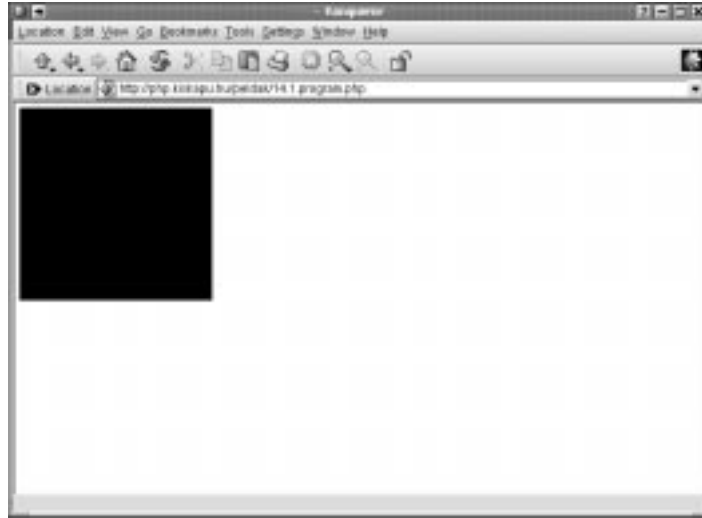
```

A 14.1. ábrán láthatjuk a 14.1. program kimenetét.

Egy téglalapot hoztunk létre, de egyelőre nem tudjuk annak színét beállítani.

14.1. ábra

Dinamikusan létrehozott alakzat



A szín beállítása

A szín beállításához egy színazonosítóra van szükségünk. Ezt az `imagecolorallocate()` függvénnyel kaphatjuk meg, amelynek paramétere a képezonosító, illetve három 0 és 255 közötti egész szám, amelyek a vörös, zöld és kék színösszetevőket jelentik. Egy színazonosítót kapunk vissza, amellyel később alakzatok, kitöltések vagy szövegek színét határozhatjuk meg.

```
$piros = imagecolorallocate($kep, 255,0,0);
```

Az `imagecolorallocate()` függvény első meghívásakor egyúttal átállítja a paraméterül kapott alakzat színét is. Így ha az előző kódrészletet hozzáadjuk a 14.1. programhoz, egy vörös téglalapot kapunk.

Vonalak rajzolása

Mielőtt egy képre vonalat rajzolhatnánk, meg kell adnunk annak két végpontját.

A képet úgy képzeljük el, mint képernyőpontokból álló tömböt, melynek mindkét tengelye 0-tól számozódik. A számozás kezdőpontja a kép bal felső sarka.

Más szóval az (5, 8) koordinátájú képpont a hatodik képpont fentről lefelé és a kilencedik képpont balról jobbra számolva.

Az `imageline()` függvény két képpont közé rajzol egy egyenest. A függvény paraméterként egy képazonosítót vár, négy egész számot, amelyek az egyenes két végpontjának koordinátáit jelentik, valamint egy színazonosítót.

Ha a 14.2. programban megrajzoljuk a téglalap egy átlóját.

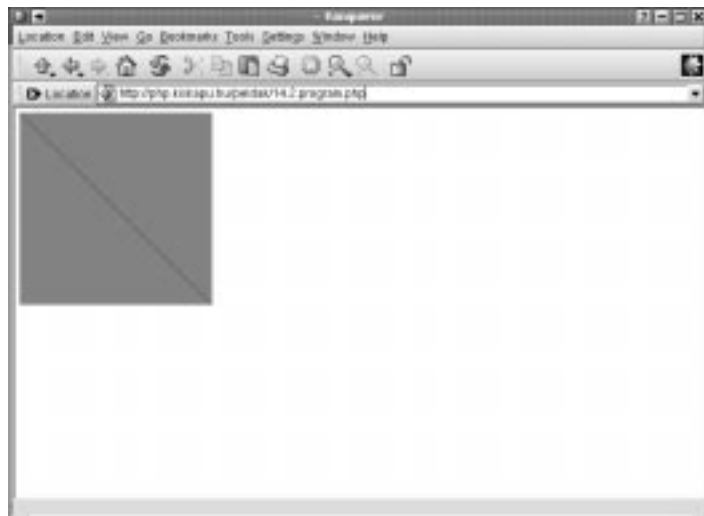
14.2. program Egyenes rajzolása az `imageline()` függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imageline( $kep, 0, 0, 199, 199, $kek );
7: imagegif($kep);
8: ?>
```

Két színazonosítót kapunk, egyet a piros és egyet a kék színnek. Ezután a `$kek` változóban tárolt azonosítót használjuk a vonal színének megadására. Fontos megjegyeznünk, hogy a vonal a (199, 199) és nem a (200, 200) koordinátákban végződik, tekintve, hogy a képpontokat 0-tól kezdve számozzuk. A 14.2. ábrán a 14.2. program eredményét láthatjuk.

14.2. ábra

*Egyenes rajzolása
az `imageline()`
függvénnyel*



Alakzatok kitöltése

A PHP segítségével ugyanúgy kiszínezhetünk alakzatokat, mint kedvenc grafikai programunkkal. Az `imagefill()` függvény bemenete egy képazonosító, a kitöltés kezdőkoordinátái, valamint egy színazonosító. A függvény a kezdő képponttal megegyező színű szomszédos képpontokat a kijelölt színre állítja. A 14.3. program a képet az `imagefill()` függvény meghívásával teszi egy kicsit érdekesebbé.

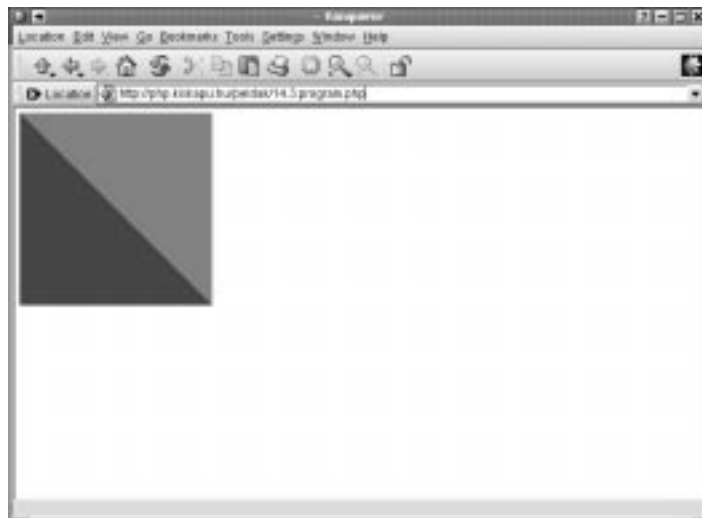
14.3. program Az `imagefill()` függvény használata

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imageline( $kep, 0, 0, 199, 199, $kek );
7: imagefill( $kep, 0, 199, $kek );
8: imagegif($kep);
9: ?>
```

A 14.3. ábrán a 14.3. program kimenetét láthatjuk.

14.3. ábra

Az `imagefill()` függvény használata



Körív rajzolása

Az `imagearc()` függvény segítségével köríveket rajzolhatunk. Bemenete egy képazonosító, a középpont koordinátája, a szélességet meghatározó egész szám, a magasságot meghatározó egész szám, egy kezdő- és egy végpont (fokban mérve), valamint egy színazonosító. A köríveket az óramutató járásának megfelelően, 3 órától kezdve rajzoljuk. A következő kódrészlet egy negyed körívet rajzol ki:

```
imagearc( $kep, 99, 99, 200, 200, 0, 90, $kek );
```

Ez egy olyan körívrészletet jelenít meg, melynek középpontja a (99, 99) koordinátájú pontban van. A teljes magasság és szélesség 200-200 képpontnyi. A körív 3 óránál kezdődik és 90 fokot rajzolunk (azaz 6 óráig).

A 14.4. program teljes kört rajzol és kék színnel tölti ki.

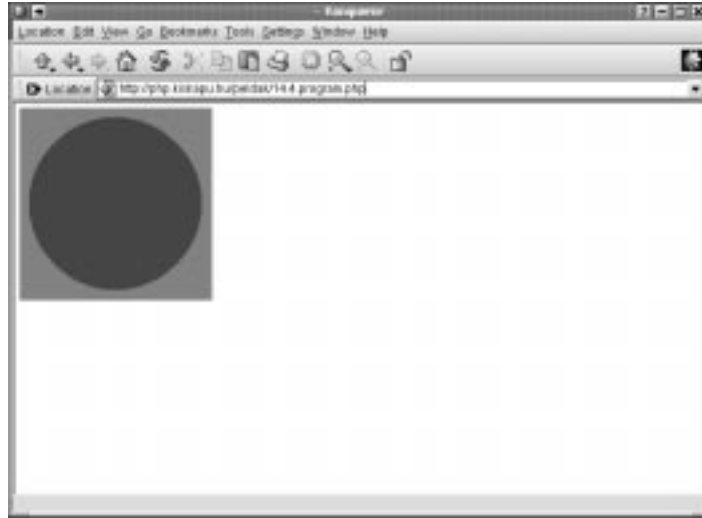
14.4. program Kör rajzolása az `imagearc()` függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imagearc( $kep, 99, 99, 180, 180, 0, 360, $kek );
7: imagefill( $kep, 99, 99, $kek );
8: imagegif($kep);
9: ?>
```

A 14.4. program kimenetét a 14.4. ábrán láthatjuk.

14.4. ábra

*Kör rajzolása
az `imagearc()` függ-
vényrel*



Téglalap rajzolása

A PHP `imagerectangle()` függvényével téglalapot rajzolhatunk.

Az `imagerectangle()` bemenete egy képazonosító, a téglalap bal felső és jobb alsó sarkának koordinátája, valamint egy színazonosító. A következő kódrészlet olyan téglalapot rajzol, melynek bal felső és jobb alsó koordinátái rendre (19, 19) és (179, 179):

```
imagerectangle( $kep, 19, 19, 179, 179, $kek );
```

Az alakzatot ezután az `imagefill()` függvénnyel tölthetjük ki. Mivel ez meglehetősen gyakori művelet, a PHP-ban létezik az `imagefilledrectangle()` függvény, amely ugyanazokat a bemeneti értékeket várja, mint az `imagerectangle()`, de az általunk meghatározott színnel ki is tölti a téglalapot. A 14.5. program egy kiszínezett téglalapot hoz létre és megjeleníti a böngészőben.

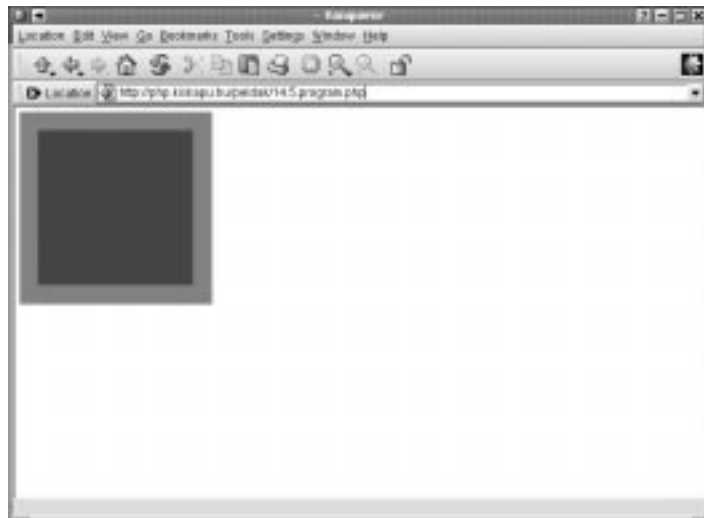
14.5. program Téglalap rajzolása az imagefilledrectangle() függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imagefilledrectangle( $kep, 19, 19, 179, 179, $kek );
7: imagegif( $kep );
8: ?>
```

A 14.5. ábrán a 14.5. program által előállított képet láthatjuk.

14.5. ábra

*Téglalap rajzolása az
imagefilledrectangle()
függvénnyel*



Sokszög rajzolása

Az `imagepolygon()` függvény segítségével kifinomultabb alakzatokat is rajzolhatunk. E függvény bemenete egy képazonosító, a pontok koordinátáiból álló tömb, az alakzat pontjainak számát jelző egész szám és egy színazonosító.

Az `imagepolygon()` bemenetét képező tömbnek számmal indexeltnek kell lennie. Az első két elem az első pont koordinátáit adja meg, a második kettő a második ponttét és így tovább. Az `imagepolygon()` függvény megrajzolja a pontok közötti vonalakat és az utolsó pontot automatikusan összeköti az elsővel. Az `imagefilledpolygon()` függvény segítségével színnel kitöltött sokszögek hozhatók létre.

A 14.6. program egy kitöltött sokszöget rajzol és jelenít meg a böngészőben.

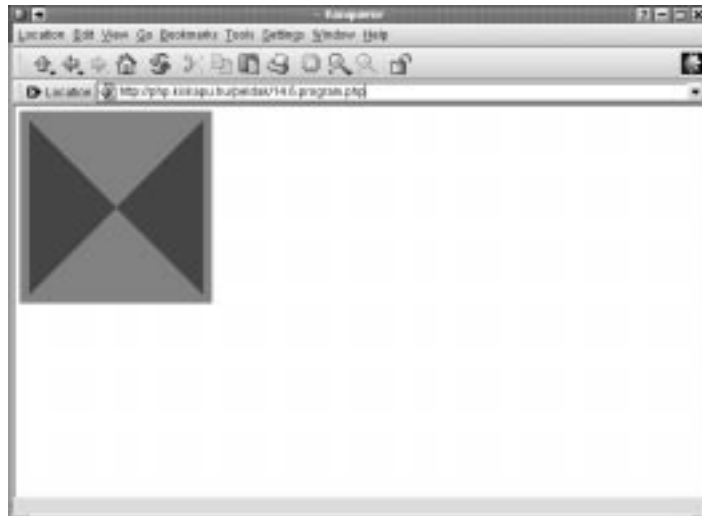
14.6. program Sokszög rajzolása az imagefilledpolygon() függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: $pontok = array (10, 10,
7:                 190, 190,
8:                 190, 10,
9:                 10, 190
10:                );
11: imagefilledpolygon( $kep, $pontok, count
12:                   ( $pontok )/2, $kek );
13: imagegif($kep);
14: ?>
```

Vegyük észre, hogy az összekötendő pontok száma az imagefilledpolygon() függvény hívásakor megegyezik a \$pontok tömb elemei számának felével. A 14.6. ábrán a 14.6. program eredményét láthatjuk.

14.6. ábra

*Sokszög rajzolása az
imagefilledpolygon()
függvénnyel*



A színek átlátszóvá tétele

A PHP az `imagecolortransparent()` függvénnyel lehetővé teszi, hogy a kiválasztott színeket az ábrán belül áttetszővé tegyük. A függvény bemenete egy kép- és egy színazonosító. Ha a képet megjelenítjük egy böngészőben, az `imagecolortransparent()` függvénynek átadott szín áttetsző lesz. A 14.7. program kimenete a sokszöget rajzoló programot úgy változtatja meg, hogy az alakzat „úszni” fog a böngészőben, ahelyett, hogy egy háttérszín előtt jelenne meg.

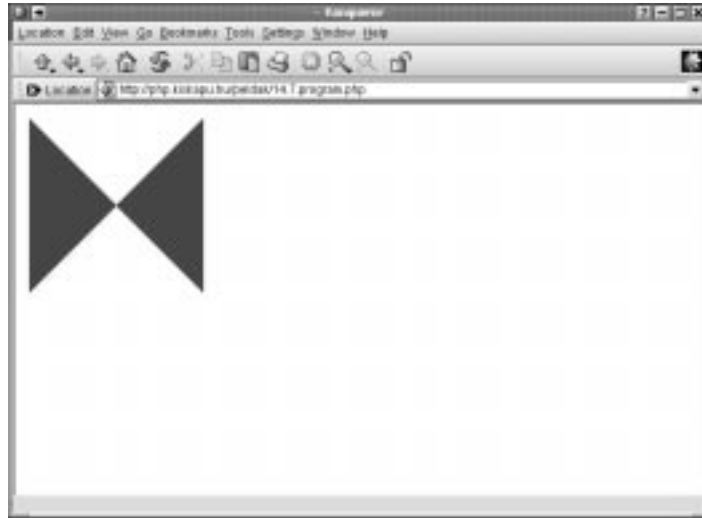
14.7. program A színek átlátszóvá tétele az `imagecolortransparent()` függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3:
4: $kep = imagecreate( 200, 200 );
5: $piros = imagecolorallocate($kep, 255,0,0);
6: $kek = imagecolorallocate($kep, 0,0,255 );
7:
8: $pontok = array (10, 10,
9:                 190, 190,
10:                190, 10,
11:                10, 190
12:                );
13:
14: imagefilledpolygon( $kep, $pontok, count
                    ( $pontok )/2, $kek );
15: imagecolortransparent( $kep, $piros );
16: imagegif($kep);
17: ?>
```

A 14.7. ábrán a 14.7. program kimenetét láthatjuk.

14.7. ábra

*A színek átlátszósá-
tétele az `image-
colortransparent()`
függvénnyel*



Szövegek kezelése

Ha rendszerünkön vannak TrueType betűk, a képekre írhatunk is. A GD programkönyvtár mellett szükségünk lesz a FreeType programkönyvtár telepítésére is. Ha ezek rendelkezésünkre állnak, egy képen megjelenő diagramokat és navigációs elemeket is létrehozhatunk. A PHP biztosít számunkra egy olyan eszközt is, amellyel ellenőrizhetjük, hogy a beírandó szöveg elfér-e a rendelkezésre álló helyen.

Szövegírás az `imageTTFtext()` függvénnyel

Az `imageTTFtext()` függvénnyel az ábrákra szöveget írhatunk. A függvény nyolc bemenő paramétere a következő: képazonosító, méret, amely a kiírandó szöveg magasságára utal, szög, a kezdő *x* és *y* koordináták, színazonosító, a TrueType betűtípus elérési útvonala és a kiírandó szöveg.

A kiírandó szöveg kezdőkoordinátája határozza meg, hol lesz a szöveg első karakterének alapvonala.

A 14.8. program egy karakterláncot ír az ábrára, majd az eredményt megjeleníti a böngészőben.

14.8. program Szövegírás az imageTTFtext() függvénnyel

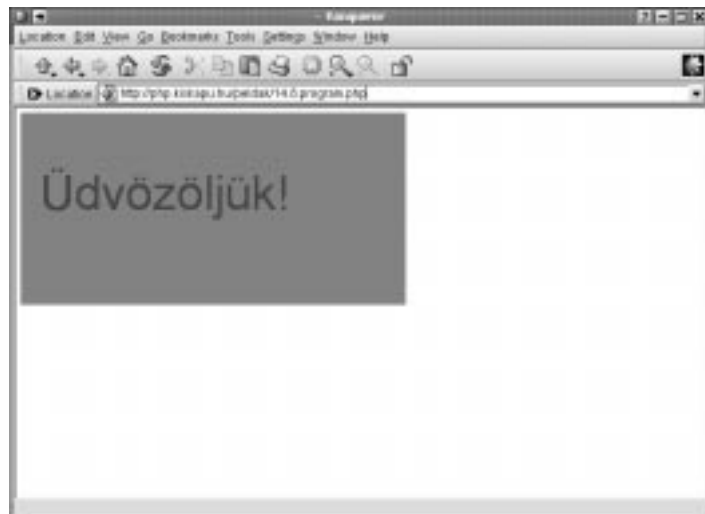
```
1: <?php
2: header("Content-type: image/gif");
3:
4: $kep = imagecreate( 400, 200 );
5: $piros = imagecolorallocate($kep, 255,0,0);
6: $kek = imagecolorallocate($kep, 0,0,255 );
7: $betukeszlet = "/usr/local/office52/share/fonts/
    /TrueType/arial.ttf";
8:
9: imageTTFtext( $kep, 50, 0, 20, 100, $kek,
    $betukeszlet, "Üdvözöljük!" );
10:
11: imagegif($kep);
12: ?>
```

Létrehozunk egy 400 képpontnyi széles és 200 képpontnyi magas vásznat, megadunk két színt és a `$betukeszlet` változóban tároljuk a TrueType betűtípus elérési útját. Ezután ráírjuk az ábrára a "Üdvözöljük!" szöveget. Figyelem, a betűtípusok a kiszolgálón feltehetőleg más könyvtárban találhatóak! Ha nem vagyunk biztosak a helyben, keressük a `.ttf` kiterjesztésű fájlokat.

Az `imageTTFtext()` meghívásához az 50-es magasságot, 0 fokos szöveget és (20, 100) kezdőkoordinátát adtuk át. Emellett a függvény megkapja még a `$kek` változóban tárolt színazonosítót és a `$betukeszlet` változóban tárolt betűtípust, végül a kiírandó szöveget. Az eredményt a 14.8. ábrán láthatjuk.

14.8. ábra

*Szöveg írása az
imageTTFtext() függ-
vénnyel*



Persze most még csak találgatunk, hová tegyük a szöveget. A betűméret nem árul el sokat a szöveg magasságáról, a szélességéről még kevesebbet.

Az `imageTTFtext()` függvény persze visszaad információt a szöveg kiterjedéséről, de akkorra már minden eldőlt. Szerencsére a PHP lehetőséget nyújt arra, hogy próbálkozzunk, mielőtt még élesben elvégeznénk a műveletet.

Szöveg kiterjedésének ellenőrzése az `imageTTFbox()` függvénnyel

A szöveg kiterjedéséről az `imageTTFbox()` függvénnyel szerezhetünk információt, amely onnan kapta nevét, hogy a szöveget határoló dobozt írja le. A függvény bemenete a betűméret, a szög, a betűtípus elérési útja és a kérdéses szöveg.

Azon kevés függvények egyike, melyekhez nem kell a képazonosító. Egy nyolc-elemű tömböt ad vissza, amely a szöveg kiírásához szükséges terület paramétereit (koordinátáit) tartalmazza. A 14.1. táblázatban a függvény által visszaadott tömböt értelmezzük.

14.1. táblázat Az `imageTTFbox()` függvény által visszaadott tömb

Sorszám	Leírás
0	bal alsó (vízszintes tengely)
1	bal alsó (függőleges tengely)
2	jobb alsó (vízszintes tengely)
3	jobb alsó (függőleges tengely)
4	jobb felső (vízszintes tengely)
5	jobb felső (függőleges tengely)
6	bal felső (vízszintes tengely)
7	bal felső (függőleges tengely)

A függőleges tengelyhez visszaadott számok (1-es, 3-as, 5-ös, és 7-es tömbelemek) a szöveg alapvonalához viszonyulnak, amelynek koordinátája 0. A szöveg tetejének a függőleges tengelyen mért értékei ehhez képest általában negatív számok, míg a szöveg aljának itt mért értékei általában pozitívak. Így kapjuk meg, hány képponttal nyúlik a szöveg az alapvonal alá. Az `imageTTFbox()` által figyelembe vett koordinátarendszert úgy kell elképzelnünk, mintha a szöveg alapvonala lenne az y irányban vett 0 érték és a negatív koordináták attól felfelé, a pozitívak pedig attól lefelé helyezkednének el. Így például ha az `imageTTFbox()` függvénnyel ellenőrizendő szövegben van egy "y", a visszakapott tömbben az 1-es indexű elem lehet, hogy 3 lesz, mivel az "y" alsó szára három képponttal nyúlik

az alapvonal alá. Lehetséges, hogy a 7-es elem -10 lesz, mivel a szöveg 10 képponttal van az alapvonal fölé emelve. A pontos értékek betűtípustól és betűmérettől függenek.

Csak hogy bonyolítsuk a helyzetet, úgy tűnik, hogy az `imageTTFbox()` függvény által visszaadott alapvonal és a rajzolás közben látható között 2 képpontos eltolás van. Ezt úgy ellensúlyozhatjuk, hogy az alapvonalat két képpontnyival magasabbnak képzeljük, mint amilyennek az `imageTTFbox()` függvény mutatja.

A vízszintes tengelyen a bal oldali számok (a 0-ás és a 6-os elem) esetében az `imageTTFbox()` függvény az adott kezdőpont előtt kezdődő szöveget negatív számmal kifejezett eltolással jelöli. Ez általában alacsony szám, mivel a betűk jellemzően jelentősebb eltérést mutatnak az alapvonalától, mint az x irányú kezdőkoordinátától (gondoljunk a p, g vagy j betűkre). A vízszintes tengelyen így már csak az elvárt pontosság kérdése, hogy szükségünk van-e a koordináták kiigazítására.

Az `imageTTFbox()` függvény által visszaadott értékeket arra is használhatjuk, hogy az ábrán belül a szöveget igazítsuk. A 14.9. program dinamikusan küldi a böngészőnek a szöveget, úgy, hogy mind a vízszintes, mind a függőleges tengelyen középre helyezi azt.

14.9. program Szöveg elhelyezése az `imageTTFbox()` függvény használatával

```
1: <?php
2: header("Content-type: image/gif");
3: $magassag = 100;
4: $szelesseg = 200;
5: $betumeret = 50;
6: if ( ! isset ( $szoveg ) )
7:     $szoveg = "Változtasson meg!";
8: $kep = imagecreate( $szelesseg, $magassag );
9: $piros = imagecolorallocate($kep, 255,0,0);
10: $kek = imagecolorallocate($kep, 0,0,255 );
11: $betukeszlet = "/usr/local/office52/share/fonts/
    /TrueType/arial.ttf";
12: $szovegszelesseg = $szelesseg;
13: $szovegmagassag;
14: while ( 1 )
15:     {
16:         $doboz = imageTTFbox( $betumeret, 0,
            $betukeszlet, $szoveg );
17:         $szovegszelesseg = abs( $doboz[2] );
18:         $szovegmagassag = ( abs($doboz[7]) )-2;
```

14.9. program (folytatás)

```
19:      if ( $szovegszelesseg < $szelesseg - 20 )
20:          break;
21:      $betumeret--;
22:  }
23:  $gifXkozep = (int) ( $szelesseg/2 );
24:  $gifYkozep = (int) ( $magassag/2 );
25:  imageTTFtext(      $kep, $betumeret, 0,
26:                (int) ($gifXkozep-($szovegszelesseg/2)),
27:                (int) ($gifYkozep+($szovegmagassag/2)),
28:                $kek, $betukeszlet, $szoveg );
29:  imagegif($kep);
30:  ?>
```

A kép magasságát és szélességét a \$magassag és \$szelesseg változóban tároljuk, a betűtípus alapbeállítású méretét pedig 50-re állítjuk. Megvizsgáljuk, hogy létezik-e már a \$szoveg nevű változó; ha nem, beállítjuk egy alapértékre. Ezzel az eljárással egy ábrának is lehetnek paraméterei, fogadhat adatot egy weboldaltól – akár egy URL keresőkifejezéséből, akár egy kitöltött űrlapról. A képzazonosító előállításához az imagecreate() függvényt használjuk. A színazonosítókat a szokásos módon állítjuk elő, a \$betukeszlet nevű változóban pedig a TrueType betűtípus elérési útját rögzítjük.

Azt szeretnénk, hogy a \$szoveg változóban tárolt karaktersorozat elférjen a rendelkezésére álló helyen, de egyelőre nem tudhatjuk, hogy ez sikerülni fog-e. A while utasításon belül átadjuk a betűtípus elérési útját és a szöveget az imageTTFbox() függvénynek, az eredményül kapott tömböt pedig a \$doboz változóba helyezzük. A \$doboz[2] tartalmazza a jobb alsó sarok helyét a vízszintes tengelyen. Ezt vesszük a karaktersorozat szélességének és a \$szovegszelesseg változóban tároljuk.

A szöveget függőlegesen is középre szeretnénk helyezni, de a szöveg alapvonala alatt lévő részt figyelmen kívül hagyva. Az alapvonal feletti magasság meghatározásához használhatjuk a \$doboz[7] változóban található szám abszolút értékét, bár ezt még ki kell igazítanunk a fent említett két képponttal. Ezt az értéket a \$szovegmagassag változóban tároljuk.

Most, hogy kiszámítottuk a szöveg szélességének értékét, összevethetjük az ábra szélességével (tíz képpontnyiival kell kevesebbnek lennie, a keret miatt). Ha a szöveg keskenyebb, mint a kép szélessége, befejezzük a ciklust. Ellenkező esetben csökkentjük a betűméretet és újra próbálkozunk.

A \$magassag és a \$szelesseg változók értékét elfelelve megkapjuk az ábra hozzávetőleges középpontját. A szöveget az ábra középpontjából és a szöveg magasságából, illetve szélességéből kiszámított eltolással az ábrára írjuk, végül megjelenítjük a képet a böngészőben. A 14.9. ábrán láthatjuk a 14.9. program ki menetét.

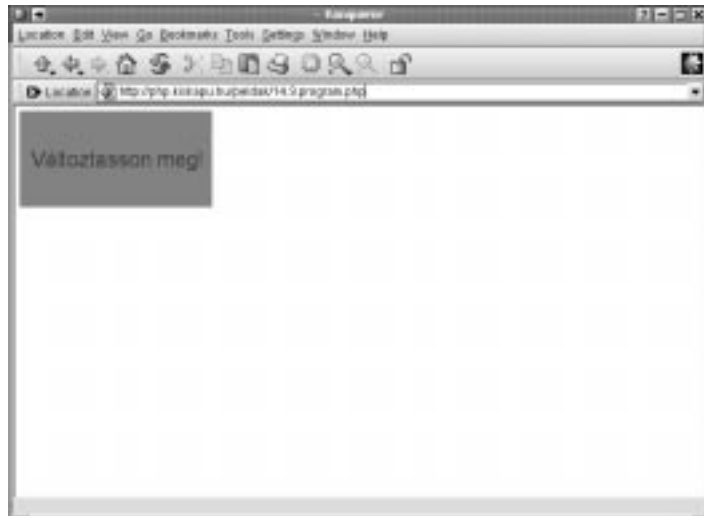
Ezt a kódot egy másik oldal is meghívhatja, egy IMG elem részeként. A következő részlet egy olyan programot hoz létre, amely lehetővé teszi az egyes felhasználóknak, hogy saját szöveget jelenítsenek meg az ábrán:

```
1: <?php
2: if ( ! isset( $szoveg ) )
3:     $szoveg = "Dinamikus szöveg!";
4: ?>
5: <form action="<? print $PHP_SELF ?>" method="POST">
6: <input type="text" name="szoveg">
7: </form>
8: <p>
9: ">
```

Amikor a 14.9. programot meghívjuk, egy olyan keresőkifejezéssel egészítjük ki a hívást, amely tartalmazza a megjelenítendő szöveget. A tizenkilencedik órában többet is tanulhatunk az információk programról programra való átadásáról.

14.9. ábra

*Szöveg elhelyezése az
imageTTFbox() függ-
vény használatával*



A fenti elemek összegyűrése

Készítsünk egy példát az ezen az órán tanult függvények használatával! Tegyük fel, hogy arra kértek bennünket, hogy készítsünk egy dinamikus oszlopgrafikont, amely több címkézett számot hasonlít össze. Az egyes oszlopok alatt a megfelelő címkének kell lennie. Ügyfelünknek tudnia kell módosítani a grafikon oszlopainak számát, a kép szélességét és magasságát és a grafikon körülvevő keret méretét. A grafikon fogyasztói szavazáshoz fogják használni, így az adatokat csak látszólagos pontossággal kell megjeleníteni. Egy sokkal részletesebb lebontást az ábrát tartalmazó oldal HTML részében találhatunk.

Az oszlopokat és értékeiket a legegyszerűbb módon egy asszociatív tömbben tárolhatjuk. Miután megvan ez a tömb, ki kell számolnunk az oszlopok számát és a tömb legnagyobb értékét:

```
$cellak = array ( "tetszik"=>200, "nemetszik"=>400,  
                 "közömbös"=>900 );  
$max = max( $cellak );  
$cellaszam = count( $cellak );
```

Létre kell hoznunk néhány változót, így ügyfelünk egyéni igényeihez igazíthatja az ábrát:

```
$teljesszelesseg = 400;  
$teljesmagassag = 200;  
$xmargo = 20; // jobb és bal margó  
$ymargo = 20; // felső és alsó margó  
$oszlopkoz = 5; // az oszlopok közötti hely  
$alsokeret = 40; // az ábra alján kimaradó hely  
    ➤ (a margót nem számolva)  
$betukeszlet = "/usr/local/office52/share/fonts/  
    ➤ /TrueType/arial.ttf";
```

Ügyfelünk a változók módosításával meghatározhatja az ábra magasságát és szélességét. Az `$xmargo` és `$ymargo` a grafikon körüli margóvastagságot adja meg. Az `$oszlopkoz` az oszlopok közötti távolságot határozza meg, az `$alsokeret` változó pedig az oszlopok alatti címkéknek szánt hely nagyságát.

Most hogy megvannak ezek az értékek, a némi számolgatás árán belőlük kapott hasznos számokat változókba tesszük:

```

$vaszonszelesseg = ( $teljesszelesseg - $xmargo*2 );
$vaszonmagassag = ( $teljesmagassag - $ymargo*2 - $alsokeret
);
$xPoz = $xmargo; // a rajzolás kiindulópontja az x tengelyen
$yPoz = $teljesmagassag - $ymargo - $alsokeret; //
    ➔ a rajzolás kiindulópontja az y tengelyen
$cellaszelesseg = (int) (( $vaszonszelesseg - ( $oszlopkoz *
    ➔ ( $cellaszam-1 ) )) / $cellaszam) ;
$szovegmeret = (int)($alsokeret);

```

Kiszámítjuk a rajzfelületet (azt a területet, ahová majd az oszlopok kerülnek). A vízszintes összetevőt úgy kapjuk meg, hogy a teljes szélességből kivonjuk a margó kétszeresét. A függőleges mérethez az \$alsokeret változót is számításba kell vennünk, hogy a címkéknek helyet hagyjunk. Az \$xPoz tárolja azt az x koordinátát, amelytől az oszlopok rajzolását kezdjük, így ennek értékét az \$xmargo változó értékére állítjuk, amely a margó értékének vízszintes összetevőjét tárolja. Az oszlopok talppontját tároló \$yPoz változó értékét úgy kapjuk meg, ha az ábra teljes magasságából kivonjuk a margót és a címkéknek kihagyott hely nagyságát, amelyet az \$alsokeret változóban tároltunk.

A \$cellaszelesseg az egyes oszlopok szélességét tárolja. Ahhoz, hogy az értékét megkapjuk, számoljuk ki az oszlopok közötti helyek összegét, ezt vonjuk ki a diagram szélességéből, majd az eredményt osszuk el az oszlopok számával.

Kezdetben úgy állítjuk be a szöveg méretét, hogy egyezzen a szöveg számára szabadon maradt terület magasságával (amit az \$alsokeret változóban tárolunk).

Mielőtt megkezdénénk a munkát az ábrával, meg kell határoznunk a szöveg méretét. Viszont nem tudjuk, milyen szélesek lesznek a címkék és mindenképpen azt szeretnénk, hogy az egyes címkék elférjenek a felettük levő oszlop szélességén belül. A \$cellak tömbön egy ciklus segítségével meghatározzuk a legnagyobb címke méretét:

```

foreach( $cellak as $kulcs => $ertek )
{
    while ( 1 )
    {
        $doboz = ImageTTFbBox( $szovegmeret, 0,
                                $betukeszlet, $kulcs );
        $szovegszelesseg = $doboz[2];
        if ( $szovegszelesseg < $cellaszelesseg )
            break;
        $szovegmeret--;
    }
}

```

Minden elem esetében egy ciklust kezdve, az `imageTTFbox()` segítségével információt szerzünk a címke méretét illetően. A kapott értéket a `$doboz[2]` változóban találhatjuk meg, amit összevetünk a `$cellaszelesseg` változóval, amely egy oszlop szélességét tartalmazza. A ciklust abban az esetben állítjuk le, ha a szöveg szélessége kisebb, mint az oszlopszélesség. Egyébként pedig csökkentjük a `$szovegmeret` értékét és újra ellenőrizzük a méreteket. A `$szovegmeret` addig csökken, míg a tömb összes címkéje kisebb nem lesz az oszlopszélességnél.

Most már végre létrehozhatunk egy képazonosítót és elkezdhetünk dolgozni vele.

```
$kep = imagecreate( $teljesszelesseg, $teljesmagassag );
$piros = ImageColorAllocate($kep, 255, 0, 0);
$kek = ImageColorAllocate($kep, 0, 0, 255 );
$fekete = ImageColorAllocate($kep, 0, 0, 0 );
foreach( $cellak as $kulcs => $ertekek )
{
    $cellamagassag = (int) (($ertekek/$max) * $vaszonmagassag);
    $kozeppont = (int) ($xPoz+($cellaszelesseg/2));
    imagefilledrectangle( $kep, $xPoz, ($yPoz-$cellamagassag),
        ($xPoz+$cellaszelesseg), $yPoz, $kek );
    $doboz = ImageTTFBox( $szovegmeret, 0, $betukeszlet,
        $kulcs );
    $szovszel = $doboz[2];
    ImageTTFText( $kep, $szovegmeret, 0, ($kozeppont-
        ($szovszel/2)),
        ($teljesmagassag-$ymargo), $fekete, $betukeszlet,
        $kulcs );
    $xPoz += ( $cellaszelesseg + $oszlopkoz);
}
imagegif( $kep );
```

Ezt az `imagecreate()` függvénnyel tesszük meg, majd kiosztunk néhány színt is. Újra végiglépkedünk a `$cellak` tömbön. Kiszámítjuk az oszlop magasságát és az eredményt a `$cellamagassag`-ba helyezzük. Kiszámoljuk a középpont x koordinátáját, amely megegyezik az `$xPoz` és a fél oszlop szélességének összegével.

Az `imagefilledrectangle()` függvénynek az `$xPoz`, `$yPoz`, `$cellamagassag` és `$cellaszelesseg` változókat átadva megrajzoljuk az oszlopot.

A szöveg középre igazításához megint szükségünk van az `imageTTFbox()` függvényre, amelynek eredménytömbjét a `$doboz` változóban tároljuk. A `$doboz[2]` értéket fogjuk munkaszélességnek választani, ezt bemásoljuk a `$szovszel` átmeneti változóba. Most már elegendő információ áll rendelkezésünkre ahhoz, hogy kiírassuk a címkét. Az `x` koordináta a `$kozeppont` változó és a szövegszélesség felének különbsége lesz, az `y` koordináta pedig az alakzat magasságának és a margónak a különbsége.

Megnöveljük az `$xPoz` változót, felkészülve ezzel a következő oszlop feldolgozására.

A ciklus végeztével megjelenítjük az elkészült képet.

A teljes programot a 14.10. példában láthatjuk, a végeredményt pedig a 14.10. ábrán.

14.10. program Egy dinamikus oszlopdiagram

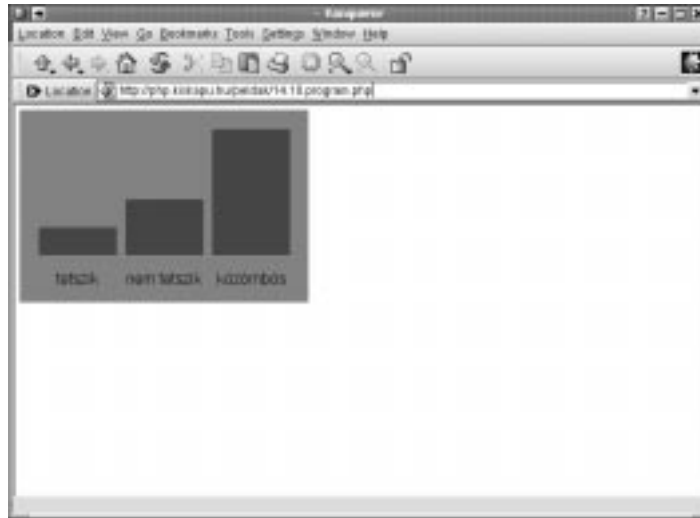
```
1: <?php
2: header("Content-type: image/gif");
3: $cellak = array ( "tetszik"=>200,
                   "nem tetszik"=>400,
                   "közömbös"=>900 );
4: $max = max( $cellak );
5: $cellaszam = count( $cellak );
6: $teljesszelesseg = 300;
7: $teljesmagassag = 200;
8: $xmargo = 20; // jobb és bal margó
9: $ymargo = 20; // fenti és lenti margó
10: $oszlopkoz = 10; // az oszlopok közötti hely
11: $alsokeret = 30; // az ábra alján kimaradó hely
    (a margót nem számolva)
12: $betukeszlet = "/usr/local/office52/share/fonts/
    /TrueType/arial.ttf";
13: $vaszonszelesseg = ( $teljesszelesseg - $xmargo*2 );
14: $vaszonmagassag = ( $teljesmagassag - $ymargo*2
    - $alsokeret );
15: $xPoz = $xmargo; // a rajzolás kiindulópontja
    az x tengelyen
16: $yPoz = $teljesmagassag - $ymargo - $alsokeret; //
    a rajzolás kiindulópontja az y tengelyen
17: $cellaszelesseg = (int) (( $vaszonszelesseg -
    ( $oszlopkoz * ( $cellaszam-1 ) )) / $cellaszam) ;
```

14.10. program (folytatás)

```
18: $szovegmeret = (int)($salsokeret);
19: // betűméret kiigazítása
20: foreach( $cellak as $kulcs => $ertek )
21:     {
22:         while ( 1 )
23:         {
24:             $doboz = ImageTTFbBox( $szovegmeret, 0,
                                     $betukeszlet, $kulcs );
25:             $szovegszelesseg = abs( $doboz[2] );
26:             if ( $szovegszelesseg < $cellaszelesseg )
27:                 break;
28:             $szovegmeret--;
29:         }
30:     }
31: $kep = imagecreate( $teljesszelesseg,
                     $teljesmagassag );
32: $piros = ImageColorAllocate($kep, 255, 0, 0);
33: $kek = ImageColorAllocate($kep, 0, 0, 255 );
34: $fekete = ImageColorAllocate($kep, 0, 0, 0 );
35: $szurke = ImageColorAllocate($kep, 100, 100, 100 );
36:
37: foreach( $cellak as $kulcs => $ertek )
38:     {
39:         $cellamagassag = (int) (($ertek/$max) *
                                $vaszonmagassag);
40:         $kozeppont = (int)($xPoz+($cellaszelesseg/2));
41:         imagefilledrectangle( $kep, $xPoz,
                                ($yPoz-$cellamagassag),
                                ($xPoz+$cellaszelesseg),
                                $yPoz, $kek );
42:         $doboz = ImageTTFbBox( $szovegmeret, 0,
                                $betukeszlet, $kulcs );
43:         $szovszel = $doboz[2];
44:         ImageTTFText( $kep, $szovegmeret, 0,
                        ($kozeppont-($szovszel/2)),
45:                        ($teljesmagassag-$ymargo), $fekete,
                        $betukeszlet, $kulcs );
46:         $xPoz += ( $cellaszelesseg + $oszlopkoz);
47:     }
48: imagegif( $kep );
49: ?>
```

14.10. ábra

Egy dinamikus oszlopdiagram



Összefoglalás

A GD programkönyvtár PHP-s támogatása lehetővé teszi, hogy dinamikus oszlopdiagramokat és navigációs elemeket hozhassunk létre, viszonylag könnyen.

Ezen az órán megtanultuk, hogyan használjuk az `imagecreate()` és az `imagegif()` függvényeket képek létrehozására és megjelenítésére. Azt is megtanultuk, hogyan szerezzünk színazonosítót és hogyan töltünk ki színnel területeket az `imagecolorallocate()` és az `imagefill()` függvényekkel. Megnéztük, hogyan használjunk vonal- és alakzat függvényeket alakzatok körvonalainak megrajzolására és kitöltésére. Megtanultuk, hogyan használjuk a PHP által támogatott FreeType programkönyvtárat a TrueType betűtípusokkal való munkára, valamint elemeztünk egy programot, amelyben egy alakzatra szöveget írtunk és megnéztünk egy példát, amelyben a tanult eljárásokat a gyakorlatban is láthattuk.

Kérdések és válaszok

Fellépnek-e teljesítménybeli problémák dinamikus képek használata esetén?

Egy dinamikusan létrehozott kép lassabban töltődik be a böngészőbe, mint egy már létező fájl. A program hatékonyságától függően ezt a felhasználó nem feltétlenül veszi észre, de azért a dinamikus képeket módjával használjuk.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Milyen fejlécsort kell a böngészőnek küldenünk a GIF kép létrehozása és megjelenítése előtt?
2. Milyen függvényt használnánk egy képazonosító előállításához, amelyet a többi függvéynél használhatunk?
3. Milyen függvényt használnánk a létrehozott GIF kép megjelenítésére?
4. Milyen függvény segítségével szerezhetjük meg a színazonosítót?
5. Melyik függvénnyel rajzolnánk vonalat egy dinamikus képre?
6. Melyik függvénnyel töltenénk ki színnel egy dinamikus alakzatot?
7. Melyik függvényt használhatjuk körív rajzolására?
8. Hogyan rajzoljunk téglalapot?
9. Hogyan rajzoljunk sokszöget?
10. Melyik függvénnyel írunk egy dinamikus alakzatra (a FreeType program-könyvtár segítségével)?

Feladatok

1. Írjunk egy programot, amely egy folyamatjelzőt hoz létre, amely mutatja, mennyi pénz folyt be egy gyűjtés alkalmával az adott célra szükséges pénzből.
2. Írjunk egy programot, amely egy főcímet ábrázoló képet ír ki egy bejövő űrlap vagy lekérdezés adataiból. Tegyük lehetővé, hogy a felhasználó határozza meg a rajzterületet, az előtér és a háttér színét, valamint az árnyék meglétét és annak méretét.



15. ÓRA

Dátumok kezelése

A dátumok annyira mindennapjaink részét képezik, hogy nem is gondolkodunk, amikor velük dolgozunk, naptárunk fortélyainak helyes kezelése azonban a programokat meglehetősen bonyolulttá teheti. Szerencsére a PHP hatékony eszközöket biztosít a dátumszámításhoz, amely megkönnyíti a feladatot.

Ebben a fejezetben a következőkről tanulunk:

- Hogyan kérdezhetjük le a pontos időt és dátumot?
- Hogyan szerezhetünk információt egy dátumról?
- Hogyan formázzunk meg egy dátumot?
- Hogyan ellenőrizzük a dátumok érvényességét?
- Hogyan állítsuk be a dátumot?
- Hogyan írjunk egyszerű naptárprogramot?

A dátum kiderítése a `time()` függvénnel

A PHP `time()` függvénye mindent elárul, amire szükségünk lehet a pontos idővel és dátummal kapcsolatban. A függvénynek nincs bemenete (paramétere) és egy egész számot ad vissza, amely egy kicsit nehezen értelmezhető, viszont nagyon informatív.

A `time()` által visszaadott egész szám a greenwichi középido szerinti 1970. január 1. éjféltől eltelt másodpercek számát adja meg. Ez a dátum a „UNIX kor kezdete” néven ismeretes, az azóta eltelt másodpercek számát pedig időbélyegnek (néha időbélyegző, time stamp) nevezik. Az időbélyeget a PHP eszközeivel „emberi fogyasztásra alkalmasabb” formára hozhatjuk, de felmerül a kérdés: még ha ilyen nagyszerű eszközök is állnak rendelkezésünkre, az időbélyeg nem feleslegesen túlbonyolított módja a dátum tárolásának? Nem, éppen ellenkezőleg: egyetlen számból rengeteg információt szerezhetünk. És ami még ennél is jobb, az időbélyeggel sokkalta könnyebben lehet számításokat végezni, mint azt gondolnánk.

Képzeljünk csak el egy házi dátumrendszert, amelyben az éveket, hónapokat és a hónapok napjait jegyezzük fel. Most képzeljünk el egy olyan programot, amellyel egy napot kell hozzáadnunk egy adott dátumhoz. Ha ez a dátum éppenséggel 1999. december 31., akkor ahelyett, hogy 1-et adnánk a dátumhoz, olyan programot kellene írni, amely a napot 1-re írta át, a hónapot januárra, az évet pedig 2000-re. Az időbélyegben viszont csak egy napnyi másodpercet kell hozzáadni az aktuális dátumhoz és már készen is vagyunk. Ezt az új számot pedig kedvünk szerint alakíthatjuk át valamilyen barátságosabb formára.

Az időbélyeg átalakítása a `getdate()` függvénnel

Most, hogy rendelkezésünkre áll az időbélyeg, át kell alakítanunk, mielőtt megmutatnánk a felhasználónak. A `getdate()` elhagyható bemenete egy időbélyeg, visszatérési értéke pedig egy asszociatív tömb. Ha nem adjuk meg az időbélyeget, a függvény az éppen aktuális időbélyeggel dolgozik, mintha azt a `time()` függvénytől kapta volna. A 15.1. táblázat a `getdate()` által visszaadott tömb elemeit részletezi.

15.1. táblázat A `getdate()` függvény által visszaadott asszociatív tömb

<i>Kulcs</i>	<i>Leírás</i>	<i>Példa</i>
<code>seconds</code>	A percből eltelt másodpercek száma (0-59)	28
<code>minutes</code>	Az órából eltelt percek száma (0-59)	7
<code>hours</code>	A nap órája (0-23)	12
<code>mday</code>	A hónap napja (1-31)	20

15.1. táblázat (folytatás)

wday	A hét napja (0-6)	4
mon	Az év hónapja (1-12)	1
year	Év (négy számjeggyel)	2000
yday	Az év napja (0-365)	19
weekday	A hét napja (névvel)	Thursday
month	Az év hónapja (névvel)	January
0	Az időbélyeg	948370048

A 15.1. programban a `getdate()` függvénnyel szétbontjuk az időbélyeget, majd a `foreach` utasítással kiírjuk az egyes elemeket. A 15.1. ábrán a `getdate()` függvény egy lehetséges kimenetét láthatjuk. A `getdate()` a helyi időzóna szerinti dátumot adja vissza.

15.1. program Dátum lekérdezése a `getdate()` függvénnyel

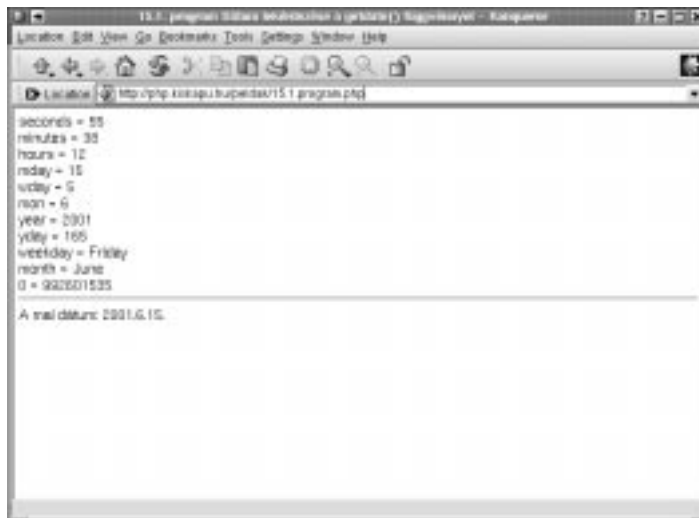
```

1: <html>
2: <head>
3: <title>15.1. program Dátum lekérdezése a getdate()
   függvénnyel</title>
4: </head>
5: <body>
6: <?php
7: $datum_tomb = getdate(); // nem adtunk meg bemenetet,
   így a mai dátumot fogja visszaadni
8: foreach ( $datum_tomb as $kulcs => $ertek )
9:     {
10:        print "$kulcs = $ertek<br>";
11:    }
12: ?>
13: <hr>
14: <?
15: print "A mai dátum: " $datum_tomb["year"] "
   " $datum_tomb["mon"] "
   " $datum_tomb["mday"] <p>";
16: ?>
17: </body>
18: </html>

```

15.1. ábra

A getDate() függvény használata



Az időbélyeg átalakítása a date() függvénnyel

A getDate() függvényt akkor használjuk, amikor a kimenetét képező elemeket szeretnénk használni, a dátumot azonban általában szöveggként szeretnénk megjeleníteni. A date() függvény a dátumot formázott karakterlánc formájában adja vissza. A date() függvény által visszaadott érték formátuma a paraméterként átadható formázó karakterlánccal nagymértékben befolyásolható. A formázó karakterláncon kívül a date() függvénynek átadhatjuk az időbélyeget is, bár ez nem kötelező. A 15.2. táblázatban láthatjuk azokat a kódokat, amelyeket a formázó karakterlánc tartalmazhat. A date() függvénynek átadott formázóban található minden egyéb szöveg a visszaadott értékben változatlanul meg fog jelenni.

15.2. táblázat A date() által használt formázási kódok.

Formátum	Leírás	Példa
a	'am' (délelőtt) vagy 'pm' (délután), kisbetűvel	pm
A	'AM' (délelőtt) vagy 'PM' (délután), nagybetűvel	PM
d	A hónap napja (bevezető nullákkal írt szám)	05
D	A hét napja (három betűvel)	Thu
F	A hónap neve	January
h	Óra (12 órás formátum, bevezető nullákkal)	03
H	Óra (24 órás formátum, bevezető nullákkal)	05
g	Óra (12 órás formátum, bevezető nullák nélkül)	3
G	Óra (24 órás formátum, bevezető nullák nélkül)	5

15.2. táblázat (folytatás)

<i>Formátum</i>	<i>Leírás</i>	<i>Példa</i>
i	Perc	47
j	A hónap napja (bevezető nullák nélkül)	5
l	A hét napja (névvel)	Thursday
L	Szökőév ('1' ha igen, '0' ha nem)	1
m	Az év hónapja (számmal, bevezető nullákkal)	01
M	Az év hónapja (három betűvel)	Jan
n	Az év hónapja (számmal, bevezető nullák nélkül) 1	
s	Az óra percei	24
U	Időbélyeg	948372444
y	Év (két számjegy)	00
Y	Év (négy számjegy)	2000
z	Az év napja (0-365)	19
Z	A greenwichi középидőtől való eltérés másodpercben	0

A 15.2. program néhány formátumkódot mutat be.

15.2. program Dátum formázása a date() függvénnyel

```

1: <html>
2: <head>
3: <title>15.2. program Dátum formázása a date()
   függvénnyel</title>
4: </head>
5: <body>
6: <?php
7: print date("Y.m.d. H:i:s<br>", time());
8: // 2000.12.10. 14:42:55
9:
10: $de_du = Array ("am" => "délelőtt", "pm"
   => "délután");
11: $honapok = Array("január", "február", "március",
   "április",
12: "május", "június", "július", "augusztus",
   "szeptember",
13: "október", "november", "december");

```

15.2. program (folytatás)

```

14:
15: $napszak = $de_du[date("a")];
16: $honap = $honapok[date("n")-1];
17:
18: print " Ma " . date("Y ") . $honap .
    date(" j. \n\ap\j\ a v\ a\n, " );
19: print $napszak . date(" g ór\ a i perc");
20: // Ma 2000 december tizedike van,
    délután 2 óra 42 perc.
21: ?>
22: </body>
23: </html>

```

A magyar dátumok formázása kétségtelenül bonyolultabb, mint a belsőleg támogatott angol dátumformátumok, ezért létre kell hoznunk egy tömböt, amely a „délelőtt” és „délután” szavakat tartalmazza, valamint egy másik tömböt a hónapok nevei számára. A \$napszak egyszerűen a `date()` függvény "a" formátumra adott kimenetéből származtatható, a \$honap-hoz pedig a tömb `date("n")-1`-edik elemét kell vennünk, mivel a tömbök nullától számozódnak.

Bár a formázás bonyolultnak tűnik, nagyon könnyű elvégezni. Ha olyan szöveget szeretnénk a formázáshoz adni, amely formátumkód-betűket tartalmaz, a fordított perjel (\) karakterrel jelezhetjük, hogy az utána jövő karakter nem formátumkód. Azon karakterek esetében, amelyek a \ után válnak vezérlőkarakterré, ezt a \ elé tett újabb \ segítségével kell jeleznünk. A "\n"-t például ennek megfelelően "\\n"-nek kell írunk, ha egy "n"-t szeretnénk a formázási karaktersorozatba írni. A fenti példában a formátumkarakterek ütközését elkerülendő vettük külön a \$honap és \$napszak kiírását, hogy a `date()` függvény ne tekintse ezek karaktereit értelmezendő formátum-meghatározásoknak. A `date()` függvény a helyi időzóna szerint adja vissza a dátumot. Ha azt szeretnénk, hogy greenwichi középido (Greenwich Mean Time, GMT) szerint kapjuk meg, használjuk a `gmdate()` függvényt, amely egyébként minden más tekintetben ugyanígy működik.

Időbélyeg készítése az `mktime()` függvénnyel

Már képesek vagyunk az aktuális dátumot használni, de tetszőleges dátumokat még nem tudunk kezelni. Az `mktime()` függvény által visszaadott időbélyeget a `date()` vagy a `getdate()` függvényekkel használhatjuk. Az `mktime()` függvény bemenete hat egész szám, a következő sorrendben:

óra
perc
másodperc
hónap
hónap napja
év

A 15.3. program az `mktime()` függvénnyel állít elő időbélyeget, amit aztán a `date()` függvénnyel használ.

15.3. program Időbélyeg készítése az `mktime()` függvénnyel

```
1: <html>
2: <head>
3: <title>15.3. program Időbélyeg készítése az mktime()
   függvénnyel </title>
4: </head>
5: <body>
6: <?php
7: // időbélyeget készít 1999. 05. 01. 12 óra 30 perchez
8: $ido = mktime( 12, 30, 0, 5, 1, 1999 );
9: print date("Y.m.d. H:i:s<br>", $ido);
10: // 1999.05.01. 12:30:00
11: ?>
12: </body>
13: </html>
```

Az `mktime()` függvény néhány vagy akár az összes eleme elhagyható, ilyenkor az aktuális időnek megfelelő értékeket használja. Az `mktime()` emellett azokat az értékeket is kiigazítja, amelyek túl vannak a megfelelő értéktartományon, így ha 25-öt adunk meg óra paraméterként, akkor a meghatározott nap, hónap, év paraméterek utáni nap reggeli 1.00-ként fogja kezelni azt. Az `mktime()` függvény kimenete általában adatbázisban vagy fájlban való tároláshoz, illetve dátumszámításokhoz lehet hasznos.

A dátum ellenőrzése a `checkdate()` függvénnyel

Előfordulhat, hogy a felhasználó által bevitt dátummal kell dolgoznunk. Mielőtt nekiikedenénk a munkának vagy tárolnánk a dátumot, meg kell bizonyosodnunk róla, hogy az érvényes-e. A `checkdate()` függvény három egész számot, a hónapot, a napot, és az évet kéri paraméterként (ebben a sorrendben) és `true` (igaz) értéket ad vissza, ha a hónap 1 és 12 közt van, a nap elfogadható az adott

hónapban és évben (számításba véve a szökőéveket) és az év 0 és 32 767 közé esik. Vigyázzunk azonban, mert előfordulhat, hogy egy dátum ugyan érvényes, de más dátumfüggvények számára mégsem elfogadható. Például a következő sor `true` (igaz) értéket ad vissza:

```
checkdate( 4, 4, 1066 )
```

Viszont ha ezekkel az értékekkel egy dátumot szeretnénk létrehozni, az `mktime()` által visszaadott időbélyeg `-1` lenne. Általános szabályként fogadjuk el, hogy az 1902 előtti évekre nem használjuk az `mktime()` függvényt és az 1970 előttiekre is csak elővigyázatosan.

Egy példa

Próbáljunk meg ezen függvények közül egyetlen példán belül minél többet használni. Készítsünk naptárat, amely egy 1980 és 2010 közötti tetszőleges hónap napjait mutatja. A felhasználó lenyíló menüvel választhatja ki a hónapot és az évet. A program kimenetében az adott hónap dátumai a hét napjainak megfelelően kerülnek elrendezésre. Két globális változót használunk, a `$honap` és az `$ev` változókat, melyek adatait a felhasználó adja meg. A két változó segítségével elkészítjük a meghatározott hónap első napjának időbélyegét. Ha nem adtuk meg az évet vagy a hónapot, vagy megadtuk, de a bevitt adat nem érvényes, a program bemenetének alapbeállításként a folyó év aktuális hónapjának első napját tekintjük.

A felhasználó által bevitt adatok ellenőrzése

Amikor egy felhasználó először látogat el weboldalunkra, valószínűleg nem fogja megadni a bekért adatokat. Erre az `isset()` függvény segítségével készíthetjük fel programunkat. Ez a függvény a `false` (hamis) értéket adja vissza, ha a neki átadott változó nem meghatározott. (Használhatjuk helyette a `checkdate()` függvényt is.) A 15.4. példa azt a kódrészletet tartalmazza, amely ellenőrzi a `$honap` és az `$ev` változókat és egy időbélyeget hoz létre belőlük.

15.4. program A felhasználó által bevitt adatok ellenőrzése a naptárprogram számára

```
1: <?php
2: if ( ! checkdate( $honap, 1, $ev ) )
3:     {
4:         $mostTomb = getdate();
5:         $honap = $mostTomb["mon"];
6:         $ev = $mostTomb["year"];
7:     }
```


15.4. program (folytatás)

```
8: $kezdet = mktime ( 0, 0, 0, $honap, 1, $ev );
9: $elsoNapTombje = getdate($kezdet);
10: if ($elsoNapTombje["wday"] == 0)
    { $elsoNapTombje["wday"] = 6; }
11: else { $elsoNapTombje["wday"]--; }
12: ?>
```

A 15.4. program egy nagyobb program részlete, így önmagában nincs kimenete. Az `if` utasításban a `checkdate()` függvényt használjuk a `$honap` és az `$ev` változók ellenőrzésére. Ha ezek nem meghatározottak, a `checkdate()` `false` (hamis) értéket ad vissza, mivel nem hozhatunk létre érvényes dátumot meghatározatlan hónap és év paraméterekből. E megközelítés további haszna, hogy egyúttal a felhasználó által bevitt adat érvényességét is ellenőrizzük.

Ha a dátum nem érvényes, a `getdate()` függvénnyel létrehozunk egy, az aktuális dátumon alapuló asszociatív tömböt. Ezután a tömb `mon` és `year` elemeivel magunk állítjuk be a `$honap` és az `$ev` változók értékeit.

Most hogy megbizonyosodtunk, hogy a program bemenetét képező két változó érvényes adatot tartalmaz, az `mktime()` függvényt használjuk a hónap első napja időbélyegének létrehozására. A későbbiek során még szükségünk lesz ennek az időbélyegnek a napjára és hónapjára, ezért létrehozunk egy `$elsoNapTombje` nevű változót, amely a `getdate()` függvény által visszaadott és ezen az időbélyegen alapuló asszociatív tömb lesz. Végül a magyar héthasználatnak megfelelően át kell alakítanunk a hét napjára vonatkozó értéket. A `getdate()` és más dátumfüggvények a hét első napjának a vasárnapot tekintik. Ezért ha a hét első napján vagyunk (ez kapja a nullás számot), akkor hetedikre írjuk át, egyébként pedig a nap számát eggyel csökkentjük. Így a vasárnap a hét végére kerül, a többi nap pedig eggyel előrébb, azaz a hétfő az első helyen szerepel.

A HTML űrlap létrehozása

Olyan kezelőfelületet kell létrehoznunk, amellyel a felhasználók lekérdezhetik egy hónap és év adatait. Ennek érdekében `SELECT` elemeket fogunk használni. Bár ezeket módosíthatatlan elemekként, a HTML kódban is megadhatnánk, azt is biztosítanunk kell, hogy a lenyíló menük alapbeállítása az adott hónap legyen, ezért a menüket dinamikusan hozzuk létre és csak ott adjuk hozzá a `SELECTED` tulajdonságot az `OPTION` elemhez, ahol szükséges. A 15.5. programban a létrehozott űrlapot láthatjuk.

15.5. program A naptárprogram HTML úrlapja

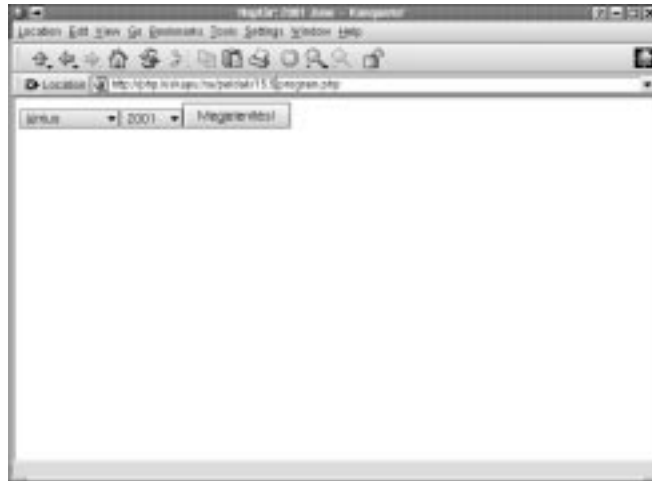
```
1: <?php
2: if ( ! checkdate( $honap, 1, $ev ) )
3:     {
4:         $mostTomb = getdate();
5:         $honap = $mostTomb["mon"];
6:         $ev = $mostTomb["year"];
7:     }
8: $kezdet = mktime ( 0, 0, 0, $honap, 1, $ev );
9: $elsoNapTombje = getdate($kezdet);
10: if ( $elsoNapTombje["wday"] == 0 )
11:     { $elsoNapTombje["wday"] = 6; }
12: else { $elsoNapTombje["wday"]--; }
13: ?>
14: <html>
15: <head>
16: <title><?php print "Naptár:" $elsoNapTombje["year"]
17:         $elsoNapTombje["month"] ?></title>
18: </head>
19: <body>
20: <form method="post">
21: <select name="honap">
22: <?php
23: $honapok = Array ( "január", "február", "március",
24:                   "április",
25:                   "május", "június", "július", "augusztus",
26:                   "szeptember",
27:                   "október", "november", "december");
28: for ( $x=1; $x <= count( $honapok ); $x++ )
29:     {
30:         print "\t<option value=\"$x\"";
31:         print ( $x == $honap ) ? " SELECTED":"";
32:         print ">". $honapok[ $x-1 ]. "\n";
33:     }
34: ?>
35: </select>
36: <select name="ev">
37: <?php
38: for ( $x=1980; $x<2010; $x++ )
39:     {
40:         print "\t<option";
41:         print ( $x == $ev ) ? " SELECTED":"";
```

15.5. program (folytatás)

```
39:      print ">$x\n";
40:      }
41:  ?>
42:  </select>
43:  <input type="submit" value="Megjelenítés!">
44:  </form>
45:  </body>
46:  </html>
```

Miután létrehoztuk a `$kezdet` időbélyeget és az `$elsoNapTombje` változót, megírjuk az oldal HTML kódját. Vegyük észre, hogy az `$elsoNapTombje` változót arra használjuk, hogy az évet és a hónapot a `TITLE` (cím) elemhez adhassuk. Az előző példákban a `FORM` elemen belüli `$PHP_SELF` globális változót használtuk arra, hogy biztosítsuk, az űrlap az elküldés után újra megjeleníti magát, e programban viszont azt a tényt használtuk ki, hogy ha egy `FORM` kódból kihagyjuk az `ACTION` paramétert, alapbeállítás szerint ugyanez történik. A lenyíló menük `SELECT` elemének létrehozásához visszalépünk `PHP` módba, hogy létrehozzuk az egyes `OPTION` címkéket. Először létrehozunk egy `$honapok` nevű tömböt, amely a 12 hónap nevét tartalmazza. Ezen aztán végiglépünk egy ciklussal, mindegyikből készítve egy `OPTION` címkét. Ez az egyszerű `SELECT` elem létrehozásának túlbonyolított módja lenne, ha nem kellene közben figyelni az `$x` változót (a `for` ciklusváltozóját). Ha az `$x` és a `$honap` megegyezik, a `SELECTED` kifejezést hozzáadjuk az `OPTION` címkéhez, biztosítván ezzel, hogy a megfelelő hónap automatikusan kijelölődik a lap betöltésekor. Hasonló módszerrel írjuk meg az év menüjét is, végül `HTML` módba visszatérve létrehozzuk a Megjelenítés gombot.

Így egy olyan űrlapot kapunk, amely saját magának elküldi a hónap és év paramétereket, és amelynek vagy az aktuális év és hónap, vagy pedig az előzőleg megadott év és hónap az alapbeállítása. A kész űrlapot a 15.2. ábrán láthatjuk.

15.2. ábra*A naptár űrlapja***A naptár táblázatának létrehozása**

Most egy táblázatot kell létrehoznunk, amit a kiválasztott hónap napjaival kell feltöltenünk. Ezt a 15.6. példában követhetjük nyomon, amely a teljes naptár-programot tartalmazza.

15.6. program A teljes naptárprogram

```

1: <?php
2: define("EGYNAP", (60*60*24) );
3: if ( ! checkdate( $honap, 1, $ev ) )
4:     {
5:         $mostTomb = getdate();
6:         $honap = $mostTomb["mon"];
7:         $ev = $mostTomb["year"];
8:     }
9: $kezdet = mktime ( 0, 0, 0, $honap, 1, $ev );
10: $elsoNapTombje = getdate($kezdet);
11: if ($elsoNapTombje["wday"] == 0)
12:     { $elsoNapTombje["wday"] = 6; }
13: else { $elsoNapTombje["wday"]--; }
14: ?>
15: <html>
16: <head>
17: <title><?php print "Naptár:" $elsoNapTombje["year"]
18:     $elsoNapTombje["month"] ?></title>
19: </head>
20: <body>

```

15.6. program (folytatás)

```
20: <form method="post">
21: <select name="honap">
22: <?php
23: $honapok = Array ( "január", "február", "március",
                    "április",
24: "május", "június", "július", "augusztus",
                    "szeptember",
25: "október", "november", "december");
26: for ( $x=1; $x <= count( $honapok ); $x++ )
27:     {
28:     print "\t<option value=\"$x\"";
29:     print ($x == $honap)? " SELECTED":"";
30:     print ">".$honapok[$x-1]."\n";
31:     }
32: ?>
33: </select>
34: <select name="ev">
35: <?php
36: for ( $x=1980; $x<2010; $x++ )
37:     {
38:     print "\t<option";
39:     print ($x == $ev)? " SELECTED":"";
40:     print ">$x\n";
41:     }
42: ?>
43: </select>
44: <input type="submit" value="Megjelenítés!">
45: </form>
46: <p>
47: <?php
48: $napok = Array ( "hétfő", "kedd", "szerda",
                  "csütörtök",
49: "péntek", "szombat", "vasárnap");
50: print "<TABLE BORDER=1 CELLPADDING=5>\n";
51: foreach ( $napok as $nap )
52:     print "\t<td><b>$nap</b></td>\n";
53: $kiirando = $kezdet;
54: for ( $szamlalo=0; $szamlalo < (6*7); $szamlalo++ )
55:     {
56:     $napTomb = getdate( $kiirando );
```

15.6. program (folytatás)

```

57:         if ( (($szamlalo) % 7) == 0 )
58:             {
59:                 if ( $napTomb["mon"] != $honap )
60:                     break;
61:                 print "</tr><tr>\n";
62:             }
63:         if ( $szamlalo < $elsoNapTombje["wday"] ||
              $napTomb["mon"] != $honap )
64:             {
65:                 print "\t<td><br></td>\n";
66:             }
67:         else
68:             {
69:                 print "\t<td>" . $honapok[$napTomb["mon"]-1]
                          " $napTomb["mday"] "</td>\n";
70:                 $kiirando += EGYNAP;
71:             }
72:         }
73: print "</tr></table>";
74: ?>
75: </body>
76: </html>
77:

```

Mivel a táblázat fejlécébe a hét napjai kell, hogy kerüljenek, a programot végigléptetjük a napok neveinek tömbjén és mindegyiket saját cellájába írjuk. A lényeg a program utolsó `for` ciklusában történik.

Megadjuk a `$szamlalo` változó kezdeti értékét, majd biztosítjuk, hogy a ciklus 42 ismétlés után leáll. Erre azért van szükség, hogy a dátumoknak elegendő cellát hozzunk létre. A ciklusban a `$kiirando` változót a `getdate()` függvénnyel egy dátumtömbbé alakítjuk és az eredményt a `$napTomb` változóba tesszük. Bár a ciklus indításakor a `$kiirando` változó értéke a hónap első napja lesz, az időbélyeget a ciklus minden lefutásakor 24 órával növeljük.

A maradékos osztás segítségével megnézzük, hogy a `$szamlalo` változó értéke osztható-e 7-tel. Az ehhez az `if` utasításhoz tartozó kódrész csak akkor fut le, ha a `$szamlalo` 0 vagy a 7 többszöröse. Így eldönthetjük, hogy befejezzük-e a ciklust vagy új sort kezdünk.

Ha látjuk, hogy még a ciklus első lefutásában vagy egy cellasor végénél vagyunk, elvégzünk még egy ellenőrzést: megvizsgáljuk, hogy a `$napTomb[mon]` (hónap-szám) eleme nem egyenlő-e a `$hónap` változóval. Ha nem egyenlő, a ciklust befejezhetjük. Emlékezzünk vissza, hogy a `$napTomb`-ben ugyanaz az időpont van, mint ami a `$kezdet`-ben, ami viszont a hónapnak pontosan az a része, amelyet megjelenítettünk. Ha a `$kezdet` túljut az aktuális hónapon, a `$napTomb[mon]` más számot fog tartalmazni, mint a felhasználó által megadott `$hónap` szám. A maradékosztás igazolta, hogy éppen egy sor végén állunk: ha új hónapban járunk, elhagyhatjuk a ciklust, ha viszont a sor végére értünk és még mindig ugyanabban a hónapban vagyunk, befejezzük a sort és újat kezdünk.

A következő `if` utasításban azt határozzuk meg, hogy írjunk-e szöveget a cellába. Nem minden hónap kezdődik hétfővel, így könnyen meglehet, hogy egy-két üres cellával indítunk. Emellett kevés hónap végződik sor végén, így az is valószínű, hogy mielőtt lezárnánk a táblázatot, egy pár üres cellát is kell írunk. A hónap első napjának adatait az `$elsőNapTombje` változóban találjuk. Ebből az `$elsőNapTombje["wday"]` kifejezés segítségével kideríthető, hogy az adott dátum hányadik nap a héten. Ha a `$szamlalo` kisebb, mint ez a szám, akkor tudjuk, hogy még nem értük el azt a cellát, amelybe már írhatnánk. Ugyanezt felhasználva ha a `$hónap` változó már nem egyenlő a `$napTomb["mon"]` változóval, tudhatjuk, hogy elértük a hónap végét (de a sor végét még nem). Mindkét esetben egy üres cellát írunk ki a böngészőbe.

A végső `else` utasításnál jön a feladat érdekes része. Azzal már tisztában vagyunk, hogy a kiírandó hónapban vagyunk és az aktuális nap oszlopa megegyezik az `$elsőNapTombje` változóban tárolt számmal. Most kerül felhasználásra a ciklus korábbi részében létrehozott `$napTomb` asszociatív tömb, valamint a hónapok magyar neveit tartalmazó `$hónapok` tömb, hogy kiírjuk a hónap nevét és napját egy cellába.

Végül növeljük a `$kiirando` változót, amely az időbélyeget tartalmazza. Egyszerűen hozzáadjuk egy nap másodperceinek számát (ezt az értéket a program elején határoztuk meg) és már újra is kezdhetjük a ciklust a `$kiirando` változó egy új értékével. A 15.3. ábrán a program egy lehetséges eredményét láthatjuk.

15.3. ábra

A naptárprogram

hétfő	kedd	szerda	csütörtök	péntek	szombat	vasárnap
				június 1	június 2	június 3
június 4	június 5	június 6	június 7	június 8	június 9	június 10
június 11	június 12	június 13	június 14	június 15	június 16	június 17
június 18	június 19	június 20	június 21	június 22	június 23	június 24
június 25	június 26	június 27	június 28	június 29	június 30	

Összefoglalás

Ezen az órán megtanultuk, hogyan használjuk a `time()` függvényt az aktuális időbélyeg megszerzésére. Megtanultuk, hogyan használjuk az időbélyegből a dátum részeinek kinyerésére a `getdate()` függvényt és az időbélyeget formázott szöveggé alakító `date()` függvényt. Arról is tanultunk, hogyan hozunk létre időbélyeget az `mktime()` függvénnyel. Megtanultuk ellenőrizni a dátum érvényességét a `checkdate()` függvénnyel, magyar elvárások szerint formáztunk dátumokat, végül tanulmányoztunk egy példaprogramot, amely az említett függvényeket alkalmazta.

Kérdések és válaszok

Vannak olyan függvények, amelyekkel a különböző naptárak között váltogatni lehet?

Igen, a PHP naptárak teljes sorozatát biztosítja. Ezekről a hivatalos PHP kézikönyvben olvashatunk, a <http://www.php.net/manual/ref.calendar.php> címen.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan kapunk meg egy UNIX időbélyeget, amely a pontos dátumot és időt jelöli?
2. Melyik az a függvény, amelynek bemenete egy időbélyeg és egy, az adott dátumot jelképező asszociatív tömböt ad vissza?
3. Mely függvényt használnánk a dátum formázására?
4. Hogyan kapjuk meg egy tetszőleges dátum időbélyegét?
5. Melyik függvényt használjuk egy dátum érvényességének ellenőrzésére?

Feladatok

1. Készítsünk egy születésnapig visszaszámláló programot. Űrlappal lehessen megadni az év, hó és nap értékeit, kimenete pedig egy üzenet legyen, amely közli a felhasználóval, hogy hány nap, óra, perc, másodperc van még hátra a nagy napig.



16. ÓRA

Az adatok kezelése

Ezen az órán az adatellenőrzésben és az adاتمűveletekben fogunk egy kicsit elmélyedni. Újra áttekintjük az adattípusokat. A PHP ugyan automatikusan kezeli ezeket, de elengedhetetlen, hogy értsük az adatkezelést, ha nagy, megbízható hálózati alkalmazásokat kell írunk. Visszatérünk a tömbökhöz is és végül megismerkedünk a PHP fejlettebb adatkezelési, adatosztályozási eljárásaival.

Az óra folyamán a következőkről tanulunk:

- Hogyan alakítsuk át az egyes adattípusokat más adattípusokká?
- Hogyan alakítja át automatikusan a PHP az adattípusokat a kifejezésekben?
- Milyen további módjai vannak az adattípusok ellenőrzésének?
- Miért fontos megértenünk az adattípusokat?
- Hogyan ellenőrizzük, hogy egy változónak van-e értéke?
- Hogyan lehet másként bejárni egy tömböt?
- Hogyan deríthetjük ki, hogy egy tömb tartalmaz-e egy bizonyos elemet?
- Hogyan alakítsuk át egy tömb minden elemét?
- Hogyan lehet saját szempontok alapján tömbrendezést végezni?

Újra az adattípusokról

A negyedik órában már tanultunk néhány dolgot a PHP adattípusairól. Van azonban még valami, ami eddig kimaradt. Ebben a részben a változók adattípusának ellenőrzésére használt függvények közül ismerkedünk meg néhányval, majd sorra vesszük azokat a feltételeket, amelyek mellett a PHP automatikusan elvégzi helyettünk az adattípus-átalakításokat.

Egy kis ismételés

Azt már tudjuk, hogy a PHP változóinak értéke egész szám (integer), lebegőpontos szám (double), karakterlánc (string), logikai érték (boolean), objektum vagy tömb lehet. A `gettype()` függvénnyel bármelyik változótípust lekérdezhetjük. A függvény bemenete egy tetszőleges típusú változó, kimenete pedig a változó adattípusát leíró szöveg:

```
$adat = 454;
print gettype( $adat );
// azt írja ki, hogy "integer", azaz egész szám
```

A változók adattípusát közvetlen típusátalakítással vagy a `settype()` függvénnyel módosíthatjuk. Az adattípusok átalakításához a zárójelbe tett adattípust a megváltoztatandó változó vagy érték elé írjuk. A folyamat közben a változó tartalmát a legcsekélyebb mértékben sem módosítjuk, ehelyett egy átalakított másolatot kapunk vissza. A következő kódrészlet egy tizedestörtöt alakít át egész számmá.

```
$adat = 4.333;
print ( integer ) $adat;
// "4"-et ír ki
```

Az `$adat` változó lebegőpontos szám típusú maradt, mi csupán a típusátalakítás eredményét írtuk ki.

A változók adattípusának módosításához a `settype()` függvényt használhatjuk, amelynek bemenete az átalakítandó változó és a céladattípus neve.

```
$adat = 4.333;
settype( $adat, "integer" );
print $adat;
// "4"-et ír ki
```

Az `$adat` változó most már egy egész számot tartalmaz.

Összetett adattípusok átalakítása

Néhány egyszerű (skaláris és szöveges) adattípus közötti átváltást már láttuk részleteiben is. Mi történik viszont akkor, ha egyszerű (egész és nem egész számok) és összetett adattípusok (objektumok és tömbök) között kell típust váltani?

Amikor egy egyszerű adattípust tömbbé alakítunk, olyan tömb jön létre, melynek első eleme az eredeti érték lesz:

```
$szoveg = "ez az én szövegem"
$tomb_szoveg = (array) $szoveg;
print $tomb_szoveg[0];
// kiírja az "ez az én szövegem" karakterláncot
```

Amikor a skaláris vagy szöveges változókat objektumokká alakítjuk, egy egyetlen scalar nevű tulajdonságot tartalmazó objektum jön létre, ez tartalmazza az eredeti értéket:

```
$szoveg = "ez az én szövegem"
$obj_szoveg = (object) $szoveg;
print $obj_szoveg->scalar;
// kiírja az "ez az én szövegem" karakterláncot
```

A dolog a tömbök és objektumok közötti váltáskor válik igazán izgalmassá. Amikor egy tömböt objektummá alakítunk, egy olyan új objektum jön létre, amelyben a tömb egyes kulcsainak egy-egy tulajdonság felel meg.

```
$cimek = array ( "utca" => "Főutca", "varos" => "Nagyfalva" );
$obj_cimek = ( object ) $cimek;
print $obj_cimek->utca;
// azt írja ki, hogy "Főutca"
```

Fordított esetben, egy objektum tömbbé alakításakor, olyan tömb jön létre, amelyben minden objektum tulajdonságnak megfelelő egy tömbelem. A metódusokat az átalakítás figyelmen kívül hagyja.

```
class Pont
{
    var $x;
    var $y;
    function Pont( $x, $y )
    {
        $this->x = $x;
        $this->y = $y;
    }
}
```

```
    }  
}  
$pont = new Pont( 5, 7 );  
$tomb_pont = (array) $pont;  
print $tomb_pont["x"];  
// azt írja ki, hogy "5"
```

Az adattípusok automatikus átalakítása

Ha olyan kifejezést hozunk létre, amelyben két különböző adattípus szerepel operandusként, a PHP automatikusan átalakítja az egyiket a számíthatóság kedvéért. Valószínűleg ezt már igénybe vettük, anélkül, hogy tudtunk volna róla. Az űrlapokból származó változók mindig karakterláncok, de ezeket használhattuk igaz-hamis értékű kifejezésekben is vagy számolásoknál, ha éppen arra volt szükségünk.

Tegyük fel, hogy egy felhasználót megkérdezünk, hány órát tölt a hálózaton hetente, és a választ az `$orak` nevű változóban tároljuk. Ez kezdetben szöveggént kerül tárolásra.

```
$orak = "15"  
if ($orak > 15)  
    print "Ilyen gyakori felhasználónak akár árendedmény  
        is járhatna";
```

Az `$orak` ellenőrzésekor a "15" karakterlánc egész számmá alakul, így a kifejezés eredménye `true` (igaz) lesz.

Az automatikus átalakítás szabályai viszonylag egyszerűek. Egész számokból vagy lebegőpontosakból álló környezetben a karakterláncok tartalmuknak megfelelően kerülnek átalakításra. Ha egy karakterlánc egész számmal kezdődik, a szám az annak megfelelő érték. Így a következő kifejezés eredménye 80:

```
4 * "20mb"
```

Ha a karakterlánc nem számmal kezdődik, 0-vá alakul. Így a következő sor 0-át ad eredményül:

```
4 * "körülbelül 20mb"
```

Ha a karakterláncban a számot egy pont követi, akkor az lebegőpontos értékké alakul. Ennek megfelelően a következő sor eredménye 4,8 lesz:

```
4 * "1.2"
```

Az ++ és -- műveletek karakterláncokra való alkalmazása különleges eset. A karakterlánc növelése, mint ahogy azt elvártuk, 1-et ad a karakterlánc átalakított értékéhez, de csak abban az esetben, ha a karakterlánc kizárólag számokból áll. Maga az új érték azonban karakterlánc marad:

```
$szoveg = "4";  
$szoveg++;  
print $szoveg; // kiírja az 5-öt  
print gettype( $szoveg ); // azt írja ki, hogy "string"
```

Ha egy betűket tartalmazó karakterláncot próbálunk növelni, egyedül az utolsó karakter kódja fog megnőni:

```
$szoveg = "alszol";  
$szoveg++;  
print $szoveg; // azt írja ki, hogy "alszom"
```

Vessük össze ezt a karakterlánc növelésének egy másik megközelítésével:

```
$szoveg = "alszol";  
$szoveg += 1;  
print $szoveg; // 1-et ír ki  
print gettype( $szoveg ); // azt írja ki, hogy "integer",  
                           azaz egész szám
```

Az előző példában a \$szoveg változó először 0-vá (egész számmá) alakul, aztán adjuk hozzá az 1-et. Ennek a műveletnek az eredménye 1, amelyet újra a \$szoveg változóba helyezünk. A változó most már egy egész számot fog tartalmazni.

Az egész számok és a lebegőpontosak közötti automatikus átváltás még ennél is egyszerűbb. Ha egy kifejezés bármelyik operandusa lebegőpontos, a PHP a másik operandust is lebegőpontosá alakítja és az eredmény is lebegőpontos lesz:

```
$eredmeny = ( 1 + 20.0 );  
print gettype( $eredmeny );  
// azt írja ki, hogy "double", azaz lebegőpontos
```

Fontos megjegyeznünk, hogy ha az automatikus átalakítás egy kifejezés kiértékeléséhez szükséges, a kifejezés egyik operandusa sem változik meg. Ha egy operandust át kell alakítani, akkor annak átalakított másolata fog szerepelni a kifejezés részeként.

Az adattípusok ellenőrzése

Már le tudunk kérdezni adattípusokat a `gettype()` függvénnyel, ami a hibakeresésnél jól jöhet, hiszen minden változóról meg tudjuk mondani, hogy milyen típusú. Gyakran viszont csak azt szeretnénk megnézni, hogy a változó egy bizonyos adattípusú értéket tartalmaz-e. A PHP ehhez az egyes adattípusoknak megfelelő függvényeket biztosít, amelyek bemenete egy változó vagy egy érték, visszatérési értéke pedig logikai. A függvények listáját a 16.1. táblázat tartalmazza.

16.1. táblázat Az adattípusok ellenőrzésének függvényei

Függvény	Leírás
<code>is_array()</code>	Igaz (<code>true</code>) értéket ad vissza, ha a paraméter tömb
<code>is_bool()</code>	Igaz (<code>true</code>) értéket ad vissza, ha a paraméter logikai érték
<code>is_double()</code>	Igaz (<code>true</code>) értéket ad vissza, ha a paraméter lebegőpontos szám
<code>is_int()</code>	Igaz (<code>true</code>) értéket ad vissza, ha a paraméter egész szám
<code>is_object()</code>	Igaz (<code>true</code>) értéket ad vissza, ha a paraméter objektum
<code>is_string()</code>	Igaz (<code>true</code>) értéket ad vissza, ha a paraméter karakterlánc

Ezek a függvények egy kicsit megkönnyítik az adattípusok ellenőrzését. Az

```
if ( gettype( $valtozo ) == "array" )
    print "ez egy tömb";
```

megegyezik a

```
if ( is_array( $valtozo ) )
    print "ez egy tömb";
```

kóddal. (A `$valtozo` ellenőrzésének második módja egy kicsit tömörebb.)

Az adattípus-váltás további módjai

Eddig az adattípus-váltás két módjával ismerkedtünk meg: vagy egy értéket alakítunk át meghatározott adattípussá, vagy a `settype()` függvényt használjuk. Ezek mellett a PHP bizonyos függvényekkel lehetővé teszi, hogy értékeket egész számmá, lebegőpontos számmá vagy szöveggé alakítsunk át. E függvények bemenete tömbön és objektumon kívül bármilyen adattípus lehet, kimenetük pedig az átalakított érték. Leírásukat a 16.2. táblázatban találhatjuk meg.

16.2. táblázat Az adattípus-váltás függvényei

<i>Függvény</i>	<i>Leírás</i>
<code>doubleval()</code>	Bemenete egy érték, kimenete egy azonos értékű lebegőpontos szám
<code>intval()</code>	Bemenete egy érték, kimenete egy azonos értékű egész szám
<code>strval()</code>	Bemenete egy érték, kimenete egy azonos értékű karakterlánc

Miért olyan fontosak az adattípusok?

A PHP nem követeli meg tőlünk, hogy egy változó létrehozásakor megadjuk annak adattípusát, de elvégzi az adattípus átalakításokat helyettünk, ha a kifejezésekben különböző adattípusú változókat használunk. Ha a PHP ennyire leegyszerűsíti az életünket, akkor miért van szükségünk mégis az adattípusok nyomon követésére? Azért, hogy megelőzhessük a hibákat. Képzeljünk el egy olyan függvényt, amely egy tömb kulcsait és értékeit írja ki egy böngészőbe. A PHP-ben a függvényparaméterek típusát sem adhatjuk meg, így nem írhatjuk elő, hogy a meghívó kód egy tömböt adjon át nekünk, amikor a függvényt meghatározzuk.

```
function tombKiir( $tomb )
{
    foreach ( $tomb as $kulcs => $ertek )
        print "$kulcs: $ertek<P>";
}
```

A következő függvény jól működik, ha tömb paraméterrel hívják meg.

```
tombKiir ( array(4, 4, 55) );
```

Ha figyelmetlenségünkben skaláris paramétert adunk át neki, az hibához vezet, amint azt a következő példa is mutatja:

```
tombKiir ( 4 );
// Warning: Non array argument supplied for foreach() in
// /home/httpd/htdocs/proba2.php on line 5
// azaz "Figyelmeztetés: a foreach() függvénynek nem tömb
// ➡ paramétert adtunk át
// a /home/httpd/htdocs/proba2.php program 5.
// ➡ sorában"
```

Azzal, hogy ellenőrizzük a kapott paraméter adattípusát, sokkal alkalmazkodóbbá tesszük a függvényt. Azt is megtehetjük, hogy a függvény szép csöndben visszatér, ha skaláris értéket kap:

```
function tombKiir( $tomb )
{
    if ( ! is_array( $tomb ) )
        return false;
    foreach ( $tomb as $kulcs => $ertek )
        print "$kulcs: $ertek<P>";
    return true;
}
```

Most már a hívó kód ellenőrizheti a függvény visszaadott értékeit, hogy megállapíthassa, végre tudta-e hajtani feladatát.

Adatátalakítást is használhatunk, hogy a skaláris adatot tömbként használhassuk fel:

```
function tombKiir( $tomb )
{
    if ( ! is_array( $tomb ) )
        $tomb = (array) $tomb;
    foreach ( $tomb as $kulcs => $ertek )
        print "$kulcs: $ertek<P>";
    return true;
}
```

A `tombKiir()` függvény nagyon rugalmassá vált, most már tetszőleges adattípust képes feldolgozni, akár objektumot is.

Az adattípus ellenőrzése akkor is jól jöhet, ha a függvények visszaadott értékeit szeretnénk ellenőrizni. Néhány nyelv, például a Java, minden módszere mindig egy előre meghatározott adattípust ad vissza. A PHP-nak nincsenek ilyen megkötései, viszont az ilyen rugalmasság alkalmanként kétértelműségekhez vezethet.

Erre láttunk egy példát a tizedik órában. A `readdir()` függvény hamis (`false`) értéket ad vissza, amikor eléri a beolvasott könyvtár végét, minden más esetben viszont a könyvtár egyik elemének nevét tartalmazó karakterláncot. Hagyományosan a következőhöz hasonló szerkezetet használnánk, ha egy könyvtár elemeit szeretnénk beolvasatni:

```
$kvt = opendir( "konyvtar" );
while ( $nev = readdir( $kvt ) )
    print "$nev<br>";
closedir( $kvt );
```

Ha azonban a `readdir()` által visszaadott egyik alkönyvtár neve "0", a `while` utasítás kifejezése erre a karakterláncra `false` (hamis) értéket ad vissza és befejezi a felsorolást. Ha ellenőrizzük a `readdir()` visszaadott értékének adattípusát, elkerülhetjük ezt a problémát:

```
$kvt = opendir( "konyvtar" );
while ( is_string( $nev = readdir( $kvt ) ) )
    print "$nev<br>";
closedir( $kvt );
```

A változók meglétének és ürességének ellenőrzése

Az adattípusok ellenőrzése hasznos lehet, ám előzőleg meg kell bizonyosodnunk róla, hogy a változó egyáltalán létezik-e, és meg kell néznünk, milyen értéket tartalmaz. Ezt az `isset()` függvénnyel tehetjük meg, amelynek bemenete egy változó, kimenete pedig `true` (igaz), ha a változó tartalmaz valamiféle értéket:

```
$nincsertek;
if ( isset( $nincsertek ) )
    print "a \ $nincsertek változónak van értéke";
else
    print "a \ $nincsertek változónak nincs értéke";
// kiírja, hogy "a $nincsertek változónak nincs értéke"
```

Azt a változót, amelyet már bevezettünk, de amelynek még nem adtunk értéket, a függvény nem beállítottként, érték nélkülként fogja kezelni.

Egy veszélyre azonban hadd hívjuk fel a figyelmet: ha 0-at vagy üres karakterláncot rendelünk egy változóhoz, a függvény azt már beállítottként, értékkel rendelkezőnek fogja tekinteni:

```
$nincsertek = "";
if ( isset( $nincsertek ) )
    print "a \ $nincsertek változónak van értéke";
else
    print " a \ $nincsertek változónak nincs értéke ";
// kiírja, hogy "a $nincsertek változónak van értéke"
```

Azok a változók, amelyeket a bejövő űrlapok töltenek fel értékkel, mindig beállítottként fognak szerepelni, még akkor is, ha a felhasználó egyetlen mezőt sem töltött ki adattal. Hogy az ilyen helyzetekkel megbirkózhassunk, először ellenőriznünk kell, hogy üres-e a változó. Az `empty()` függvény bemenete egy változó,

kimenete pedig true (igaz), ha a változó nincs beállítva vagy olyan adatot tartalmaz, mint a 0 vagy egy üres karakterlánc. Akkor is igaz értéket ad vissza, ha a változó üres tömböt tartalmaz:

```
$nincsertek = "";
if ( empty( $nincsertek ) )
    print "a \$nincsertek változó üres";
else
    print " a \$nincsertek változó adatot tartalmaz";
// kiírja, hogy "a \$nincsertek változó üres"
```

További tudnivalók a tömbökről

A hetedik órában már bemutattuk a tömböket és a tömbök kezeléséhez szükséges függvényeket. Ebben a részben további függvényekkel és ötletekkel ismerkedhetünk meg.

Tömbök bejárása más megközelítésben

A PHP 4 új eszköze a foreach utasítás, amellyel tömbök elemeit olvashatjuk be. A könyvben lévő példák legnagyobb részében ezt használjuk. A PHP 3 esetében még egész másképpen kellett bejárni a tömböket. Ha a PHP 3-nak is megfelelő programokat szeretnénk írni vagy szeretnénk érteni a PHP 4 előtti forráskódokat is, ezzel tisztában kell lennünk.

Tömb létrehozásakor a PHP egy belső mutatót alkalmaz, amely a tömb első elemére mutat. Ennek az elemnek a kulcsához és értékéhez az each() függvénnyel férhetünk hozzá. Az each() függvény bemenete egy tömbváltozó, kimenete pedig egy négyelemű tömb. Ezek közül az elemek közül kettő számmal indexelt, kettő pedig a "key" (kulcs) és az "value" (érték) címkét viseli. A függvény meghívása után a belső mutató a vizsgált tömb következő elemére fog mutatni, kivéve, ha elérte a tömb végét, ilyenkor false (hamis) értéket ad vissza. Hozzunk létre egy tömböt és próbáljuk ki az each() függvényt:

```
$reszletek = array( "iskola" => "Képzőművészeti", "tantargy"
    => "Térbeli ábrázolás" );
$elem = each( $reszletek );
print "$elem[0]<br>"; // azt írja ki, hogy "iskola"
print "$elem[1]<p>"; // azt írja ki, hogy "Képzőművészeti"
print "$elem["key"]<br>"; // azt írja ki, hogy "iskola"
print "$elem["value"]<br>"; // azt írja ki,
                                hogy "Képzőművészeti"
```

A `$reszletek` nevű tömböt két elemmel hozzuk létre, ezután átadjuk az `each()` függvénynek és a visszaadott értéket az `$elem` nevű tömbbe helyezzük. Az `$elem` tartalmazza a `$reszletek` tömbváltozó első elemének kulcsát és értékét.

Az `each()` által visszaadott tömböt kicsit nehézkes skaláris változókhoz rendelni, de szerencsére a PHP `list()` függvénye megoldja ezt a problémát. A `list()` függvény tetszőleges számú változót elfogad bemenetként és ezek mindegyikét a jobb oldalán megadott tömbváltozó megfelelő értékeivel tölti fel:

```
$tomb = array( 44, 55, 66, 77 );
list( $első, $második ) = $tomb;
print "$első"; // azt írja ki, hogy "44"
print "<BR>";
print "$második"; // azt írja ki, hogy "55"
```

Vegyük észre, hogy a `list()` függvénnyel könnyedén másolhatjuk át az előző példa tömbjének elemeit a külön változókba.

Használjuk arra a `list()` függvényt, hogy az `each()` minden egyes meghívásakor két változónak adjon értéket.

```
$reszletek = array( "iskola" => "Képzőművészeti",
                  "tantargy" => "Térbeli ábrázolás" );
while ( list( $kulcs, $ertek ) = each( $reszletek ) )
    print "$kulcs: $ertek<BR>";
```

Bár a kód működni fog, valójában még egy sor hiányzik. Ha tömbünkre már használtuk a belső mutatót módosító függvények egyikét, az már nem a tömb elejére mutat. A `reset()` függvénnyel visszaállíthatjuk a mutatót a tömb kezdetére. Ez a függvény paraméterként egy tömbváltozót vár.

Így az alábbi ismerősebb szerkezet

```
foreach( $reszletek as $kulcs => $ertek ) ;
    print "$kulcs: $ertek<BR>";
```

egyenértékű a következővel:

```
reset( $reszletek );
while ( list( $kulcs, $ertek ) = each( $reszletek ) )
    print "$kulcs: $ertek<BR>";
```

Elem keresése tömbben

A PHP 4-et megelőzőleg ha azt szerettük volna megtudni, hogy egy elem előfordul-e egy tömbben, addig kellett bejárnunk a tömböt, míg megtaláltuk az elemet vagy elértük a tömb végét. A PHP 4-ben azonban már rendelkezésünkre áll az `in_array()` függvény. Két paramétere van, az egyik a keresett érték, a másik az a tömb, amelyben keresni kívánunk. A függvény `true` (igaz) értéket ad vissza, ha megtalálja a keresett értéket, egyébként `false` (hamis) értéket kapunk.

```
$reszletek = array( "iskola" => "Képzőművészeti", "tantargy"
    => "Térbeli ábrázolás" );
if ( in_array( "Térbeli ábrázolás", $reszletek ) )
    print "A kurzus további intézkedésig
        felfüggesztve<P>\n";
```

Elemek eltávolítása a tömbből

Az `unset()` függvénnyel elemeket is eltávolíthatunk egy tömbből. A függvény bemenetéül egy változót vagy egy tömbelemet vár, majd azt minden teketória nélkül megsemmisíti. Ha a paraméter egy tömb egy eleme, a tömböt automatikusan lerövidíti.

```
unset( $proba["cim"] );
unset( $szamok[1] );
```

Az `unset()` függvény egyetlen csapdája az lehet, hogy a tömb indexei nem követik a tömb megváltozott méretét. Az előző példa tömbje a `$szamok[1]` elem eltávolítása után a következőképpen fest:

```
$szamok[0]
$szamok[2]
$szamok[3]
```

Szerencsére a `foreach()` függvénnyel ezen is gond nélkül végiglépkedhetünk.

Függvények alkalmazása a tömb összes elemére

A kifejezésekben szereplő skaláris változókat könnyen módosíthatjuk, egy tömb összes elemét megváltoztatni már egy kicsit nehezebb. Ha például a tömb összes elemének értékéhez egy számot akarunk adni, azt úgy tehetnénk meg, hogy a tömb összes elemén végiglépkedve frissítjük az értékeket. A PHP azonban ennél elegánsabb megoldást kínál.

Az `array_walk()` függvény egy tömb minden elemének kulcsát és értékét átadja egy, a felhasználó által meghatározott függvénynek. A függvény bemenete egy tömbváltozó, egy, a függvény nevét megadó karakterlánc érték, és egy elhagyható harmadik paraméter, amelyet a választott függvénynek szeretnénk még átadni.

Vegyünk egy példát. Van egy adatbázisból kinyert ártömbünk, de mielőtt munkához kezdenénk vele, az összes árhoz hozzá kell adnunk a forgalmi adót. Először azt a függvényt hozzuk létre, amely hozzáadja az adót:

```
function ado_hozzaado( &$sertek, $kulcs, $adoszazalek )
{
    $sertek += ( ($adoszazalek/100) * $sertek );
}
```

Az `array_walk()` számára készített összes függvénynek egyértéket, egy kulcsot és egy elhagyható harmadik paramétert kell várnia.

Ha paraméterként nem értéket szeretnénk átadni, hanem egy változót, melyben tükröződhetnek a függvény okozta változtatások a függvény hatókörén kívül is, a függvény-meghatározásban egy ÉS jelet (&) kell az adott paraméter elé írni. Ez fogja biztosítani, hogy ha a függvényen belül módosítjuk az értéket, az megjelenik a függvényen kívül a tömbben is. Ez az oka annak is, hogy példánkban az `ado_hozzaado()` függvénynek nincs szokásos értelemben vett visszatérési értéke.

Most hogy megvan a függvényünk, máris meghívhatjuk az `array_walk()` függvényt a megfelelő paraméterekkel:

```
function ado_hozzaado( &$sertek, $kulcs, $adoszazalek )
{
    $sertek += ( ($adoszazalek/100) * $sertek );
}

$arak = array( 10, 17.25, 14.30 );
array_walk( $arak, "ado_hozzaado", 10 );
foreach( $arak as $sertek )
    print "$sertek<BR>";
// kimenete:
// 11
// 18.975
// 15.73
```

A `$arak` tömbváltozót az `ado_hozzaado()` függvény nevével együtt átadjuk az `array_walk()` függvénynek. Az `ado_hozzaado()` függvénynek tudnia kell az érvényes adókulcsot. Az `array_walk()` harmadik, elhagyható paramétere átadódik a megnevezett függvénynek és ezzel elérjük, hogy az értesüljön az adókulcsról.

Tömbök egyéni rendezése

A kulcs vagy érték alapján történő rendezéssel már megismerkedtünk, nem mindig szoktunk azonban ilyen egyszerűen rendezni tömböket. Előfordulhat, hogy többdimenziós tömbbe beágyazott értékek alapján vagy a szokványos alfanumerikus összehasonlítástól eltérő szempont szerint szeretnénk rendezni.

A PHP lehetővé teszi, hogy magunk határozzuk meg az összehasonlító tömbrendező függvényeket. Számmal indexelt tömbök esetében ilyenkor az `usort()` függvényt kell meghívunk, melynek bemenete a rendeznivaló tömb és annak a függvénynek a neve, amely egy elempár összehasonlítását képes elvégezni.

Az általunk meghatározott függvénynek két paramétert kell elfogadnia, amelyek az összehasonlítandó tömbértékeket tartalmazzák. Ha a feltételek alapján ezek azonosak, a függvény a 0 értéket kell, hogy visszaadja, ha a tárgy tömbben az első paraméternek a második előtt kell jönnie, akkor -1-et, ha pedig ez első paraméternek kell a második után szerepelnie, akkor 1-et.

A 16.1. programban azt láthatjuk, hogyan használjuk az `usort()` függvényt egy többdimenziós tömb rendezésére.

16.1. program Többdimenziós tömb rendezése mező alapján az `usort()` függvénnyel

```
1: <?php
2: $stermek = array(
3:     array( "nev"=>"HAL 2000",    "ar"=>4500.5  ),
4:     array( "nev"=>"Modem",       "ar"=>55.5    ),
5:     array( "nev"=>"Nyomtató",    "ar"=>2200.5  ),
6:     array( "nev"=>"Csavarhúzó",  "ar"=>22.5    )
7: );
8: function arHasonlito( $a, $b )
9: {
10:     if ( $a["ar"] == $b["ar"] )
11:         return 0;
12:     if ( $a["ar"] < $b["ar"] )
13:         return -1;
14:     return 1;
15: }
16: usort( $stermek, "arHasonlito" );
17: foreach ( $stermek as $sertek )
18:     print $sertek["nev"] ":" $sertek["ar"] "<br>\n";
19: ?>
```

Először létrehozzuk a `$stermekek` tömböt, amelyet az egyes értékek ár mezője alapján szeretnénk rendezni. Ezután létrehozuk az `arHasonlito()` függvényt, amelynek két paramétere van, `$a` és `$b`. Ezek tartalmazzák azt a két tömböt, amely a `$stermekek` tömb második szintjét alkotja. Összehasonlítjuk az `"ar"` elemeket és ha a két ár azonos, 0-át, ha az első kevesebb, mint a másik, -1-et, egyébként pedig 1-et adunk vissza.

Miután meghatároztuk a rendező függvényt és a tömböt is, meghívhatjuk az `usort()` függvényt, amelynek átadjuk a `$stermekek` tömböt és az összehasonlító függvény nevét. Az `usort()` ismételten meghívva függvényünket, mindig átadja annak a `$stermekek` egy elemét és felcseréli az elemeket, a visszaadott értékeknek megfelelően. Végül végigléptetünk a tömbön, hogy megmutassuk az új elrendezést.

Az `usort()` függvényt számmal indexelt tömbök esetében használjuk. Ha más egyéni rendezést szeretnénk egy asszociatív tömbön végrehajtani, használjuk az `uasort()` függvényt. Az `uasort()` úgy rendez, hogy megtartja a kulcsok és az értékek közötti társítást is. A 16.2. program az `uasort()` használatát egy asszociatív tömb rendezésén keresztül mutatja be.

16.2. program Többdimenziós tömb rendezése mező alapján az `uasort()` függvénnyel

```
1: <?php
2: $stermekek = array(
3:     "HAL 2000" => array( "szin" =>"piros",
4:         "ar"=>4500.5 ),
5:     "Modem" => array( "szin" =>"kék",
6:         "ar"=>55.5 ),
7:     "Nyomtató" => array( "szin" =>"zöld",
8:         "ar"=>2200.5 ),
9:     "Csavarhúzó" => array( "szin" =>"piros",
10:        "ar"=>22.5 )
11: );
12: function arHasonlito( $a, $b )
13: {
14:     if ( $a["ar"] == $b["ar"] )
15:         return 0;
16:     if ( $a["ar"] < $b["ar"] )
17:         return -1;
18:     return 1;
19: }
20: uasort( $stermekek, "arHasonlito" );
```

16.2. program (folytatás)

```
17: foreach ( $stermek as $kulcs => $ertek )
18:     print "$kulcs: " $ertek["ar"] "<br>\n";
19: ?>
```

Az `uksort()` függvénnyel az asszociatív tömbökön a kulcsok alapján végezhetünk egyéni rendezést. Az `uksort()` pontosan ugyanúgy működik, mint az `usort()` és az `uasort()`, azzal a különbséggel, hogy az `uksort()` a tömb kulcsait hasonlítja össze.

A 16.3. programban az `uksort()` függvénnyel az egyes kulcsok karakterszáma alapján rendezünk egy tömböt. Megelőzve a következő óra anyagát, az `strlen()` függvénnyel állapítjuk meg a kulcsok hosszúságát. Az `strlen()` függvény bemenete egy karakterlánc, visszaadott értéke pedig annak hosszúsága karakterben mérve.

16.3. program Asszociatív tömb rendezése kulchosszúság alapján az `uksort()` függvénnyel

```
1: <?php
2: $ikszek = array(
3:     "xxxx" => 4,
4:     "xxx" => 5,
5:     "xx" => 7,
6:     "xxxxx" => 2,
7:     "x" => 8
8: );
9: function hosszHasonlito( $a, $b )
10: {
11:     if ( strlen( $a ) == strlen( $b ) )
12:         return 0;
13:     if ( strlen( $a ) < strlen( $b ) )
14:         return -1;
15:     return 1;
16: }
17: uksort( $ikszek, "hosszHasonlito" );
18: foreach ( $ikszek as $kulcs => $ertek )
19:     print "$kulcs: $ertek <br>\n";
20:
21: // a kimenet:
```

16.3. program (folytatás)

```
22: // x: 8
23: // xx: 7
24: // xxx: 5
25: // xxxx: 4
26: // xxxxx: 2
27:
28: ?>
```

Összefoglalás

Az óra során a tömbökkel és adattípusokkal kapcsolatos ismeretekben mélyedtünk el. Megtanultuk, mi történik, ha összetett adattípust skalárisá alakítunk és fordítva. Megtudtuk, hogyan kezeli a PHP a különböző adattípusokat egy kifejezésben, hogyan határozza meg automatikusan az eredmény adattípusát helyettünk. Megismerkedtünk számos függvénnyel; például az `is_array()`-jel, amely különféle adattípusokat ellenőriz, vagy az `intval()`-lal, amely az adatot egész értékűvé alakítja át. Megtanultuk, hogyan lehet a PHP-ben hagyományos módon tömböt bejárni, az `each()` és a `list()` függvénnyel. Az `in_array()` függvénnyel képesek vagyunk ellenőrizni, hogy létezik-e egy adott érték egy tömbben, és el tudunk távolítani egy elemet egy tömbből az `unset()` függvénnyel. Az `array_walk()` függvénnyel már egy tömb összes elemén is végezhetünk műveleteket, végül azt is tudjuk, hogyan használjuk az `usort()`, az `uasort()` és az `uksort()` függvényeket arra, hogy a tömbök egyéni rendezését végezzük.

Kérdések és válaszok

A PHP minden tömbkezelő függvényével megismerkedtünk?

Nem, az egész könyv sem lenne elég az összes tömbkezelő függvény bemutatásához. Teljes listájukat és leírásukat a <http://www.php.net/manual/ref.array.php> weboldalon találhatjuk.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik az a függvény, amellyel adattípusokat tetszőleges más adattípussá alakíthatunk?
2. Sikerülhet ez függvény nélkül is?
3. Mit ír ki a következő kód?

```
print "four" * 200;
```
4. Hogyan határoznánk meg, hogy egy adott változó tömb-e?
5. Melyik függvény adja vissza paraméterének értékét egész számként?
6. Hogyan ellenőrizzük, hogy egy változónak adtunk-e már értéket?
7. Hogyan ellenőrizzük, hogy egy változó üres értéket (például 0-át vagy üres karakterláncot) tartalmaz?
8. Melyik függvénnyel törölnénk egy tömb egy elemét?
9. Melyik függvénnyel rendeznénk egy számmal indexelt tömböt?

Feladatok

1. Nézzük végig még egyszer a könyv során megoldott feladatokat. Alakítsunk át minden foreach utasítást úgy, hogy az megfeleljen a PHP 3 követelményeinek is.
2. Hozzunk létre egy vegyes adattípusú tömböt. Rendeztessük a tömböt adattípusok szerint.



17. ÓRA

Karakterláncok kezelése

A Világháló valójában szöveges fájlokra épülő környezet, és igazából nem számít, mivel gazdagodik a jövőben a tartalma, a mélyén mindig szöveges állományokat fogunk találni. Így nem meglepő, hogy a PHP 4 sok olyan függvényt biztosít, amellyel szövegműveletek végezhetők.

Az óra során a következőket tanuljuk meg:

- Hogyan formázzunk karakterláncokat?
- Hogyan határozzuk meg a karakterláncok hosszát?
- Hogyan találjunk meg egy karakterláncon belüli karakterláncot?
- Hogyan bontsuk szét a karakterláncot alkotóelemeire?
- Hogyan távolítsuk el a szóközöket a karakterláncok végéről vagy elejéről?
- Hogyan cseréljünk le karakterlánc-részleteket?
- Hogyan változtassuk egy karakterláncban a kisbetűket nagybetűre és fordítva?

Karakterláncok formázása

A megjeleníteni kívánt karakterláncot eddig egyszerűen kiírtuk a böngészőbe. A PHP két olyan függvényt tartalmaz, amely lehetővé teszi az előzetes formázást, függetlenül attól, hogy tizedestörteket kell valahány tizedes pontosságra kerekíteni, egy mezőn belül kell jobbra vagy balra igazítani valamit, vagy egy számot kell különböző számrendszerekben megjeleníteni. Ebben a részben a `printf()` és az `sprintf()` függvények által biztosított formázási lehetőségekkel ismerkedünk meg.

A `printf()` függvény használata

Ha már dolgoztunk C-vel, biztosan ismerős lesz a `printf()` függvény, amelynek PHP-s változata hasonló, de nem azonos azzal. A függvény bemenete egy karakterlánc, más néven a formátumvezérlő karakterlánc (röviden formázó karakterlánc vagy egyszerűen formázó), emellett további, különböző típusú paraméterek. A formázó karakterlánc ezen további paraméterek megjelenítését határozza meg. A következő kódrészlet például a `printf()` függvényt használja, hogy egy egész számot decimális értékként írjon ki:

```
printf("az én számom az %d", 55 );  
// azt írja ki, hogy "az én számom az 55"
```

A formázó karakterláncban (ami az első paraméter) egy különleges kódot helyeztünk el, amely átalakítási meghatározásként ismert.

ÚJDONSÁG

Az átalakítási meghatározás százalékjellel (%) kezdődik, és azt határozza meg, hogyan kell a `printf()` függvény neki megfelelő paraméterét kezelni. Egyetlen formátumvezérlő karakterláncba annyi átalakítási meghatározást írhatunk, amennyit csak akarunk, feltéve, hogy a `printf()` függvénynek ugyanennyi paramétert adunk át a formázót követően.

A következő kódrészlet két számot ír ki a `printf()` használatával:

```
printf("Az első szám: %d<br>\nA második szám: %d<br>\n",  
      55, 66 );  
// A kiírt szöveg:  
// Az első szám: 55  
// A második szám: 66
```

Az első átalakítási meghatározás a `printf()` második paraméterének felel meg, ami ebben az esetben az 55. A következő átalakítási meghatározás a 66-nak felel meg. A százalékjelet követő `d` betű miatt a függvény az adatot decimális egészsként értelmezi. A meghatározásnak ez a része típusparaméterként ismeretes.

A printf() és a típusparaméterek

Egy típusparaméterrel már találkoztunk, ez volt a `d`, amely az adatot decimális formátumban jeleníti meg. A többi típusparamétert a 17.1. táblázatban láthatjuk.

17.1. táblázat Típusparaméterek

<i>Paraméter</i>	<i>Leírás</i>
<code>d</code>	A paramétert decimális számként jeleníti meg.
<code>b</code>	Egész számokat bináris számként jelenít meg.
<code>c</code>	Egy egész számot annak ASCII megfelelőjeként jelenít meg.
<code>f</code>	A paramétert lebegőpontos számként ábrázolja.
<code>o</code>	Egy egész számot oktális (8-as számrendszerű) számként jelenít meg.
<code>s</code>	A paramétert karakterlánc-állandónak tekinti.
<code>x</code>	Egy egész számot kisbetűs hexadecimális (16-os számrendszerű) számként jelenít meg.
<code>X</code>	Egy egész számot nagybetűs hexadecimális (16-os számrendszerű) számként jelenít meg

A 17.1. program a `printf()` függvénnyel egy számot a 17.1. táblázat típusparaméterei segítségével jelenít meg.

Vegyük észre, hogy a formázó karakterlánc nem egyszerűen csak átalakítási meghatározásokat tartalmaz, minden további benne szereplő szöveg kiírásra kerül.

17.1. program Néhány típusparaméter használatának bemutatása

```
1: <html>
2: <head>
3: <title>17.1. program Néhány típusparaméter
   használatának bemutatása</title>
4: </head>
5: <body>
6: <?php
7: $szam = 543;
8: printf("Decimális: %d<br>", $szam );
9: printf("Bináris: %b<br>", $szam );
```

17.1. program (folytatás)

```
10: printf("Kétszeres pontosságú: %f<br>", $szam );  
11: printf("Oktális: %o<br>", $szam );  
12: printf("Karakterlánc: %s<br>", $szam );  
13: printf("Hexa (kisbetűs): %x<br>", $szam );  
14: printf("Hexa (nagybetűs): %X<br>", $szam );  
15: ?>  
16: </body>  
17: </html>
```

A 17.1. program kimenetét a 17.1. ábrán láthatjuk. A `printf()` függvénnyel gyorsan tudunk adatokat egyik számrendszerből a másikba átalakítani és az eredményt megjeleníteni.

17.1. ábra

Néhány típusparaméter használatának bemutatása



Ha a HTML-ben hivatkoznunk kell egy színre, három 00 és FF közé eső, a vörös, zöld és kék színt képviselő hexadecimális számot kell megadnunk. A `printf()` függvényt használhatjuk arra, hogy a három 0 és 255 közé eső decimális számot azok hexadecimális megfelelőire alakítsuk át:

```
$piros = 204;  
$zold = 204;  
$kek = 204;  
printf( "#%X%X%X", $piros, $zold, $kek );  
// azt írja ki, hogy "#CCCCC"
```


Bár a típusparaméterrel a decimális számokat hexadecimálissá alakíthatjuk, azt nem tudjuk meghatározni, hogy az egyes paraméterek kimenete hány karaktert foglaljon el. A HTML színkódjában minden hexadecimális számot két karakteresre kell kitölteni, ami problémát okoz, ha például az előző kódrészlet `$piros`, `$zold`, `$kek` változóit úgy módosítjuk, hogy 1-et tartsanak. Kimenetül `"#111"`-et kapnánk. A bevezető nullák használatát egy kitöltő paraméter segítségével biztosíthatjuk.

A kitöltő paraméter

Beállíthatjuk, hogy a kimenet bizonyos karakterekkel megfelelő szélességűre töltődjön ki. A kitöltő paraméter közvetlenül az átalakító paramétert kezdő százalékjelet követi. Ha a kimenetet bevezető nullákkal szeretnénk kitölteni, a kitöltő paraméterben a 0 karaktert az a szám követi, ahány karakterre szeretnénk a kimenetet bővíteni. Ha a kimenet hossza ennél a számnál kisebb lenne, a különbség nullákkal kerül kitöltésre, ha nagyobb, a kitöltő paraméternek nincs hatása:

```
printf( "%04d", 36 )  
// a kimenet "0036" lesz  
printf( "%04d", 12345 )  
// a kimenet "12345" lesz
```

Ha a kimenetet bevezető szóközzel szeretnénk kitölteni, a kitöltő paraméternek tartalmaznia kell egy szóköz karaktert, amelyet a kimenet elvárt karakterszáma követ:

```
printf( "% 4d", 36 )  
// azt írja ki, hogy " 36"
```



Bár a HTML dokumentumokban egymás után szereplő több szóközt a böngészők nem jelenítik meg, a megjelenítendő szöveg elé és után helyezett `<PRE>` címkével mégis biztosíthatjuk a szóközök és sortörések megjelenítését.

```
<pre>
<?php
print "A      szóközök      láthatóvá válnak."
?>
</pre>
```

Ha teljes dokumentumot szeretnénk szöveggént megformázni, a `header()` függvényt használhatjuk a dokumentumtípus (Content-Type) fejlécének módosításához.

```
header("Content-type: text/plain");
```

Ne feledjük, hogy programunk nem küldhet semmilyen kimenetet a böngészőnek a `header()` függvényhívást megelőzően, hogy a megfelelő módon működjön, mivel a kimenet megkezdésekor a válasz fejrészét már elküldtük a böngészőnek.

Ha szóközön vagy nullán kívül más karaktert szeretnénk a kitöltéshez használni, a kitöltő paraméteren belül a kitöltő karakter elé írunk egyszeres idézőjelet:

```
printf( "%'x4d", 36 )
// azt írja ki "xx36"
```

Most már rendelkezésünkre állnak azok az eszközök, melyekkel a korábbi HTML kódot kiegészíthetjük. Eddig ugyan már át tudtuk alakítani a három számot, de nem tudtuk bevezető nullákkal kitölteni azokat:

```
$piros = 1;
$zold = 1;
$kek = 1;
printf( "#%02X%02X%02X", $piros, $zold, $kek );
// azt írja ki, hogy "#010101"
```

Most már minden változó hexadecimális számként fog megjelenni. Ha a kimenet két karakternél rövidebb, a hiányt bevezető nullák pótolják.

A mezőszélesség meghatározása

Meghatározhatjuk a kimenet által elfoglalt mező szélességét is. A mezőszélesség paramétere egy olyan egész szám, amely a százalékjel után következik az átalakító paraméterben (feltéve, hogy nem használunk helykitöltő karaktereket). A következő kódrészlet kimenete egy négyelemű felsorolás, amelynek mezőszélessége 20 karakternyi. A szóközök láthatóvá tételéhez a kimenetet egy PRE elembe ágyazzuk.

```
print "<pre>";
printf("%20s\n", "Könyvek");
printf("%20s\n", "CDk");
printf("%20s\n", "Játékok");
printf("%20s\n", "Magazinok");
print "</pre>";
```

A 17.2. ábrán a fenti kódrészlet eredményét láthatjuk.

17.2. ábra

Igazítás a mezőszélességhez



A kimenet alapértelmezés szerint a mezőn belül jobbra igazodik. A balra igazítást a mezőszélesség paramétere elé tett mínuszjellel (-) érhetjük el.

```
printf ("%20s|<- eddig tart a balra zárás\n", "Balra zárt");
```

Fontos megjegyezni, hogy ha lebegőpontos számot írunk ki, az igazítás csak a kimenetben levő szám (ponttól jobbra levő) tizedesrészére vonatkozik. Más szóval jobbra igazításkor a lebegőpontos számnak a tizedesponttól balra eső része a mező bal oldali, túlsó végén marad.

A pontosság meghatározása

Ha az adatot lebegőpontos formában szeretnénk megjeleníteni, meghatározhatjuk a kerekítés pontosságát. Ez főleg a pénznem átváltásokor szokott fontos lenni. A pontosságot meghatározó paraméter egy pontból és egy számból áll, és közvetlenül a típusparaméter elé kell írni. A megadott szám határozza meg, hány tizedesre szeretnénk kerekíteni. Ez a paraméter csak akkor érvényes, ha a kimenet f típusparaméterű:

```
printf ("%2f\n", 5.333333);  
// azt írja ki, hogy "5.33"
```



A C nyelv printf() függvényében akkor is lehetőségünk van a pontosság megadására, ha decimális kimenet esetén kérünk kitöltést. A PHP 4-ben a pontosság paraméterének nincs semmilyen hatása a decimális kimenetre. Egész számok nullákkal való bevezetéséhez a kitöltő paramétert kell használnunk.

Átalakító paraméterek (Ismétlés)

A 17.2. táblázatban az átalakító paramétereket foglaljuk össze. Megjegyzendő, hogy a két (kitöltő és mezőszélesség) paraméter együttes használata bonyolult, így azt tanácsoljuk, egyszerre csak az egyiket használjuk.

17.2. táblázat Az átalakítás lépései

Név	Leírás	Példa
Kitöltő paraméter	A kimenet által elfoglalandó karakter-számot és a kitöltésre használt karaktert adja meg.	' 4 '
Mezőszélesség paraméter	A formázandó karakterlánc méretét adja meg.	' 20 '
Pontosság paraméter	Meghatározza, hogy a kétszeres pontosságú számokat hány tizedesre kell kerekíteni.	' .4 '
Típusparaméter	Meghatározza az eredmény adattípusát.	' d '

A 17.2. program a `printf()` használatával egy termékárlistát hoz létre.

17.2. program Termékárlista formázása a `printf()` függvénnyel

```
1: <html>
2: <head>
3: <title>17.2. program Termékárlista formázása
   a printf() függvénnyel</title>
4: </head>
5: <body>
6: <?php
7: $stermekek = Array("Zöld karosszék"=>"222.4",
8:                    "Gyertyatartó" => "4",
9:                    "Kávézóasztal" => "80.6"
10:                  );
11: print "<pre>";
12: printf("%-20s%23s\n", "Név", "Ár");
13: printf("%'-43s\n", "");
14: foreach ( $stermekek as $kulcs=>$ertek )
15:     printf( "%-20s%20.2f\n", $kulcs, $ertek );
16: print "</pre>";
17: ?>
18: </body>
19: </html>
```

Először a termékek nevét és árát tartalmazó tömböt adjuk meg. Egy PRE elemmel jelezzük a böngészőnek, hogy a szöközőket és a soremeléseket értelmezze. A `printf()` első meghívása a következő formázó karakterláncot adja meg:

```
"%-20s%23s\n"
```

Az első átalakító meghatározás ("`%-20s`") a mezőszélességet balra zárt 20 karakterre állítja. Ebben a mezőben egy karakterlánc típusú paramétert helyezünk el. A második meghatározás ("`%23s`") egy jobbra zárt mezőszélességet ad meg. A `printf()` függvénynek ez a meghívása hozza létre leendő táblázatunk fejlécét.

A `printf()` második meghívásával egy 43 karakter hosszú, mínuszjelekből (-) álló vonalat húzunk. Ezt úgy érjük el, hogy egy üres karakterláncot kitöltő paraméterrel jelenítünk meg.

A `printf()` legutolsó meghívása annak a `foreach` utasításnak a része, amely végiglépked a termékek tömbjén. Két átalakító meghatározást használunk.

Az első ("%20s") a termék nevét írja ki egy 20 karakteres mezőbe, balra igazítva. A másik ("%20.2f") a mezőszélesség paraméterrel azt biztosítja, hogy az eredmény egy 20 karakteres mezőben jobbra igazítva jelenjen meg, a pontossági paraméterrel pedig azt, hogy a megjelenített kétszeres pontosságú érték két tizedesre legyen kerekítve.

A 17.3. ábrán a 17.2. program eredményét láthatjuk.

17.3. ábra

Termékárlista formázása a printf() függvényvel



The screenshot shows a terminal window with a table of product prices. The table has two columns: 'Név' (Name) and 'Ár' (Price). The data is as follows:

Név	Ár
0000 Termékneve	223.40
000000000000	4.00
000000000000	80.00

Formázott karakterlánc tárolása

A `printf()` az adatokat a böngészőben jeleníti meg, ami azzal jár, hogy eredménye programunk számára nem elérhető. Használhatjuk azonban a `sprintf()` függvényt is, amelynek használata megegyezik a `printf()` függvényével, viszont az eredményt egy karakterláncban adja vissza, amelyet aztán későbbi használatra egy változóba helyezhetünk. A következő kódrészlet a `sprintf()` függvény segítségével egy lebegőpontos értéket két tizedesre kerekít és az eredményt a `$kerek` változóban tárolja:

```
$kerek = sprintf("%.2f", 23.34454);  
print "Még $kerek forintot költhetsz";
```

A `sprintf()` függvény egy lehetséges felhasználása, hogy vele a megformázott adatot fájlban lehet tárolni. Ennek az a módja, hogy először meghívjuk a `sprintf()` függvényt, a visszaadott értékét egy változóban tároljuk, majd az `fputs()` függvényvel fájlba írjuk.

Részletesebben a karakterláncokról

Nem minden esetben tudunk mindent az adatokról, amelyekkel dolgozunk. A karakterláncok többféle forrásból is érkehetnek, beviheti a felhasználó, érkehetnek adatbázisokból, fájlokból és weboldalakról. A PHP 4 több függvénye segítségünkre lehet abban, hogy ezekről a külső forrásból érkező adatokról többet tudjunk meg.

Szövegek indexelése

A karakterláncokkal kapcsolatban gyakran használjuk az indexelés szót, de a tömböknél még gyakrabban találkozhatunk vele. Valójában a karakterláncok és a tömbök nem állnak olyan messze egymástól, mint azt gondolnánk. Egy karakterláncot elképzelhetünk egy karakterekből álló tömbként is. Ennek megfelelően a karakterláncok egyes karaktereihez ugyanúgy férhetünk hozzá, mint egy tömb elemeihez:

```
$proba = "gazfickó";  
print $proba[0]; // azt írja ki, hogy "g"  
print $proba[2]; // azt írja ki, hogy "z"
```

Ne felejtjük el, hogy amikor egy karakterláncon belüli karakter indexéről vagy helyéről beszélünk, akkor a karaktereket – ugyanúgy, mint a tömb elemeit – 0-tól kezdve számozzuk. A tömbökkel kapcsolatos félreértések elkerülése érdekében a PHP 4 bevezette a `$proba{0}` formát is erre a célra.

Szöveg hosszának megállapítása az `strlen()` függvénnyel

Az `strlen()` függvény segítségével megállapíthatjuk egy karakterlánc hosszát. A függvény bemenete egy karakterlánc, visszatérési értéke pedig egy egész szám, amely a függvénynek átadott változó karaktereinek száma. Ez a függvény például kimondottan jól jöhet a felhasználó által megadott bemenet hosszának ellenőrzésére. A következő kódrészlettel ellenőrizhetjük, hogy a tagnyilvántartó azonosító négykarakteres-e:

```
if ( strlen( $tagazonosito ) == 4 )  
    print "Köszönöm!";  
else  
    print "Az azonosítónak négykarakteresnek kell lennie<P>";
```

Ha a `$tagazonosito` globális változó értéke négykarakteres, megköszönjük a felhasználónak, hogy rendelkezésünkre bocsátotta, más esetben hibaüzenetet jelenítünk meg.

Szövegrész megkeresése az `strstr()` függvénnyel

Ezzel a függvénnyel azt állapíthatjuk meg, hogy egy karakterlánc megtalálható-e beágyazva egy másik karakterláncban. Az `strstr()` függvény két paramétert kap bemenetül: a keresendő szöveget és a forrásláncot, azaz azt a karakterláncot, amelyben keresnie kell. Ha a keresett karakterlánc nem található a szövegben, a visszatérési érték `false` (hamis) lesz, ellenkező esetben a függvény a forrásláncból a keresett karakterlánccal kezdődő részt adja vissza. A következő példában megkülönböztetetten kezeljük azokat a tagazonosítókat, amelyekben megtalálható az AB karakterlánc:

```
$tagazonosito = "pAB7";  
if ( strstr( $tagazonosito, "AB" ) )  
    print "Köszönöm. Ne feledje, hogy tagsága hamarosan  
        lejár!";  
else  
    print "Köszönöm!";
```

Mivel a `$tagazonosito` változó tartalmazza az AB karakterláncot, az `strstr()` az AB7 karakterláncot adja vissza. Ez `true` (igaz) eredménynek számít, így egy különleges üzenetet ír ki. Mi történik akkor, ha a felhasználó a "pab7" karakterláncot adja meg? Az `strstr()` megkülönbözteti a kis- és nagybetűket, így nem találja meg az AB karakterláncot. Az `if` utasítás feltétele nem válik igazzá, így a szokványos üzenet kerül a böngészőhöz. Ha vagy AB-t, vagy ab-t szeretnénk kerestetni a szövegben, használjuk az `stristr()` függvényt, melynek működése azonos az `strstr()`-ével, viszont nem különbözteti meg a kis- és nagybetűket.

Részlánc elhelyezkedésének meghatározása az `strpos()` függvénnyel

Az `strpos()` függvény segítségével kideríthetjük, hogy egy szöveg megtalálható-e egy másik szöveg részeként, és ha igen, hol. Bemenetét két paraméter képezi, a forráslánc, amelyben keresünk, és a karakterlánc, amelyet keresünk. Ezek mellett létezik még egy harmadik, nem kötelező paraméter, mégpedig azt az indexet megadó egész szám, amelytől kezdve keresni szeretnénk. Ha a keresett karakterlánc nem található, a függvény a `false` (hamis) értéket adja vissza, ellenkező esetben azt az egész számot, mely indextől a keresett szöveg kezdődik. A következő programrészletben az `strpos()` függvénnyel győződünk meg arról, hogy egy karakterlánc az `mz` karakterekkel kezdődik-e:

```
$tagazonosito = "mz00xyz";  
if ( strpos($tagazonosito, "mz") === 0 )  
    print "Üdv, mz.";
```


Vegyük szemügyre azt a trükköt, amellyel a kívánt eredményt kaptuk. Az `strpos()` megtalálja az `mz` karaktersort a lánc elején. Ez azt jelenti, hogy `0` értéket ad vissza, ami viszont a kifejezés kiértékelése során hamis értéket eredményezne. Hogy ezt elkerüljük, a PHP `4` új azonosság műveleti jelét (`===`) alkalmazzuk, amely akkor ad vissza `true` (igaz) értéket, ha bal és jobb oldali operandusa egyenlő értékű és azonos típusú is egyben.

Szövegrészlet kinyerése a `substr()` függvénnyel

A `substr()` függvény egy kezdőindextől kezdve meghatározott hosszúságú karakterláncot ad vissza. Bemenetét két paraméter képezi, az egyik a forráslánc, a másik a kezdőindex. A függvény az összes karaktert visszaadja a kezdőindextől a forráslánc végéig. Harmadik, nem kötelező paramétere egy egész szám, amely a visszaadandó szöveg hosszát jelenti. Ha ezt is megadjuk, a függvény csak a meghatározott mennyiségű karaktert adja vissza a kezdőindextől számítva.

```
$proba = "gazfickó";  
print substr($proba,3); // azt írja ki "fickó"  
print substr($proba,3,4); // azt írja ki "fick"
```

Ha kezdőindexként negatív számot adunk meg, a függvény nem a karakterlánc elejétől számolja a karaktereket, hanem a végétől. A következő kódrészlet meghatározott üzenetet ír ki azoknak, akiknek e-mail címe `.hu`-val végződik.

```
$cim = "felhasznalo@szolgaltato.hu"  
if ( substr( $cim, -3 ) == ".hu" )  
    print "Ne felejtse el magyar vásárlóinknak járó  
        speciális ajánlatainkat!";  
else  
    print "Üdvözzöljük üzletünkben!";
```

Karakterlánc elemekre bontása az `strtok()` függvénnyel

Az `strtok()` függvénnyel karakterláncok szintaktikai elemzését végezhetjük. A függvény legelső meghívásakor két paramétert vár, az elemzendő karakterláncot és egy határolójelet, amely alapján a karakterláncot felbontja. A határolójel tetszőleges számú karakterből állhat. A függvény első meghívásakor átmenetileg a memóriába helyezi a teljes forrásláncot, így a további meghívások alkalmával már csak a határolójelet kell megadnunk. Az `strtok()` minden meghívásakor a következő megtalált elemet adja vissza, a karakterlánc végére érkezést a `false` (hamis) érték visszaadásával jelzi. Ezt a függvényt legtöbbször cikluson belül használjuk. A 17.3. program egy URL-t bont elemeire: először leválasztja a gazdagépet és az elérési útvonalat a karakterláncról, majd a változó-érték párokat bontja szét. A 17.3. program eredményét a 17.3. ábrán láthatjuk.

17.3. program Karakterlánc elemekre bontása az strtok() függvénnyel

```
1: <html>
2: <head>
3: <title>17.3. program Karakterlánc elemekre bontása
4:         az strtok() függvénnyel</title>
5: </head>
6: <body>
7: <?php
8: $proba = "http://www.deja.com/qs.xp?
9: OP=dnquery.xp&ST=MS&DBS=2&QRY=developer+php";
10: $hatarolo = "&";
11: $szo = strtok( $proba, $hatarolo );
12: while ( is_string( $szo ) )
13: {
14:     if ( $szo )
15:         print "$szo<br>";
16:     $szo = strtok( $hatarolo );
17: }
18: ?>
19: </body>
20: </html>
```

Az `strtok()` függvény önmagában nem sok mindent támogat, így csak különféle trükkökkel bírhatjuk igazán hasznos munkára. Először a `$hatarolo` változóban tároljuk a határolójelet. Meghívjuk az `strtok()` függvényt, átadjuk neki az elemzendő URL-t és a `$hatarolo` karakterláncot. Az első eredményt a `$szo` változóba helyezzük. A `while` ciklus feltételében azt ellenőrizzük, hogy a `$szo` karakterlánc-e. Ha nem az, tudhatjuk, hogy elértük az URL végét és a feladat befejeződött.

Azért ellenőrizzük a visszaadott érték típusát, mert ha egy karakterlánc egy sorában két határolójelet van, az `strtok()` az első határolójelet elérésekor üres karakterláncot ad vissza. Így, ha a `$szo` egy üres karakterlánc, az alábbi példában látható szokványosabb próbálkozás sikertelen lesz, még akkor is, ha a függvény még nem érte el a forráslánc végét, mivel a `while` feltétele hamissá válik:

```
while ( $szo )
{
    $szo = strtok( $hatarolo );
}
```

Ha meggyőződünk róla, hogy a `$szo` változó karakterláncot tartalmaz, elkezdhetünk dolgozni vele. Ha a `$szo` nem üres karakterlánc, megjelenítjük a böngészőben. Aztán újra meg kell hívnunk a `strtok()` függvényt, hogy a `$szo` változót újabb karakterláncokkal töltsse fel a következő kiértékeléshez. Vegyük észre, hogy a második alkalommal nem adtuk át a függvénynek a forrásláncot. Ha ezt mégis megtennénk, újra a forráslánc első szavát találná meg, így végtelen ciklusba kerülnénk.

A karakterláncok kezelése

A PHP 4 a karakterlánc paraméterek kisebb-nagyobb átalakításához számos függvényt biztosít.

Szöveg tisztogatása a `trim()` típusú függvényekkel

Ha egy felhasználótól vagy fájlból kapunk információt, sohasem lehetünk biztosak benne, hogy az adat előtt vagy után nincs egy vagy több fölösleges elválasztó karakter. A `trim()` függvény ezeket az elválasztó karaktereket (soremelés, tabulátorjel, szóköz stb.) hagyja el a karakterlánc elejéről és végéről. Bemenete a megtisztítandó szöveg, kimenete pedig a megtisztított.

```
$szoveg = "\t\t\teléggé levegős ez a szöveg";  
$szoveg = trim( $szoveg );  
print $szoveg  
// azt írja ki, hogy "eléggé levegős ez a szöveg"
```

Persze lehet, hogy ez a túlbuzgó függvény nem a legmegfelelőbb számunkra. Elképzelhető, hogy a szöveg elején levő elválasztó karaktereket meg szeretnénk tartani és csak a szöveg végéről akarjuk eltávolítani azokat. Pontosan erre a feladatra találták ki a `chop()` függvényt. Vigyázzunk, ha Perl ismeretekkel is rendelkezünk, mivel ott a `chop()` függvény egy kicsit más jelentéssel bír. A probléma áthidalására a PHP fejlesztői ugyanezen szolgáltatás eléréséhez megadták az `rtim()` függvénynevet is.

```
$szoveg = "\t\t\teléggé levegős ez a szöveg";  
$szoveg = chop( $szoveg );  
print $szoveg  
// azt írja ki, hogy "          eléggé levegős ez a szöveg"
```

Lehetőségünk van az `ltrim()` függvény használatára is, amely az elválasztó karaktereket csak a karakterlánc elejéről távolítja el. Ahogy az előzőeknél is, bemenete az átalakítandó szöveg, kimenete pedig az elválasztó karakterektől csupán a bal oldalán mentes karakterlánc:

```
$szoveg = "\t\tteléggé levegős ez a szöveg" ;  
$szoveg = ltrim( $szoveg );  
print $szoveg  
// azt írja ki, hogy "eléggé levegős ez a szöveg"
```

Karakterlánc részének lecserélése a substr_replace() függvénnyel

A `substr_replace()` hasonlóan működik, mint a `substr()`, a különbség abban rejlik, hogy itt lehetőség nyílik a kivonatolt karakterlánc-részlet lecserélésére is. A függvény három paramétert vár: az átalakítandó karakterláncot, a csereszöveget és a kezdőindexet. Ezek mellett létezik egy nem kötelező, negyedik hosszúság paraméter is. A `substr_replace()` függvény megtalálja a kezdőindex és a hosszúság paraméter által meghatározott részláncot, lecseréli azt a csereszövegre, és a teljes átalakított karakterláncot adja vissza.

A következő kódrészletben egy felhasználó tagazonosítójának megújításához le kell cserélnünk annak harmadik és negyedik karakterét:

```
<?  
$tagazonosito = "mz99xyz";  
$tagazonosito = substr_replace( $tagazonosito, "00", 2, 2 );  
print "Az új tagnyilvántartó azonosító: $tagazonosito<p>";  
// azt írja ki, hogy "Az új tagnyilvántartó azonosító:  
mz00xyz"  
?>
```

Az összes részlánc lecserélése az str_replace() függvénnyel

Az `str_replace()` függvény a keresett karakterlánc-rész összes előfordulását lecseréli egy másik karakterláncra. Bemenetének három paramétere a lecserélendő karakterlánc, a csereszöveg és a forrásszöveg. A függvény kimenete az átalakított karakterlánc. A következő példában egy karakterláncban az 1999 összes előfordulását 2000-re cseréljük:

```
$karakterlanc = "Ezt az oldal 1999-ben szerzői jog által  
védett";  
$karakterlanc .= "Felsőoktatási tájékoztató 1999";  
print str_replace("1999", "2000", $karakterlanc);
```

Kis- és nagybetűk közti váltás

A PHP több függvénnyel is segítségünkre van a kis- és nagybetűk cseréjében. Amikor felhasználók által beírt adattal dolgozunk, fontos lehet mindent csupa nagybetűsre vagy csupa kisbetűsre alakítani, hogy aztán könnyebben összehasonlíthatóak legyenek. Az `strtoupper()` függvény segítségével egy karakterláncot csupa nagybetűsre alakíthatunk. A függvény egyetlen bemenete az átalakítandó szöveg, visszatérési értéke pedig a csupa nagybetűs karakterlánc:

```
$tagazonosito = "mz00xyz";
$tagazonosito = strtoupper( $tagazonosito );
print "$tagazonosito<P>"; // azt írja ki, hogy "MZ00XYZ"
```

Karakterláncunk csupa kisbetűssé való alakításához használjuk az `strtolower()` függvényt. Ennek egyetlen bemenete az átalakítandó szöveg és a csupa kisbetűs karakterláncot adja vissza:

```
$honlap_url = "WWW.KISKAPU.HU";
$honlap_url = strtolower( $honlap_url );
if ( ! ( strpos ( $honlap_url, "http://" ) === 0 ) )
    $honlap_url = "http://$honlap_url";
print $honlap_url; // azt írja ki, hogy
                  "http://www.kiskapu.hu"
```

A PHP-nek van egy nagyszerű, „tüneti kezelést” biztosító függvénye, az `ucwords()`. Ez a függvény egy karakterlánc minden szavának első betűjét teszi nagybetűssé. A következő programrészletben a felhasználó által beírt karakterláncban a szavak első betűjét nagybetűre cseréljük:

```
$teljes_nev = "vitéz tinódi lantos sebestyén";
$teljes_nev = ucwords( $teljes_nev );
print $teljes_nev; // azt írja ki, hogy "Vitéz Tinódi
                          Lantos Sebestyén"
```

A függvény kizárólag a szavak első betűjét cseréli le, így ha a felhasználónak nehézségei vannak a SHIFT billentyűvel és azt írta be, hogy "ViTÉz tINóDi laNtos sEBeStYén", ez a függvény nem sokban lesz segítségére, hiszen a tüneti kezelés eredménye "ViTÉz TiNóDi LaNtos SEBeStYén" lesz. Ezen úgy segíthetünk, hogy az `ucwords()` meghívása előtt az `strtolower()` függvénnyel először csupa kisbetűssé alakítjuk a karakterláncot:

```
$teljes_nev = "ViTÉz tINóDi laNtos sEBeStYén";
$teljes_nev = ucwords( strtolower($teljes_nev) );
print $teljes_nev; // azt írja ki, hogy "Vitéz Tinódi
                          Lantos Sebestyén"
```

Fel kell, hogy hívjuk a kedves olvasó figyelmét arra, hogy a magyar szövegekkel az ékezetes betűk miatt alapbeállításban problémáink akadhatnak. A nemzeti beállítások testreszabására használható `setlocale()` függvényt kell alkalmaznunk, hogy a kívánt eredményt elérjük.

Karakterláncok tömbbé alakítása az `explode()` függvénnyel

A mókásan „robbantó”-nak nevezett függvény bizonyos mértékben hasonló az `strtok()` függvényhez. Ez a függvény azonban egy karakterláncot tömbbé bont fel, amit aztán tárolhatunk, rendezhetünk, vagy azt tehetünk vele, amit csak szeretnénk. Bemenetét két paraméter alkotja, ez egyik egy határolójel, ami alapján fel szeretnénk bontani a forrásláncot, a másik maga a forráslánc. A határolójel több karakterből is állhat, ezek együtt fogják alkotni a határolójelet. (Ez eltér az `strtok()` függvény működésétől, ahol a megadott karakterlánc minden karaktere egy-egy önálló határolójel lesz.) A következő kódrészlet egy dátumot bont fel részekre és az eredményt egy tömbben tárolja:

```
$kezdet = "2000.12.01";  
$datum_tomb = explode(".", $kezdet);  
// $datum_tomb[0] == "2000"  
// $datum_tomb[1] == "12"  
// $datum_tomb[2] == "00"
```

Összefoglalás

A PHP külvilággal való kapcsolattartása és adattárolása leginkább karakterláncokon keresztül valósul meg. Az órán a programjainkban levő karakterláncok kezelésével ismerkedtünk meg.

A `printf()` és az `sprintf()` függvények segítségével megtanultuk formázni a karakterláncokat. Ezt a két függvényt olyan karakterláncok létrehozására használjuk, amelyek egyrészt átalakítják, másrészt el is rendezik az adatokat. Tanultunk olyan függvényekről, amelyek információt árulnak el a karakterláncokról. Meg tudjuk állapítani egy karakterlánc hosszúságát az `strlen()`, egy részlánc jelenlétét az `strpos()` függvénnyel, vagy kiszakíthatunk részláncokat az `strtok()` segítségével.

Végül azokról a függvényekről tanultunk, amelyek karakterláncokat alakítanak át. Most már el tudjuk tüntetni az elválasztó karaktereket a `trim()`, `ltrim()` vagy a `chop()` függvénnyel, válthatunk a kis- és nagybetűk között az `strtoupper()`, az `strtolower()` és az `ucwords()` függvényekkel, valamint egy karakterlánc összes előfordulását lecserélhetjük az `str_replace()` segítségével.

Ezek után nehéz elhinni, de még mindig nem végeztünk a karakterláncokkal. A PHP ugyanis támogatja a szabályos kifejezéseket, amelyek a karakterlánc-kezelés még hatékonyabb formáját biztosítják. A szabályos kifejezések alkotják a következő óra anyagát.

Kérdések és válaszok

Vannak még egyéb hasznos karakterlánc-függvények?

Vannak. A PHP több mint 60 ilyen függvénnyel rendelkezik! Ezekről a PHP 4 kézikönyvében olvashatunk, amelynek megfelelő része a `http://php.net/manual/ref.strings.php` címen található.

A `printf()` működését bemutató példában a formázást úgy jelenítettük meg, hogy a kimenetet `<PRE>` elemek közé tettük. Ez a legjobb módja a formázott szöveg megjelenítésének a böngészőben?

A `<PRE>` címkék akkor szükségesek, ha a sima szöveg (`plain text`) formázást HTML-es környezetben is meg szeretnénk tartani. Ha viszont a teljes dokumentumot így szeretnénk megjeleníteni, a legokosabb, ha közöljük a böngészővel, hogy sima szöveggént formázza meg azt. Ez a `header()` függvénnyel érhető el:

```
Header("Content-type: text/plain");
```

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Milyen átalakító paramétert használnánk a `printf()` függvényben egy egész szám lebegőpontos számként való megformázására?
2. Hogyan egészítsük ki az 1. kérdésben átalakított számot nullákkal úgy, hogy a tizedespont előtti (attól balra eső) rész 4 karakter hosszúságú legyen?
3. Hogyan kerekítenénk az előző kérdés lebegőpontos számát két tizedesjegyre?
4. Milyen függvényeket használnánk egy szöveg hosszának kiderítéséhez?
5. Milyen függvényeket használnánk egy részlánc más karakterláncon belüli kezdetének meghatározására?

6. Milyen függvényeket használnánk arra, hogy egy szövegből kivonjuk annak egy darabját?
7. Hogyan távolíthatjuk el az elválasztó karaktereket egy karakterlánc elejéről?
8. Hogyan alakítanánk át egy karakterláncot csupa nagybetűsre?
9. Hogyan bontanánk fel egy határolójelekkel elválasztott karakterláncot tömbökre?

Feladatok

1. Hozzunk létre egy olyan vélemény-visszajelző űrlapot, amely a felhasználó nevét és e-mail címét kéri be. Használjuk a kis- és nagybetűket átalakító függvényeket a nevek első betűjének nagybetűsítésére, majd jelenítsük meg az eredményt a böngészőben. Ellenőrizzük, hogy a cím tartalmaz-e @-jelet, ha nem, figyelmeztessük a felhasználót.
2. Hozzunk létre egy lebegőpontos és egész számokból álló tömböt. Léptessünk végig a tömbön és kerekítsük az összes lebegőpontos számot két tizedesjegyre. Igazítsuk jobbra a kimenetet egy 20 karakter szélességű mezőben.



18. ÓRA

A szabályos kifejezések használata

A szövegek vizsgálatának és elemzésének egyszerű módja a szabályos kifejezések (regular expressions) használata. Ezek lehetővé teszik, hogy egy karakterláncon belül mintákat keressünk, a találatokat pedig rugalmasan és pontosan kapjuk vissza. Mivel hatékonyabbak az előző órában tanult karakterlánc-kezelő függvényeknél, lassabbak is azoknál. Így azt tanácsoljuk, hogy ha nincs különösebb szükségünk a szabályos kifejezések által biztosított hatékonyságra, inkább használjuk a hagyományos karakterlánc-kezelő függvényeket.

A PHP a szabályos kifejezések két típusát támogatja. Egyrészt támogatja a Perl által használtakat, az azoknak megfelelő függvényekkel, másrészt a korlátozottabb tudású POSIX szabályos kifejezés formát. Mindkettővel meg fogunk ismerkedni.

Az óra során a következőkről tanulunk:

- Hogyan illesszünk mintát egy karakterláncra a szabályos kifejezések segítségével?
- Mik a szabályos kifejezések formai követelményei?

- Hogyan cseréljük le karakterláncokat szabályos kifejezésekkel?
- Hogyan keressünk és cseréljük le mintákat egy szövegben a hatékony Perl típusú szabályos kifejezésekkel?

A POSIX szabályos kifejezések függvényei

A POSIX szabályos kifejezéseket kezelő függvények lehetővé teszik, hogy egy szövegben bonyolult mintákat keressünk és cseréljük le. Ezeket általában egyszerűen szabályoskifejezés-függvényeknek szokták nevezni, mi azonban POSIX szabályoskifejezés-függvényekként utalunk rájuk, egyrészt azért, hogy megkülönböztethessük őket a hasonló, ám hatékonyabb Perl típusú szabályos kifejezésektől, másrészt azért, mert a POSIX bővített szabályos kifejezés (extended regular expression) szabványát követik.

A szabályos kifejezések olyan jelegyüttesek, amelyek egy szövegben egy mintára illeszkednek. Használatuk elsajátítása így jóval többet jelent annál, hogy megtanuljuk a PHP szabályoskifejezés-függvényeinek be- és kimeneti paramétereit. Az ismerkedést a függvényekkel kezdjük és rajtuk keresztül vezetjük be az olvasót a szabályos kifejezések formai követelményeibe.

Minta keresése karakterláncokban az `ereg()` függvénnyel

Az `ereg()` bemenete egy mintát jelképező karakterlánc, egy karakterlánc, amiben keresünk, és egy tömbváltozó, amelyben a keresés eredményét tároljuk. A függvény egy egész számot ad vissza, amely a megtalált minta illeszkedő karaktereinek számát adja meg, illetve ha a függvény nem találja meg a mintát, a `false` (hamis) értéket adja vissza. Keressük a "tt"-t az "ütődött tánc tanár" karakterláncban.

```
print ereg("tt", "ütődött tánc tanár", $tomb);  
print "<br>$tomb[0]<br>";  
// a kimenet:  
// 2  
// tt
```

A "tt" az "ütődött" szóban megtalálható, ezért az `ereg()` 2-t ad vissza, ez az illeszkedő betűk száma. A `$tomb` változó első elemében a megtalált karakterlánc lesz, amit aztán a böngészőben megjelenítünk. Felmerülhet a kérdés, milyen esetben lehet ez hasznos, hiszen előre tudjuk, hogy mi a keresett minta. Nem kell azonban mindig előre meghatározott karaktereket keresnünk. A pontot (.) használhatjuk tetszőleges karakter keresésekor:

```
print ereg("d.", "ütődött tánctanár", $tomb);
print "<br>$tomb[0]<br>";
// a kimenet:
// 2
// dő
```

A d. minta illeszkedik minden olyan szövegre, amely megfelel annak a meghatározásnak, hogy „egy d betű és egy azt követő tetszőleges karakter”. Ebben az esetben nem tudjuk előre, mi lesz a második karakter, így a \$tomb[0] értéke máris értelmet nyer.

Egynél többször előforduló karakter keresése mennyiségjelzővel

Amikor egy karakterláncban egy karaktert keresünk, egy mennyiségjelzővel (kvantorral) határozhatjuk meg, hányszor forduljon elő a karakter a mintában. Az a+ minta például azt jelenti, hogy az "a"-nak legalább egyszer elő kell fordulnia, amit 0 vagy több "a" betű követ. Próbáljuk ki:

```
if ( ereg("a+", "aaaa", $tomb) )
    print $tomb[0];
// azt írja ki, hogy "aaaa";
```

Vegyük észre, hogy ez a szabályos kifejezés annyi karakterre illeszkedik, amennyire csak tud. A 18.1. táblázat az ismétlődő karakterek keresésére szolgáló mennyiségjelzőket ismerteti.

18.1. táblázat Az ismétlődő karakterek mennyiségjelzői

<i>Jel</i>	<i>Leírás</i>	<i>Példa</i>	<i>Erre illeszkedik</i>	<i>Erre nem illeszkedik</i>
*	Nulla vagy több előfordulás	a*	xxxx	(Mindenre illeszkedik)
+	Egy vagy több előfordulás	a+	xaax	xxxx
?	Nulla vagy egy előfordulás	a?	xaxx	xaax
{n}	n előfordulás	a{3}	xaaa	aaaa
{n, }	Legalább n előfordulás	a{3, }	aaaa	aaxx
{, n}	Legfeljebb n előfordulás	a{, 2}	xaax	aaax
{n1, n2}	Legalább n1 és legfeljebb n2 előfordulás	a{1, 2}	xaax	xaaa

A kapcsos zárójelben levő számok neve korlát. Segítségükkel határozhatjuk meg, hányszor ismétlődjön egy adott karakter, hogy a minta érvényes legyen rá.

ÚJDONSÁG

A korlát határozza meg, hogy egy karakternek vagy karakterláncnak hányszor kell ismétlődnie egy szabályos kifejezésben. Az alsó és felső határt a kapcsos zárójelen belül határozzuk meg. Például az

`a{4,5}`

kifejezés az `a` négynél nem kevesebb és ötnél nem több előfordulására illeszkedik.

Vegyünk egy példát. Egy klub tagsági azonosítóval jelöli tagjait. Egy érvényes azonosítóban egy és négy közötti alkalommal fordul elő a `"t"`, ezt tetszőleges számú szám vagy betű karakter követi, majd a `99`-es szám zárja. A klub arra kért fel bennünket, hogy a tennivalókat tartalmazó naplóból emeljük ki a tagazonosítókat.

```
$proba = "A ttXGDH99 tagunk befizette már a tagsági
          díjat?";
if ( ereg( "t{1,4}.*99 ", $proba, $tomb ) )
print "A megtalált azonosító: $tomb[0]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99"
```

Az előző kódrészletben a tagazonosító két `t` karakterrel kezdődik, ezt négy nagybetű követi, majd végül a `99` jön. A `t{1,4}` minta illeszkedik a két `t`-re, a négy nagybetűt pedig megtalálja a `.*`, amellyel tetszőleges számú, tetszőleges típusú karakterláncot kereshetünk.

Úgy tűnik, ezzel megoldottuk a feladatot, pedig még attól meglehetősen távol állunk. A feltételek között a `99`-cel való végződést azzal próbáltuk biztosítani, hogy megadtuk, hogy egy szóköz legyen az utolsó karakter. Ezt aztán találat esetén vissza is kaptuk. Még ennél is nagyobb baj, hogy ha a forráslánc

```
"Azonosítóm ttXGDH99 megkaptad már az 1999 -es tagsági díjat?"
```

akkor a fenti szabályos kifejezés a következő karakterláncot találná meg:

```
"tóm ttXGDH99 megkaptad már az 1999"
```

Mit rontottunk el? A szabályos kifejezés megtalálta a `t` betűt az azonosítóm szóban, majd utána a tetszőleges számú karaktert egészen a szóközzel zárt `99`-ig. Jellemző a szabályos kifejezések „mohóságára”, hogy annyi karakterre illeszkednek, amennyire csak tudnak. Emiatt történt, hogy a minta egészen az `1999`-ig illeszkedett, a `99`-re végződő azonosító helyett. A hibát kijavíthatjuk, ha biztosan tudjuk, hogy a `t` és a `99` közötti karakterek kizárólag betűk vagy számok és egyikük sem szóköz. Az ilyen feladatokat egyszerűsítik le a karakterosztályok.

Karakterlánc keresése karakterosztályokkal

Az eddigi esetekben megadott karakterekkel vagy a `.` segítségével kerestünk karaktereket. A karakterosztályok lehetővé teszik, hogy karakterek meghatározott csoportját keressük. A karakterosztály meghatározásához a keresendő karaktereket szögletes zárójelek közé írjuk. Az `[ab]` minta egyetlen betűre illeszkedik, ami vagy `a` vagy `b`. A karakterosztályokat meghatározásuk után karakterként kezelhetjük, így az `[ab]+` az `aaa`, `bbb` és `ababab` karakterláncokra egyaránt illeszkedik.

Karaktertartományokat is használhatunk karakterosztályok megadására: az `[a-z]` tetszőleges kisbetűre, az `[A-Z]` tetszőleges nagybetűre, a `[0-9]` pedig tetszőleges számjegyre illeszkedik. Ezeket a sorozatokat és az egyedi karaktereket egyetlen karakterosztályba is gyúrhatjuk, így az `[a-z5]` minta tetszőleges kisbetűre vagy az `5`-ös számra illeszkedik.

A karakterosztályokat „tagadhatjuk” is, a csúcsos ékezet (`^`) kezdő szögletes zárójel után való írásával: az `[^A-Z]` mindenre illeszkedik, csak a nagybetűkre nem.

Térjünk vissza legutóbbi példánkra. Mintát kell illesztenünk egy `1` és `4` közötti alkalommal előforduló betűre, amelyet tetszőleges számú betű vagy számjegy karakter követ, amit a `99` zár.

```
$proba = "Azonosítóm ttXGDH99 megkaptad már az 1999 -es
        tagsági díjat?";
if ( ereg( "t{1,4}[a-zA-Z0-9]*99 ", $proba, $tomb ) )
    print "A megtalált azonosító: $tomb[0]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99 "
```

Szép lassan alakul. Az a karakterosztály, amit hozzáadtunk, már nem fog szóközökre illeszkedni, így végre az azonosítót kapjuk meg. Viszont ha az azonosító után a próba-karakterlánc végére vesszőt írunk, a szabályos kifejezés megint nem illeszkedik:

```
$proba = "Azonosítóm ttXGDH99, megkaptad már az 1999 -es
        tagsági díjat?";
if ( ereg( "t{1,4}[a-zA-Z0-9]*99 ", $proba, $tomb ) )
    print "A megtalált azonosító: $tomb[0]";
// a szabályos kifejezés nem illeszkedik
```

Ez azért van, mert a minta végén egy szóközt várunk el, amely alapján meggyőződhetünk, hogy elértük az azonosító végét. Így ha egy szövegben az azonosító zárójelek közt van, kötőjel vagy vessző követi, a minta nem illeszkedik rá. Közéletben visz a megoldáshoz, ha úgy javítunk a szabályos kifejezésen, hogy mindenre illeszkedjék, csak szám és betű karakterekre nem:

```
$proba = "Azonosítóm ttXGDH99, megkaptad már az 1999 -es
        tagsági díjat?";
if ( ereg( "t{1,4}[a-zA-Z0-9]*99[^a-zA-Z0-9]", $proba,
    $tomb ) )
    print "A megtalált azonosító: $tomb[0]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99, "
```

Már majdnem kész is vagyunk, de még mindig van két probléma. Egyrészt a vesszőt is visszakapjuk, másrészt a szabályos kifejezés nem illeszkedik, ha az azonosító a karakterlánc legvégén van, hiszen megköveteltük, hogy utána még egy karakter legyen. Más szóval a szóhatárt kellene megbízható módon megtalálnunk. Erre a problémára később még visszatérünk.

Az atomok kezelése

ÚJDONSÁG

Az atom egy zárójelek közé írt minta (gyakran hivatkoznak rá részmin-taként is). Meghatározása után az atomot ugyanúgy kezelhetjük, mintha maga is egy karakter vagy karakterosztály lenne. Más szóval ugyanazt a 18.1. táblá-zatban bemutatott rendszer alapján felépített mintát annyiszor kereshetjük, ahány-szor csak akarjuk.

A következő kódrészletben meghatározunk egy mintát, zárójelezzük, és az így kapott atomnak kétszer kell illeszkednie a keresendő szövegbe:

```
$proba = "abbaxaabaxabbax";
if ( ereg( "([ab]+x){2}", $proba, $tomb ) )
    print "$tomb[0]";
// azt írja ki, hogy "abbaxaabax"
```

Az `[ab]+x` illeszkedik az "abbax" és az "aabax" karakterláncokra egyaránt, így az `([ab]+x){2}` illeszkedni fog az "abbaxaabax" karakterláncra.

Az `ereg()` függvénynek átadott tömbváltozó első elemében kapjuk vissza a teljes, megtalált, illeszkedő karakterláncot. Az ezt követő elemek az egyes megtalált atomokat tartalmazzák. Ez azt jelenti, hogy a teljes találat mellett a megtalált minta részeihez is hozzáférhetünk.

A következő kódrészletben egy IP címre illesztünk mintát, és nem csupán a teljes címet tároljuk, hanem egyes alkotóelemeit is:

```
$ipcim = "158.152.55.35";
if ( ereg( "([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)", $ipcim,
    $tomb ) )
{
    foreach ( $tomb as $ertek )
        print "$ertek<BR>";
}
// Az eredmény:
// 158.152.55.35
// 158
// 152
// 55
// 35
```

Vegyük észre, hogy a pontokat a szabályos kifejezésben fordított perjel előzte meg. Ezzel fejeztük ki, hogy a . karaktert itt minden különleges tulajdonsága nélkül, egyszerű karakterként szeretnénk kezelni. Minden olyan karakternél ez a követendő eljárás, amely egyedi szereppel bír a szabályos kifejezésekben.

Elágazások

A szűrőkarakter (|) segítségével mintákat összekötve a szabályos kifejezéseken belül elágazásokat hozhatunk létre. Egy kétágú szabályos kifejezés vagy az első mintára, vagy a második mintára illeszkedő karakterláncokat keresi meg. Ettől lesz a szabályos kifejezések nyelvtana még rugalmasabb. A következő kódrészletben a .com, a .de vagy a .hu karakterláncot keressük:

```
$domain = "www.egydomain.hu";
if ( ereg( "\.com|\.de|\.hu", $domain, $tomb ) )
print "ez egy $tomb[0] tartomány<BR>";
// azt írja ki, hogy "ez egy .hu tartomány"
```

A szabályos kifejezés helye

Nemcsak a keresendő mintát adhatjuk meg, hanem azt is, hol illeszkedjen egy karakterláncban. Ha a mintát a karakterlánc kezdetére szeretnénk illeszteni, a szabályos kifejezés elejére írunk egy csúcsos ékezetet (^). A ^a illeszkedik az "alma", de nem illeszkedik a "banán" szóra.

A karakterlánc végén úgy illeszthetjük a mintát, hogy dollárjelet (\$) írunk a szabályos kifejezés végére. Az a\$ illeszkedik a "róka", de nem illeszkedik a "hal" szóra.

A tagazonosítót kereső példa újragondolása

Most már rendelkezésünkre állnak azok az eszközök, melyekkel megoldhatjuk a tagazonosítás problémáját. Emlékeztetőül, a feladat az volt, hogy szövegeket elmezve kigyűjtsük azokat a tagazonosítókat, amelyekben a "t" egy és négy közötti alkalommal fordul elő, ezt tetszőleges számú szám vagy betű karakter követi, amelyet a 99 zár. Jelenlegi problémánk az, hogyan határozzuk meg, hogy illeszkedő mintánknak szóhatárra kell esnie. Nem használhatjuk a szóközt, mivel a szavakat írásjelekkel is el lehet választani. Az sem lehet feltétel, hogy egy nem alfanumerikus karakter alkossa a határt, hiszen a tagazonosító lehet, hogy éppen a szöveg elején vagy végén található.

Most, hogy képesek vagyunk elágazásokat megadni és helyhez kötni a szabályos kifejezéseket, megadhatjuk feltételként, hogy a tagazonosítót vagy egy nem szám vagy betű karakter kövesse, vagy pedig a karakterlánc vége. Ugyanezzel a gondolatmenettel adhatjuk meg a kód elején előforduló szóhatárt is. A zárójelek használatával a tagazonosítót minden központozás és szóköz nélkül kaphatjuk vissza:

```
$proba = "Azonosítóm ttXGDH99, megkaptad már az 1999 -es  
tagsági díjat?";  
if (ereg( "(^|[^a-zA-Z0-9])(t{1,4}[a-zA-Z0-9]*99)([^a-zA-  
Z0-9]|$)", $proba, $tomb ) )  
    print "A megtalált azonosító: $tomb[2]";  
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99"
```

Amint láthatjuk, a szabályos kifejezések elsőre egy kicsit riasztók, de miután kisebb egységekre bontjuk azokat, általában egész könnyen értelmezhetők. Elértük, hogy mintánk egy szóhatárra illeszkedjen (pontosan úgy, ahogy azt a feladat megkövetelte). Mivel nem vagyunk kíváncsiak semmilyen, a mintát megelőző vagy követő szövegre, a minta számunkra hasznos részét zárójelek közé írjuk. Az eredményt a \$tomb tömb harmadik (2-es indexű) elemében találhatjuk meg.



Az `ereg()` megkülönbözteti a kis- és nagybetűket. Ha nem szeretnénk megkülönböztetni őket, használjuk az `eregi()` függvényt, amely egyébként minden más tekintetben megegyezik az `ereg()` függvénnyel.

Minták lecserélése karakterláncokban az `ereg_replace()` függvénnyel

Eddig csupán mintákat kerestünk a karakterláncokban, magát a karakterláncot nem módosítottuk. Az `ereg_replace()` függvény lehetővé teszi, hogy megke-
ressük és lecseréljük az adott szöveg egy mintára illeszkedő részláncát. Bemenete
három karakterlánc: egy szabályos kifejezés, a csereszöveg, amivel felül szeret-
nénk írni a megtalált mintát és a forrásszöveg, amelyben keresünk. Találat esetén
a függvény a megváltoztatott karakterláncot adja vissza, ellenkező esetben az ere-
deti forrásszöveget kapjuk. A következő kódrészletben egy klubtisztviselő nevét
keressük ki, majd felülírjuk utódjának nevével:

```
$proba = "Titkárunk, Vilmos Sarolta, szeretettel  
        üdvözli önt.";  
print ereg_replace ("Vilmos Sarolta", "László István",  
                   $proba);  
// azt írja ki, hogy " Titkárunk, László István,  
                   szeretettel üdvözli önt."
```

Fontos megjegyezni, hogy míg az `ereg()` függvény csak a legelső megtalált min-
tára illeszkedik, az `ereg_replace()` a minta összes előfordulását megtalálja és
lecseréli.

Visszaulalás használata az `ereg_replace()` függvénnyel

A visszaulalások azt teszik lehetővé, hogy az illeszkedő kifejezés egy részét
felhasználhassuk a felülíró karakterláncban. Ehhez szabályos kifejezésünk összes
felhasználható elemét zárójelbe kell írunk. Ezekre a részminták segítségével
megtalált karakterláncokra két fordított perjellel és az atom számával hivatkozha-
tunk (például `\1`) a csereláncokban. Az atomok sorszámozottak, a sorszámozás
kívülről befelé, balról jobbra halad és `\1`-től indul. A `\0` az egész illeszkedő
karakterláncot jelenti.

A következő kódrészlet a `hh/nn/éééé` formátumú dátumokat alakítja
`éééé.hh.nn.` formátumúvá:

```
$datumt = "12/25/2000";  
print ereg_replace("([0-9]+)/([0-9]+)/([0-9]+)",  
                  "\3.\1.\2.", $datumt);  
// azt írja ki, hogy "2000.12.25"
```

Vegyük észre, hogy a fenti kód csereszövegében a pontok nem különleges értel-
műek, így nem is kell `\` jelet tenni eléjük. A `.` az első paraméterben kapja a tetsző-
leges karakterre illeszkedés különleges jelentését.



Az `ereg_replace()` függvény megkülönbözteti a kis- és nagybetűket. Ha nem szeretnénk megkülönböztetni ezeket, használjuk az `eregi_replace()` függvényt, amely minden más tekintetben megegyezik az `ereg_replace()` függvénnyel.

Karakterláncok felbontása a `split()` függvénnyel

Az előző órában már láttuk, hogy az `explode()` függvény segítségével egy karakterláncot elemeire bontva tömbként kezelhetünk. Az említett függvény hatékonyságát az csökkenti, hogy határolójelként csak egy karakterhalmazt használhatunk. A PHP 4 `split()` függvénye a szabályos kifejezések hatékonyságát biztosítja, vele rugalmas határolójelet adhatunk meg. A függvény bemenete a határolójelként használt minta és a forráslánc, amelyben a határolókat keressük, kimenete pedig egy tömb. A függvény harmadik, nem kötelező paramétere a visszaadandó elemek legnagyobb számát határozza meg.

A következő példában egy elágazó szabályos kifejezéssel egy karakterláncot bontunk fel, úgy, hogy a határolójel egy vessző és egy szóköz vagy egy szóközzel közrefogott és szó:

```
$szoveg = "almák, narancsok, körték és barackok";  
$gyumolcsok = split(", | és ", $szoveg);  
foreach ( $gyumolcsok as $gyumolcs )  
    print "$gyumolcs<BR>";  
// a kimenet:  
// almák  
// narancsok  
// körték  
// barackok
```

Perl típusú szabályos kifejezések

Ha Perlös háttérrel közelítjük meg a PHP-t, akkor a POSIX szabályos kifejezések függvényeit valamelyest nehézkesnek találhatjuk. A jó hír viszont az, hogy a PHP 4 támogatja a Perlnek megfelelő szabályos kifejezéseket is. Ezek formája még az eddig tanultaknál is hatékonyabb. Ebben a részben a két szabályoskifejezés-forma közötti különbségekkel ismerkedünk meg.

Minták keresése a `preg_match()` függvénnyel

A `preg_match()` függvény három paramétert vár: egy szabályos kifejezést, a forrásláncot és egy tömbváltozót, amelyben az illeszkedő karakterláncokat adja vissza. A `true` (igaz) értéket kapjuk vissza, ha illeszkedő kifejezést talál és `false` (hamis) értéket, ha nem. A `preg_match()` és az `ereg_match()` függvények közötti különbség a szabályos kifejezést tartalmazó paraméterben van. A Perl típusú szabályos kifejezéseket határolójelek közé kell helyezni. Általában perjeleket használunk határolójelként, bár más, nem szám vagy betű karaktert is használhatunk (kivétel persze a fordított perjel). A következő példában a `preg_match()` függvénnyel egy h-valami-z mintájú karakterláncot keresünk:

```
$szoveg = "üveg ház";  
if ( preg_match( "/h.*z/", $szoveg, $tomb ) )  
    print $tomb[0];  
// azt írja ki, hogy "ház"
```

A Perl típusú szabályos kifejezések és a mohóság

Alapértelmezés szerint a szabályos kifejezések megkísérelnek a lehető legtöbb karakterre illeszkedni. Így a

```
"/h.*z/"
```

minta a h-val kezdődő és z karakterre végződő lehető legtöbb karaktert közrefogó karakterláncra fog illeszkedni, a következő példában a teljes szövegre:

```
$szoveg = "ház húz hülyéz hazardíroz";  
if ( preg_match( "/h.*z/", $szoveg, $tomb ) )  
    print $tomb[0];  
// azt írja ki, hogy "ház húz hülyéz hazardíroz"
```

Viszont ha kérdőjelet (?) írunk a mennyiséget kifejező jel után, rávehetjük a Perl típusú szabályos kifejezést, hogy egy kicsit mértékletesebb legyen. Így míg a

```
"h.*z"
```

minta azt jelenti, hogy „h és z között a lehető legtöbb karakter”, a

```
"h.*?z"
```

minta jelentése a „h és z közötti lehető legkevesebb karakter”.

A következő kódrészlet ezzel a módszerrel keresi meg a legrövidebb szót, amely h-val kezdődik és z-vel végződik:

```
$szoveg = "ház húz hülyéz hazardíroz";
if ( preg_match( "/h.*?z/", $szoveg, $tomb ) )
    print $tomb[0];
// azt írja ki, hogy "ház"
```

A Perl típusú szabályos kifejezések és a fordított perjeles karakterek

A Perl típusú szabályos kifejezésekben is vezérlőkarakterré változtathatunk bizonyos karaktereket, ugyanúgy, mint ahogy azt karakterláncokkal tettük. A `\t` például a tabulátor karaktert jelenti, az `\n` pedig a soremelést. Az ilyen típusú szabályos kifejezések olyan karaktereket is leírhatnak, amelyek teljes karakterhalmazokra, karaktertípusokra illeszkednek. A fordított perjeles karakterek listáját a 18.2. táblázatban láthatjuk.

18.2. táblázat Karaktertípusokra illeszkedő beépített karakterosztályok

<i>Karakter</i>	<i>Illeszkedik</i>
<code>\d</code>	Bármely számra
<code>\D</code>	Mindenre, ami nem szám
<code>\s</code>	Bármely elválasztó karakterre
<code>\S</code>	Mindenre, ami nem elválasztó karakter
<code>\w</code>	Bármely szóalkotó karakterre (aláhúzást is beleértve)
<code>\W</code>	Mindenre, ami nem szóalkotó karakter

Ezek a karakterosztályok nagymértékben leegyszerűsíthetik a szabályos kifejezéseket. Nélkülük karaktersorozatok keresésekor saját karakterosztályokat kellene használnunk. Vessük össze a szóalkotó karakterekre illeszkedő mintát az `ereg()` és a `preg_match()` függvényben:

```
ereg( "h[a-zA-Z0-9_]+z", $szoveg, $tomb );
preg_match( "/h\w+z", $szoveg, $tomb );
```

A Perl típusú szabályos kifejezések több váltókaraktert is támogatnak, melyek hivatkozási pontként működnek. A hivatkozási pontok a karakterláncon belül helyekre illeszkednek, nem pedig karakterekre. Ismertetésüket a 18.3. táblázatban találhatjuk.

18.3. táblázat Hivatkozási pontként használt váltókarakterek

Karakter	Illeszkedik
\A	Karakterlánc kezdetére
\b	Szóhatárra
\B	Mindenre, ami nem szóhatár
\Z	Karakterlánc végére (az utolsó soremelés vagy a karakterlánc vége előtt illeszkedik)
\z	Karakterlánc végére (a karakterlánc legvégére illeszkedik)

18

Emlékszünk még, milyen problémáink voltak a szóhatárra való illesztéssel a tagsági azonosítást megvalósító példában? A Perl típusú szabályos kifejezések jelentősen leegyszerűsítik ezt a feladatot. Vessük össze, hogyan illeszt mintát szóalkotó karakterre és szóhatárra az `ereg()` és a `preg_match()`:

```
ereg ( " (^| [^a-zA-Z0-9_]) (y{1,4} [a-zA-Z0-9_]*99)
    ➡ ([^a-zA-Z0-9_] | $)", $szoveg, $tomb );
preg_match( "\bt{1,4}\w*99\b", $szoveg, $tomb );
```

Az előző példában a `preg_match()` függvény meghívásakor a szóhatáron levő legalább egy, de legfeljebb négy `t` karakterre illeszkedik, amelyet bármennyi szövegalkotó karakter követhet és amit a szóhatáron levő 99 karaktorsor zár. A szóhatárt jelölő váltókarakter igazából nem is egy karakterre illeszkedik, csupán megerősíti, hogy az illeszkedéshez szóhatár kell. Az `ereg_match()` meghívásához először létre kell hoznunk egy nem szóalkotó karakterekből álló mintát, majd vagy arra, vagy pedig szóhatárra kell illeszteni.

A váltókaraktereket arra is használhatjuk, hogy megszabaduljunk a karakterek jelentésétől. Ahhoz például, hogy egy `.` karakterre illeszthessünk, egy fordított perjelet kell elé írunk.

Teljeskörű keresés a `preg_match_all()` függvénnyel

A POSIX szabályos kifejezésekkel az az egyik probléma, hogy a minta karakterláncon belüli összes előfordulását bonyolult megkeresni. Így ha az `ereg()` függvénnyel keressük a `b`-vel kezdődő, `t`-vel végződő szavakat, csak a legelső találatot kapjuk vissza. Próbáljuk meg magunk is:

```

$szoveg = "barackot, banánt, bort, búzát és
           békákat árulok";
if ( ereg( "(^|^[a-zA-Z0-9_])(b[a-zA-Z0-9_]+t)
    ➡ ([a-zA-Z0-9_]|$)", $szoveg, $tomb ) )
{
    for ( $x=0; $x< count( $tomb ); $x++ )
        print "\$tomb[$x]: $tomb[$x]<br>\n";
}
// kimenete:
// $tomb[0]: barackot,
// $tomb[1]:
// $tomb[2]: barackot
// $tomb[3]: ,

```

Ahogy azt vártuk, a `$tomb` harmadik elemében találjuk az első találatot, a "barackot". A tömb első eleme tartalmazza a teljes találatot, a második a szóközt, a negyedik a vesszőt. Hogy megkapjuk az összes illeszkedő kifejezést a szövegben, az `ereg_replace()` függvényt kellene használnunk, ciklikusan, hogy eltávolítsuk a találatokat a szövegből, mielőtt újra illesztenénk a mintát.

Fel kell, hogy hívjuk a kedves olvasó figyelmét arra, hogy a magyar szövegekre az ékezetes betűk miatt alapbeállításban nem alkalmazható a fentihez hasonló egyszerű szabályos kifejezés. A nemzeti beállítások testreszabására használható `setlocale()` függvényt kell alkalmaznunk, hogy a kívánt eredményt elérjük.

A `preg_match_all()` függvényt használhatjuk arra, hogy egyetlen hívásból a minta összes előfordulását megkapjuk. A `preg_match_all()` bemenete egy szabályos kifejezés, a forráslánc és egy tömbváltozó. Találat esetén `true` (igaz) értéket ad vissza. A tömbváltozó egy többdimenziós tömb lesz, melynek első eleme a szabályos kifejezésben megadott mintára illeszkedő összes találatot tartalmazza.

A 18.1. programban a `preg_match_all()` függvénnyel illesztettük mintánkat egy szövegre. Két `for` ciklust használunk, hogy megjelenítsük a többdimenziós eredménytömböt.

18.1. program Minta teljeskörű illesztése a `preg_match_all()` függvénnyel

```

1: <html>
2: <head>
3: <title>18.1. program Minta teljeskörű illesztése
   a preg_match_all() függvénnyel</title>
4: </head>
5: <body>

```

18.1. program (folytatás)

```

6: <?php
7: $szoveg = "barackot, banánt, bort, búzát és
   békákat árulok";
8: if ( preg_match_all( "/\bb\w+t\b/", $szoveg, $tomb ) )
9:     {
10:        for ( $x=0; $x< count( $tomb ); $x++ )
11:            {
12:                for ( $y=0; $y< count( $tomb[$x] ); $y++ )
13:                    print "\$tomb[$x][$y]:
                        ".$tomb[$x][$y]."<br>\n";
14:            }
15:        }
16: // A kimenet:
17: // $tomb[0][0]: barackot
18: // $tomb[0][1]: banánt
19: // $tomb[0][2]: bort
20: // $tomb[0][3]: búzát
21: // $tomb[0][4]: békákat
22: ?>
23: </body>
24: </html>

```

A `$tomb` változónak a `preg_match_all()` függvénynek történő átadáskor csak az első eleme létezik és ez karakterláncok egy tömbjéből áll. Ez a tömb tartalmazza a szöveg minden olyan szavát, amely `b`-vel kezdődik és `t` betűre végződik.

A `preg_match_all()` ebből a tömbből azért készít egy többdimenziósat, hogy az atomok találatait is tárolhassa. A `preg_match_all()` függvénynek átadott tömb első eleme tartalmazza a teljes szabályos kifejezés minden egyes találatát. Minden további elem a szabályos kifejezés egyes atomjainak (zárójeles részmintáinak) értékeit tartalmazza az egyes találatokban. Így a `preg_match_all()` alábbi meghívásával

```

$szoveg = "01-05-99, 01-10-99, 01-03-00";
preg_match_all( "/(\d+)-(\d+)-(\d+)/", $szoveg, $tomb );
a $tomb[0] az összes találat tömbje:
$tomb[0][0]: 01-05-99
$tomb[0][1]: 01-10-99
$tomb[0][2]: 01-03-00
A $tomb[1] az első részminta értékeinek tömbje:
$tomb[1][0]: 01

```

```
$tomb[1][1]: 01
$tomb[1][2]: 01
A $tomb[2] a második részminta értékeinek tömbjét tárolja:
$tomb[2][0]: 05
$tomb[2][1]: 10
$tomb[2][2]: 03
```

és így tovább.

Minták lecserélése a `preg_replace()` függvénnyel

A `preg_replace()` használata megegyezik az `ereg_replace()` használatával, azzal a különbséggel, hogy hozzáférünk a Perl típusú szabályos kifejezések által kínált kényelmi lehetőségekhez. A `preg_replace()` bemenete egy szabályos kifejezés, egy csere-karakterlánc és egy forráslánc. Találat esetén a módosított karakterláncot, ellenkező esetben magát a forrásláncot kapjuk vissza változtatás nélkül. A következő kódrészlet a `nn/hh/éé` formátumú dátumokat alakítja át `éé/hh/nn` formátumúvá:

```
$szoveg = "25/12/99, 14/5/00";
$szoveg = preg_replace( "\b(\d+)/(\d+)/(\d+)\b|",
    ➡ "\\3/\\2/\\1", $szoveg );
print "$szoveg<br>";
// azt írja ki, hogy "99/12/25, 00/5/14"
```

Vegyük észre, hogy a szűrőkaraktert `()` használtuk határolójelként. Ezzel megtakarítottuk a perjelek kikerülésére használandó váltókaraktereket az illesztendő mintában. A `preg_replace()` ugyanúgy támogatja a visszautalásokat a csere-szövegben, mint az `ereg_replace()`.

A `preg_replace()` függvénynek a forráslánc helyett karakterláncok egy tömbjét is átadhatjuk, az ebben az esetben minden tömbelemet egyesével átalakít. Ilyenkor a visszaadott érték az átalakított karakterláncok tömbje lesz.

Emellett a `preg_replace()` függvénynek szabályos kifejezések és csere-karakterláncok tömbjét is átadhatjuk. A szabályos kifejezéseket egyenként illeszti a forrásláncra és a megfelelő csereszöveggel helyettesíti. A következő példában az előző példa formátumait alakítjuk át, de emellett a forráslánc szerzői jogi (copyright) információját is módosítjuk:

```
$szoveg = "25/12/99, 14/5/00. Copyright 1999";
$मित = array( "\b(\d+)/(\d+)/(\d+)\b|",
    ➡ "/([Cc]opyright) 1999/" );
$मित = array( "\\3/\\2/\\1", "\\1 2000" );
```



```
$szoveg = preg_replace( $miket, $mikre, $szoveg );
print "$szoveg<br>";
// azt írja ki, hogy "99/12/25, 00/5/14. Copyright 2000"
```

Két tömböt hozunk létre. A `$miket` tömb két szabályos kifejezést, a `$mikre` tömb pedig csere-karakterláncokat tartalmaz. A `$miket` tömb első eleme a `$mikre` tömb első elemével helyettesítődik, a második a másodikkal, és így tovább.

Ha a csere-karakterláncok tömbje kevesebb elemet tartalmaz, mint a szabályos kifejezéseké, a csere-karakterlánc nélkül maradó szabályos kifejezéseket a függvény üres karakterláncra cseréli.

Ha a `preg_replace()` függvénynek szabályos kifejezések egy tömbjét adjuk át, de csak egy csereláncot, akkor az a szabályos kifejezések tömbjének összes mintáját ugyanazzal a csereszöveggel helyettesíti.

Módosító paraméterek

A Perl típusú szabályos kifejezések lehetővé teszik, hogy módosító paraméterek segítségével pontosítsuk a minta illesztési módját.

ÚJDONSÁG

A módosító paraméter olyan betű, amelyet a Perl típusú szabályos kifejezések utolsó határolójele után írunk, és amely tovább finomítja szabályos kifejezéseink jelentését.

A Perl típusú szabályos kifejezések módosító paramétereit a 18.4. táblázatban találhatjuk.

18.4. táblázat A Perl típusú szabályos kifejezések módosító paramétereit

Minta	Leírás
i	Nem különbözteti meg a kis- és nagybetűket.
e	A <code>preg_replace()</code> csereláncát PHP-kódként kezeli.
m	A <code>\$</code> és a <code>^</code> az aktuális sor elejére és végére illeszkedik.
s	A soremelésre is illeszkedik (a soremelések általában nem illeszkednek a <code>.</code> -ra).
x	A karakterosztályokon kívüli elválasztó karakterekre az olvashatóság érdekében nem illeszkedik. Ha elválasztó karakterekre szeretnénk mintát illeszteni, használjuk a <code>\s</code> , <code>\t</code> , vagy a <code>\</code> (fordított perjel–szóköz) mintákat.

18.4. táblázat. (folytatás)

<i>Minta</i>	<i>Leírás</i>
A	Csak a karakterlánc elején illeszkedik (ilyen megszorítás nincs a Perlben).
E	Csak a karakterlánc végén illeszkedik (ilyen megszorítás nincs a Perlben).
U	Kikapcsolja a szabályos kifejezések „mohóságát”, azaz a lehető legkevesebb karaktert tartalmazó találatokat adja vissza (ez a módosító sem található meg Perlben).

Ahol ezek a megszorítások nem mondanak egymásnak ellent, egyszerre többet is használhatunk belőlük. Az `x` megszorítást például arra használhatjuk, hogy könnyebb legyen olvasni a szabályos kifejezéseket, az `i` megszorítást pedig arra, hogy mintánk kis- és nagybetűktől függetlenül illeszkedjék a szövegre.

A `/k\S*r/ix` kifejezés például illeszkedik a `"kar"` és `"KAR"` szavakra, de a `"K A R"` szóra már nem. Az `x`-szel módosított szabályos kifejezések váltókarakter nélküli szóközei nem fognak illeszkedni semmire sem a forrásláncban, a különbség csupán esztétikai lesz.

Az `m` paraméter akkor jöhet jól, ha egy szöveg több sorában szeretnénk hivatkozási pontos mintára illeszteni. A `^` és `$` minták alapértelmezés szerint a teljes karakterlánc elejét és végét jelzik. A következő kódrészletben az `m` paraméterrel módosítjuk a `$^` viselkedését:

```
$szoveg = "név: péter\nfoglalkozás: programozó\nszeme: kék\n";
preg_match_all( "/^\w+:\s+(.*)$/m", $szoveg, $tomb );
foreach ( $tomb[1] as $ertek )
    print "$ertek<br>";
// kimenet:
// péter
// programozó
// kék
```

Egy olyan szabályos kifejezést hozunk létre, amely olyan mintára illeszkedik, amelyben bármely szóalkotó karaktert vessző és tetszőleges számú szóköz követ. Ezután tetszőleges számú karaktert követő „karakterlánc vége” karakterre (`$`) illesztünk. Az `m` paraméter miatt a `$` az egyes sorok végére illeszkedik a teljes karakterlánc vége helyett.

Az `s` paraméter akkor jön jól, ha a `.` karakterrel szeretnénk többsoros karakterláncban karakterekre illeszteni. A következő példában megpróbálunk a karakterlánc első és utolsó szavára keresni:

```
$szoveg = "kezdetként indítsunk ezzel a sorral\nés így
    ➡ egy következtetéshez\nérkezhetünk el végül\n";
preg_match( "/^(\w+).*?(\w+)$/", $szoveg, $tomb );
print "$tomb[1] $tomb[2]<br>";
```

Ez a kód semmit nem ír ki. Bár a szabályos kifejezés megtalálja a szóalkotó karaktereket a karakterlánc kezdeténél, a `.` nem illeszkedik a szövegben található soremelésre. Az `s` paraméter ezen változtat:

```
$szoveg = "kezdetként indítsunk ezzel a sorral\nés így
    ➡ egy következtetéshez\nérkezhetünk el végül\n";
preg_match( "/^(\w+).*?(\w+)$/s", $szoveg, $tomb );
print "$tomb[1] $tomb[2]<br>";
// azt írja ki, hogy "kezdetként végül"
```

Az `e` paraméter különlegesen hasznos lehet számunkra, hiszen lehetővé teszi, hogy a `preg_replace()` függvény csereszövegét PHP kódként kezeljük. A függvényeknek paraméterként visszaulásokat vagy számokból álló listákat adhatunk át. A következő példában az `e` paraméterrel adunk át illesztett számokat egy függvénynek, amely ugyanazt a dátumot más formában adja vissza.

```
<?php
function datumAtalakito( $honap, $nap, $ev )
{
    $ev = ($ev < 70 )?$ev+2000:$ev;
    $ido = ( mktime( 0,0,0,$honap,$nap,$ev ) );
    return date("Y. m. j. ", $ido);
}

$datumok = "3/18/99<br>\n7/22/00";
$datumok = preg_replace( "/([0-9]+)\./([0-9]+)\./([0-9]+)/e",
    "datumAtalakito(\1,\2,\3)",
    $datumok);

print $datumok;

// ezt írja ki:
// 1999. 03. 18.
// 2000. 07. 22.
?>
```

Három, perjelekkel elválasztott számhalmazt keresünk és a zárójelek használatával rögzítjük a megtalált számokat. Mivel az `e` módosító paramétert használjuk, a cserelánc paraméterből meghívhatjuk az általunk meghatározott `datumAtalakito()` függvényt, úgy, hogy a három visszaulást adjuk át neki. A `datumAtalakito()` csupán fogja a számbemenetet és egy jobban festő dátummá alakítja, amellyel aztán kicseréli az eredetit.

Összefoglalás

A szabályos kifejezések hatalmas témakört alkotnak, mi csak felületesen ismerkedtünk meg velük ezen óra során. Arra viszont már biztosan képesek leszünk, hogy bonyolult karakterláncokat találjunk meg és cseréljünk le egy szövegben.

Megismerkedtünk az `ereg()` szabályoskifejezés-függvénnyel, amellyel karakterláncokban mintákat kereshetünk, illetve az `ereg_replace()`-szel, amellyel egy minta összes előfordulását cserélhetjük le. A karakterosztályok használatával karakterláncokat, a mennyiségjelzők használatával több mintát, az elágazásokkal mintákat vagylagosan kereshetünk. Részmintákat is kigyűjthetünk és rájuk visszaulásokkal hivatkozhatunk. A Perl típusú szabályos kifejezésekben váltókarakterekkel hivatkozási pontos illesztést kérhetünk, de karakterosztályok illesztésére is rendelkezésre állnak váltókarakterek. Ezenkívül a módosító paraméterekkel képesek vagyunk finomítani a Perl típusú szabályos kifejezések viselkedésén.

Kérdések és válaszok

A Perl típusú szabályos kifejezések nagyon hatékonnak tűnnek. Hol található róluk bővebb információ?

A szabályos kifejezésekről a PHP weboldalon (<http://www.php.net>) találhatunk némi felvilágosítást. Emellett még a <http://www.perl.com> oldalain találhatunk információt, a Perl típusú szabályos kifejezésekhez pedig a <http://www.perl.com/pub/doc/manual/html/pod/perlre.htm> címen és Tom Christiansen cikkében, a <http://www.perl.com/pub/doc/manual/html/pod/perlfaq6.html> címen találhatunk bevezetést.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

18

1. Melyik függvényt használnánk egy karakterláncban minta illesztésére, ha POSIX szabályos kifejezéssel szeretnénk a mintát meghatározni?
2. Milyen szabályos kifejezést használnánk, ha a "b" betű legalább egyszeri, de hatnál nem többszöri előfordulására szeretnénk keresni?
3. Hogyan határoznánk meg a "d" és az "f" betűk közé eső karakterek karakterosztályát?
4. Hogyan tagadnánk a 3. kérdésben meghatározott karaktertartományt?
5. Milyen formában keresnénk tetszőleges számra vagy a "bokor" szóra?
6. Melyik POSIX szabályos kifejezés függvényt használnánk egy megtalált minta lecserélésére?
7. A `.bc*` szabályos kifejezés mohón illeszkedik, azaz inkább az `"abc0000000bc"` karakterláncra illeszkedik, mint az `"abc"`-re. A Perl típusú szabályos kifejezésekkel hogyan alakítanánk át a fenti szabályos kifejezést úgy, hogy a megtalált minta legelső előfordulását keresse meg?
8. A Perl típusú szabályos kifejezéseknél melyik fordított perjeles karakterrel illesztünk elválasztó karakterre?
9. Melyik Perl típusú szabályos kifejezést használnánk, ha az a célunk, hogy egy minta összes előfordulását megtaláljuk?
10. Melyik módosító karaktert használnánk egy Perl típusú szabályoskifejezés-függvényben, ha kis- és nagybetűtől függetlenül szeretnénk egy mintára illeszteni?

Feladatok

1. Gyűjtsük ki az e-mail címeket egy fájlból. A címeket tároljuk egy tömbben és jelenítsük meg az eredményt a böngészőben. Nagy mennyiségű adattal finomítsuk szabályos kifejezéseinket.



19. ÓRA

Állapotok tárolása sütitkel és GET típusú lekérdezésekkel

A HTTP állapot nélküli protokoll. Ez azt jelenti, hogy ahány oldalt a felhasználó letölt a kiszolgálónkról, annyi önálló kapcsolat létesül, ráadásul az egyes oldalakon belül a képek, külső fájlok letöltésére is külön kapcsolatok nyílnak. Másrészt viszont a felhasználók és a weboldalak készítői környezetként, azaz olyan térként érzékelik a weboldalakat, amelyben egyetlen oldal is egy nagyobb egész része. Így viszont nem meglepő, hogy az oldalról oldalra való információátadás módszerei egyidősek magával a Világhálóval.

Ezen az órán az információtárolás azon két módszerével fogunk megismerkedni, amelyek lehetővé teszik, hogy egy adott oldalon található adat a következő oldalakon is elérhető legyen. Az óra során a következőket tanuljuk meg:

- Mik a süti és hogyan működnek?
- Hogyan olvassuk a sütiiket?

- Hogyan írjunk sütiket?
- Hogyan tároljuk adatbázisban a weboldalak lekérési statisztikáját sütik segítségével?
- Mik azok a GET módú lekérések?
- Hogyan írjunk olyan függvényt, amely egy asszociatív tömböt egy lekérdező karakterlánccá alakít át?

Sütik

A Netscape munkatársai által kiötölt „csodasüti” (magic cookie) kifejezés még a Netscape 1 aranykorából származik. A kifejezés pontos eredete mindmáig viták tárgya, bár meglehetősen valószínűnek tűnik, hogy a szerencsesütinek valami köze lehetett hozzá. Azóta viszont már más böngészők is alkalmazzák ezt a szabványt.

ÚJDONSÁG

A süti (cookie) kis mennyiségű adat, amelyet a felhasználó böngészője tárol egy kiszolgáló vagy egy program kérésének eleget téve. Egy gép legfeljebb 20 sütit tárolhat a felhasználó böngészőjében. Minden süti egy nevet, egy értéket, a lejáratási időpontot és a gazdagépre és elérési útra vonatkozó információkat tartalmazza. Az egyes sütik mérete legfeljebb 4 KB lehet.

A süti értékeinek beállítása után csak az a számítógép olvashatja az adatokat, amely azokat létrehozta, ezzel is garantálva a felhasználó biztonságát. A felhasználó azt is beállíthatja böngészőjében, hogy az értesítse az egyes sütik létrehozásának kérelméről, vagy elutasíthatja az összes sütikérelmet. Ezekből adódóan a sütiket módjával kell használni, nem lehet teljesen rájuk támaszkodni a környezet megtervezésekor anélkül, hogy a felhasználót először ne értesítenénk.

Mindezekkel együtt a sütik kiválóan alkalmasak arra, hogy kis mennyiségű információt tároljunk velük, mialatt a felhasználó oldalról-oldalra lépdel, vagy akár hosszabb távon is, a webhely egyes látogatásai között.

A sütik felépítése

A sütik létrehozása általában a HTTP fejlécében történik (bár a JavaScript közvetlenül a böngészővel is létre tudja hozni a sütiket). Egy sütibeállító PHP program ilyesféle HTTP fejlécelemet kell, hogy létrehozzon:

```
Set-Cookie: zoldseg=articsoka; expires=Friday,  
➡ 04-Feb-00 22:03:38 GMT; path=/; domain=kiskapu.hu
```


Mint ahogy azt láthatjuk, a Set-Cookie fejléc egy név-érték párt, egy greenwichi középidő szerinti lejáratási időpontot (expires), egy elérési utat (path) és egy tartományt (domain) tartalmaz. A név és az érték URL felépítésű kell, hogy legyen. A lejárat mező (expires) arra ad utasítást a böngészőnek, hogy mikor „felejtse el” a sütit. Az elérési út a weboldalon azt a helyet jelöli, amelynek elérésekor a sütit vissza kell küldeni a kiszolgálóra. A tartomány mező azokat az internetes tartományokat jelöli ki, ahová a sütit el kell küldeni. A tartomány nem különbözhet attól, ahonnan a sütit küldték, viszont egy bizonyos szintű rugalmasságot is meghatározhat. Az előző példánál a böngésző a `www.kiskapu.hu` és a `leeloo.kiskapu.hu` kiszolgálóknak egyaránt hajlandó elküldeni a sütit. A HTTP fejlécéről a tizenharmadik óra anyagában többet is olvashattunk.

Ha a böngésző úgy van beállítva, hogy tárolja a sütiket, akkor az azokban tárolt adatok a süti lejártáig elérhetők maradnak. Ha a felhasználó a böngészővel olyan oldalra jut, amely egyezik a sütiben tárolt elérési útvonallal és tartománnyal, akkor elküldi a sütit a kiszolgálónak. Ehhez általában az alábbihoz hasonló fejlécelemet állít elő:

```
Cookie: zoldseg=articsoka
```

Így aztán a PHP program hozzáférhet a sütihez a `HTTP_COOKIE` környezeti változón keresztül (amely az összes süti nevét és értékét tárolja), de a `$zoldseg` globális változón keresztül vagy a `$HTTP_COOKIE_VARS["zoldseg"]` globális tömbváltozó segítségével is elérhetjük azt:

```
print "$HTTP_COOKIE<BR>"; // azt írja ki, hogy
    ➤ "zoldseg=articsoka"
print getenv("HTTP_COOKIE")."<br>"; // azt írja ki, hogy
    ➤ "zoldseg=articsoka"
print "$zoldseg<br>"; // azt írja ki, hogy "articsoka"
print "$HTTP_COOKIE_VARS["zoldseg"]<br>"; // azt írja ki, hogy
    ➤ "articsoka"
```

Sütik beállítása a PHP-vel

A PHP-vel kétféleképpen hozhatunk létre egy sütit. Egyrészt a kilencedik óra során megismert `header()` függvény segítségével beállíthatjuk a fejléc Set-Cookie sorát. A `header()` egy karakterláncot vár, amit az oldal HTTP fejlécébe ír. Mivel a fejlécek automatikusan kerülnek kiírásra, a `header()` függvényt még azelőtt kell meghívunk, mielőtt bármi egyebet küldenénk a böngészőnek. Figyeljünk arra, hogy a PHP blokk kezdőeleme elé írt egyetlen szóköz vagy újsor karakter is kime-natként jelenik meg a böngészőben.

```
header ("Set-Cookie: zoldseg=articsoka; expires=Friday,  
    ➡ 04-Feb-00 22:03:38 GMT; path=/; domain=kiskapu.hu");
```

Bár ez a módszer nem túl bonyolult, mégis írunk kell egy olyan függvényt, amely előállítja a fejléc szövegét. Ez persze nem túl rázós feladat, hiszen csak a megfelelő formátumú dátumot, illetve az URL formátumú név-érték párt kell előállítanunk. Kár azonban ezzel vesződnünk, mert létezik egy olyan PHP függvény, amely pontosan ezt végzi el helyettünk.

A `setcookie()` függvény neve önmagáért beszél: a fejléc `Set-Cookie` sorát állítja elő. Ennek megfelelően azelőtt kell még meghívunk, mielőtt valamit kiküldենék a böngészőnek. A függvény bemenetét a süti neve, értéke, UNIX időbélyeg formátumban megadott lejárat, elérési útja, a feljogosított tartomány, valamint egy egész szám alkotja. Ezt a legutolsó paramétert 1-re állítva biztosíthatjuk, hogy a süti csak biztonságos kapcsolaton keresztül tudjon közlekedni. A süti nevén kívül egyik paraméter sem kötelező.

A 19.1. programban arra láthatunk példát, ahogy a `setcookie()` függvény segítségével beállítunk egy sütit.

19.1. program Süti értékének beállítása és kiírása

```
1: <?php  
2: setcookie( "zoldseg", "articsoka", time()+3600, "/",  
3:           "kiskapu.hu", 0 );  
4: ?>  
5: <html>  
6: <head>  
7: <title>19.1. program Süti értékének beállítása és  
   kiírása</title>  
8: </head>  
9: <body>  
10: <?php  
11: if ( isset( $zoldseg ) )  
12: print "<p>Üdv, az ön által kiválasztott zöldség a(z)  
    $zoldseg</p>";  
13: else  
14: print "<p>Üdvözljük! Ez az ön első látogatása.</p>";  
15: ?>  
16: </body>  
17: </html>
```

Bár a program első lefuttatásakor már beállítjuk a sütit, a `$zoldseg` változó ekkor még nem jön létre. A süti csak akkor lesz hozzáférhető, amikor a böngésző elküldi azt a kiszolgálónak. Ez csak akkor következik be, ha a felhasználó újra meglátogatja tartományunkat. A példában egy "zoldseg" nevű sütit hozunk létre, melynek értéke "articsoka". A `time()` függvénnyel lekérdezzük a pillanatnyi időt, majd ebből 3600-at hozzáadva számítjuk ki a lejárat idejét. (A 3600 másodpercben értendő, tehát a süti 1 óra elteltével fog elavulni.) Útvonalként a "/"-t adjuk meg, amiből az következik, hogy sütink a webhely összes oldaláról elérhető lesz. A tartományt "kiskapu.hu"-ra állítottuk, aminek következtében ennek a tartománynak mindegyik kiszolgálója jogosult a süti fogadására (a `www.kiskapu.hu` tehát ugyanúgy megkapja a sütit, mint a `leeloo.kiskapu.hu`). Ha azt szeretnénk, hogy egy sütihez csak az a kiszolgáló férjen hozzá, amelyik a sütit létrehozó programot szolgáltatta, használjuk a `$SERVER_NAME` környezeti változót, ahelyett, hogy a tartomány, illetve kiszolgálónevet beépítenénk a programba. Ez a megoldás azzal az előnnyel is rendelkezik, hogy a programok módosítás nélkül átvihetők egy másik kiszolgálóra és ott is az elvárásoknak megfelelően fognak futni. Végül egy nullát is átadunk a `setcookie()`-nak, jelezvén, hogy a sütit nem biztonságos kapcsolaton keresztül is el szabad küldeni. Bár az első kivételével mindegyik paraméter elhagyható, hasznos, ha minden paramétert megadunk és csak a tartomány és biztonság adatok meghatározásától tekintünk el. Erre azért van szükség, mert bizonyos böngészők csak az útvonal megadása esetén képesek a sütik rendeltetészerű használatára. Az útvonal elhagyásával a süti használatát az aktuális, illetve az alatta elhelyezkedő könyvtárak oldalai számára korlátozzuk.

Ha a `setcookie()` szöveges paramétereiként ""-t, azaz üres karakterláncot adunk át, a szám paramétereinek pedig 0-t, a függvény ezeket a paramétereket nem veszi figyelembe.

Süti törlése

A süti törlésének „hivatalos” módja az, hogy a `setcookie()` függvényt egyetlen paraméterrel, mégpedig a törlendő süti nevével hívjuk meg:

```
setcookie( "zoldseg" );
```

Ebben a megoldásban nem bízhatunk meg teljesen, mert nem minden esetben működik kifogástalanul. Ezért célszerűbb, ha egy múltbeli dátummal új értéket adunk a sütinek:

```
setcookie( "zoldseg", "", time()-60, "/", "kiskapu.hu", 0 );
```

Ennek alkalmazásakor viszont ügyelnünk kell, hogy a megadott útvonal, tartomány és biztonsági paraméter tökéletesen egyezzen a süti létrehozásakor megadottakkal.

Munkamenet-azonosító sütik

Ha olyan sütit szeretnénk létrehozni, amely csak addig létezik, amíg a felhasználó be nem zárja a böngészőt, csak annyi a teendőnk, hogy a süti élettartamát nullára állítjuk. Amíg ki nem lépünk a böngészőből, az így készített sütit a kiszolgáló gond nélkül megkapja, ám ha bezárjuk a programot, a sütik megszűnnek, így hiába indítjuk újra a böngészőt, már nem érjük el azokat.

Ezen eljárás legfőbb alkalmazási területe a felhasználók azonosítása. A felhasználó elküldi a nevét és jelszavát, válaszként pedig egy olyan oldalt kap, amely egy sütit hoz létre a böngészőben. Ezek után a böngésző a sütit minden személyes adatokat tartalmazó laphoz történő hozzáféréskor visszaküldi a kiszolgálónak, az pedig ennek hatására engedélyezi a hozzáférést. Ilyen esetekben nem kívánatos, hogy a böngésző újraindítása után is hozzáférhessünk ezekhez a lapokhoz, mert nem lehetünk biztosak benne, hogy nem egy másik felhasználó indította el a böngészőt. A munkamenet-azonosítót hagyományosan *sessionid*-nek nevezik, a böngésző bezárásáig élő sütit pedig *session cookie*-nak.

```
setcookie( "session_id", "55435", 0 );
```

Példa: Webhelyhasználat nyomon követése

Képzeljük el, hogy egy tartalomszolgáltató azzal bízott meg bennünket, hogy sütik, valamint egy MySQL adatbázis segítségével kimutatást készítsünk a látogatókról. Szükség van a látogatók számára, a látogatások alkalmával letöltött lapok számának átlagára és az olvasók által a webhelyen töltött átlagidőre, látogatókra lebontva.

Az első dolgunk, hogy megértessük megbízónkkal a sütik alkalmazásának korlátait. Nem mindegyik felhasználó böngészőjében engedélyezett a sütik használata. Ilyen esetben a lekért oldal mindig úgy fog tűnni, mintha ez lenne az adott felhasználó első látogatása. Így hát a kapott eredményeket kissé meghamisítják azok a böngészők, amelyek nem tudják vagy nem akarják kezelni a sütiket. Továbbá abban sem lehetünk biztosak, hogy egy felhasználó mindig ugyanazt a böngészőt használja, és abban sem, hogy egy adott böngészőn nem osztozik-e esetleg több ember.

Ha megbízónk mindezt tudomásul vette, nekiveselkedhetünk a tényleges megvalósításnak. Egy működő mintát kevesebb, mint 90 sorból összerakhatunk!

Szükségünk lesz először is egy adatbázistáblára, mely a 19.1-es táblázatban felsorolt mezőkből áll.

19.1. táblázat Adatbázismezők

<i>Név</i>	<i>Típus</i>	<i>Leírás</i>
vendeg_azon	egész	Automatikusan növekvő mező, amely minden látogató számára egyedi azonosítót állít elő és tárol.
elso_latogatas	egész	A látogató első látogatásának időbélyege.
utolso_latogatas	egész	A legutóbb lekért lap időpontjának időbélyege.
latogatas_szam	egész	Az elkülöníthető látogatások száma.
ossz_ido	egész	A webhelyen töltött idő közelítő értéke.
ossz_letoltes	egész	A látogató összes letöltési kérelmeinek száma.

A vendeg_naplo MySQL tábla létrehozásához a következő CREATE utasításra lesz szükség:

```
create table vendeg_naplo (  
    vendeg_azon INT NOT NULL AUTO_INCREMENT,  
    PRIMARY KEY( vendeg_azon ),  
    elso_latogatas INT,  
    utolso_latogatas INT,  
    latogatas_szam INT,  
    ossz_ido INT,  
    ossz_letoltes INT  
);
```

Most, hogy elkészítettük a táblát, amivel dolgozhatunk, írunk kell egy programot, amely megnyit egy adatbázis-kapcsolatot és képes ellenőrizni a süti jelenlétét. Ha a süti nem létezik, új sort hoz létre a táblában és feltölti az új látogató adataival. Ennek megvalósítása látható a 19.2. példában.

19.2 program A MySQL adatbázist új felhasználó adataival bővítő program

```
1: <?php
2: $adatbazis = mysql_connect( "localhost", "janos",
    "majomkenyerfa" );
3: if ( ! $adatbazis )
4:     die( "Nem lehet a mysql-hez kapcsolódni!" );
5: mysql_select_db( "minta", $adatbazis ) or die (
    mysql_error() );
6: if ( ! isset ( $vendeg_azon ) )
7:     $vendeg_allapot = ujvendeg( $adatbazis );
8: else
9:     print "Örülünk, hogy ismét körünkben
        üdvözölhetjük, $vendeg_azon<P>";
10: function ujvendeg( $adatbazis )
11: {
12:     // egy új vendég!
13:     $vendeg_adatok = array (
14:         "elso_latogatas" => time(),
15:         "utolso_latogatas" => time(),
16:         "latogatas_szam" => 1,
17:         "ossz_ido" => 0,
18:         "ossz_letoltes" => 1
19:     );
20:     $lekerdezes = "INSERT INTO vendeg_naplo
        ( elso_latogatas,
21:         utolso_latogatas,
22:         latogatas_szam,
23:         ossz_ido,
24:         ossz_letoltes ) ";
25:     $lekerdezes .= "values ( ".
        $vendeg_adatok["elso_latogatas"].", ".
26:         $vendeg_adatok["utolso_latogatas"].", ".
27:         $vendeg_adatok["latogatas_szam"].", ".
28:         $vendeg_adatok["ossz_ido"].", ".
29:         $vendeg_adatok["ossz_letoltes"]." )";
30:     $eredmeny = mysql_query( $lekerdezes );
31:     $vendeg_adatok["vendeg_azon"] =
        mysql_insert_id();
32:     setcookie( "vendeg_azon",
        $vendeg_adatok["vendeg_azon"],
33:         time()+(60*60*24*365*10), "/" );
34:     return $vendeg_adatok;
35: }
36: ?>
```

A szokványos módon csatlakozunk a MySQL kiszolgálóhoz és kiválasztjuk azt az adatbázist, amelyben a `vendeg_naplo` tábla található (a MySQL adatbázis-kiszolgáló kezelését a tizenkettedik óra tartalmazza). Ellenőrizzük a `$vendeg_azon` változó jelenlétét, amely a felhasználót azonosító egyedi egész számot tartalmazza. Ha ez a változó nem létezik, feltételezzük, hogy új felhasználóról van szó és bejegyzéséhez meghívjuk az `ujvendeg()` nevű függvényt.

Az `ujvendeg()` függvény egy hivatkozás-azonosítót kér és egy tömböt ad vissza. A függvényben létrehozunk egy `$vendeg_adatok` nevű tömböt. A tömb `első_látogatás` és `utolsó_látogatás` elemeit a pillanatnyi időre állítjuk. Mivel ez az első látogatás, így a `látogatás_száma` és az `összes_látogatás` elemeket 1-re állítjuk. Az `összes_idő` 0 lesz, hiszen ezen látogatás alkalmával még nem telt el idő.

A létrehozott tömb alapján elkészítjük a táblába illesztendő új sort, úgy, hogy a tömb mindegyik elemét a tábla azonos nevű oszlopának adjuk értékül. Mivel a `vendeg_azon` mező automatikusan növekszik, megadásától eltekinthetünk. A süti beállításakor azonban szükség lesz erre az újonnan előállított azonosítóra. Ezt válaszként a `mysql_insert_id()` függvénytől kaphatjuk meg. Most, hogy elláttuk az új látogatót egy azonosítóval, nincs más hátra, mint kibővíteni a tömböt az azonosítóval, amely így már megfelel az adatbázisrekordnak, amit az imént létrehoztunk.

Végző lépésként létrehozuk a `vendeg_azon` nevű sütit egy `setcookie()` hívással és visszaadjuk a `$vendeg_adatok` tömböt a hívó programnak.

Legközelebb, amikor a látogató működésbe hozza a programot, a PHP már látni fogja a `$vendeg_azon` változón keresztül a `vendeg_azon` sütit. A program ezt úgy értékeli, hogy egy régi ismerős visszatéréséről van szó és ennek megfelelően frissíti a `vendeg_naplo` táblát, majd üdvözlí a felhasználót.

A frissítés elvégzéséhez meg kell még vizsgálnunk, hogy a program futását egy folyamatban lévő böngészés eredményezte vagy új látogatás első lépéséről van szó. Ennek eldöntésében egy globális változó lesz segítségünkre, mely egy másodpercben mért időtartamot tárol. Ha a program által érzékelt legutolsó letöltés ideje az előbb említett időtartamnál régebbi, akkor a mostani letöltés egy újabb látogatás kezdetének tekintendő, ellenkező esetben a folyamatban lévő látogatás részének tekintjük.

A 19.3. példa a `regivendeg()` függvénnyel bővíti a 19.2. programot.

19.3 program Felhasználókövető program sütik és MySQL adatbázis felhasználásával

```
1: <?php
2: $lhossz = 300; // a látogatás hossza 5 perc,
                   másodpercben kifejezve
3: $adatbazis = mysql_connect( "localhost", "janos",
                               "majomkenyerfa" );
4: if ( ! $adatbazis )
5:     die( "Nem lehet a mysqld-hez kapcsolódni!" );
6: mysql_select_db( "minta", $adatbazis ) or
   die ( mysql_error() );
7: if ( ! isset ( $vendeg_azon ) )
8:     $vendeg_allapot = ujvendeg( $adatbazis );
9: else
10: {
11:     $vendeg_allapot = regivendeg( $adatbazis,
                                     $vendeg_azon, $lhossz );
12:     print "Örülünk, hogy ismét körünkben
           üdvözölhetjük, $vendeg_azon<P>";
13: }
14: function ujvendeg( $adatbazis )
15: {
16:     // egy új vendég!
17:     $vendeg_adatok = array (
18:         "elso_latogatas" => time(),
19:         "utolso_latogatas" => time(),
20:         "latogatas_szam" => 1,
21:         "ossz_ido" => 0,
22:         "ossz_letoltes" => 1
23:     );
24:     $lekerdezes = "INSERT INTO vendeg_naplo
           ( elso_latogatas,
25:             utolso_latogatas,
26:             latogatas_szam,
27:             ossz_ido,
28:             ossz_letoltes ) ";
29:     $lekerdezes .= "values ( ".
           $vendeg_adatok["elso_latogatas"].", ".
30:           $vendeg_adatok["utolso_latogatas"].", ".
31:           $vendeg_adatok["latogatas_szam"].", ".
32:           $vendeg_adatok["ossz_ido"].", ".
33:           $vendeg_adatok["ossz_letoltes"]." )";
34:     $eredmeny = mysql_query( $lekerdezes );
35:     $vendeg_adatok["vendeg_azon"] =
           mysql_insert_id();
```


19.3 program folytatás

```
36:         setcookie( "vendeg_azon",
37:                     $vendeg_adatok["vendeg_azon"],
38:                     time()+(60*60*24*365*10), "/" );
39:     }
40: function regivendeg( $adatbazis, $vendeg_azon, $lhossz )
41: {
42:     // egy ismerős vendég,
43:     // aki már járt nálunk korábban!
44:     $lekerdezes = "SELECT * FROM vendeg_naplo WHERE
45:                   vendeg_azon=$vendeg_azon";
46:     $eredmeny = mysql_query( $lekerdezes );
47:     if ( ! mysql_num_rows( $eredmeny ) )
48:     {
49:         // süti van ugyan, de azonosító nincs – tehát
50:         // mégiscsak új vendég
51:         return ujvendeg( $adatbazis );
52:     }
53:     $vendeg_adatok = mysql_fetch_array( $eredmeny,
54:                                         $adatbazis );
55:     // ez most egy újabb letöltés, tehát növeljük
56:     // a számlálót
57:     $vendeg_adatok["ossz_letoltes"]++;
58:     if ( ( $vendeg_adatok["utolso_latogatas"]
59:           + $lhossz ) > time() )
60:     {
61:         // még mindig ugyanaz a látogatás,
62:         // tehát növeljük az összidőt.
63:         $vendeg_adatok["ossz_ido"] +=
64:             ( time() - $vendeg_adatok["utolso_latogatas"] );
65:     }
66:     else
67:     {
68:         // ez már egy új látogatás,
69:         $vendeg_adatok["latogatas_szam"]++;
70:         // ennek megfelelően módosítjuk az adatbázist
71:         $lekerdezes = "UPDATE vendeg_naplo SET
72:                       utolso_latogatas=".time().",
73:                       latogatas_szam=".$vendeg_adatok
74:                       ["latogatas_szam"].", ".
75:                       "ossz_ido=".$vendeg_adatok["ossz_ido"].", ".
76:                       "ossz_letoltes=" .
77:                       $vendeg_adatok["ossz_letoltes"]." ";
78:         $lekerdezes .= "WHERE vendeg_azon=$vendeg_azon";
79:         $eredmeny = mysql_query( $lekerdezes );
80:         return $vendeg_adatok;
81:     }
82: }
```

```
67: ?>
```

Látható, hogy a programot az `$lhossz` nevű változóval bővítettük. Ebben adjuk meg azt az időtartamot, amely alapján megítéljük, hogy a letöltés mely látogatáshoz tartozik. A `$vendeg_azon` változó jelenlétének érzékelése azt jelenti számunkra, hogy kaptunk egy sütit. Erre válaszul meghívjuk a `regivendeg()` függvényt, átadva annak az adatbázisra hivatkozó, a `$vendeg_azon` és az `$lhossz` változót, amit 300 másodpercre, azaz 5 percre állítottunk.

A `regivendeg()` függvény első lépésként lekérdezi a felhasználóról tárolt adatokat. Ezt egyszerűen úgy tesszük, hogy kikérjük a `vendeg_naplo` tábla azon sorát, melyben a `vendeg_azon` mező egyenlő a `$vendeg_azon` változó értékével. A `mysql_query()` által visszaadott sorok számát megtudhatjuk, ha meghívjuk a `mysql_num_rows()` függvényt az eredmény sorok azonosítójával. Ha a `mysql_num_rows()` válasza 0, akkor mégis új felhasználóról van szó, tehát meg kell hívunk az `ujvendeg()` függvényt.

Tegyük fel, hogy találtunk egy megfelelő sort, melynek azonosítója egyezik a süti értékével. Ezt a sort a `mysql_fetch_array()` függvénnyel emelhetjük át egy PHP tömbbe (esetünkben a `$vendeg_adatok` nevűbe). A program mostani futása egy oldal letöltésének következménye, tehát a `$vendeg_adatok["ossz_letoltes"]` változó értékét eggyel növelnünk kell, hogy rögzítsük ezt a ténytet.

Megvizsgáljuk, hogy a `$vendeg_adatok["utolso_latogatas"]` és az `$lhossz` összege a pillanatnyi időnél későbbi időpont-e. Ha későbbi, akkor az azt jelenti, hogy a legutóbbi letöltés óta kevesebb, mint `$lhossz` idő telt el, vagyis a legutóbbi látogatás még folyamatban van és ez a letöltés még annak a része. Ennek tényét úgy őrizzük meg, hogy a legutóbbi letöltés óta eltelt időt hozzáadjuk a `$vendeg_adatok["ossz_ido"]` változóhoz.

Ha úgy találtuk, hogy már új látogatásról van szó, egyszerűen növeljük a `$vendeg_adatok["latogatas_szam"]` változó értékét.

Befejezésül a `$vendeg_adatok` tömb tartalmával frissítjük a `vendeg_naplo` táblát és a hívó program is megkapja ezt a tömböt. Bár ezt külön nem emeltük ki, természetesen az `utolso_latogatas` mezőt is a pillanatnyi időhöz igazítottuk.

Most, hogy a feladaton túl vagyunk, írhatunk egy kis programot, hogy ellenőrizzük, az elmélet valóban működik-e a gyakorlatban. Elkészítjük az `allapotMegjelenites()` függvényt, amely kiszámolja a lekérő felhasználó átlagértékeit és megjeleníti azokat a böngészőben. A valóságban persze komolyabb megjelenésű, átfogóbb képet kell adnunk a látogatókról, de a lényeg megértéséhez ez a példa is bőven elegendő. A függvény megvalósítását a 19.4. példában találjuk. A korábbi példák kódjához az `include()` függvény alkalmazásával férhetünk hozzá.

19.4 program A 19.3-as program által begyűjtött letöltési statisztikák megjelenítése

```
1: <?php
2: include("19.3.program.php");
3: állapotMegjelenites();
4: function állapotMegjelenites()
5:     {
6:         global $vendeg_allapot;
7:         $letoltesek = sprintf( "%.2f",
8:                                ($vendeg_allapot["ossz_letoltes"]/
8:                                /$vendeg_allapot["latogatas_szam"])
9:                                );
10:        $idotartam = sprintf( "%.2f",
10:                               ($vendeg_allapot["ossz_ido"]/
10:                               /$vendeg_allapot["latogatas_szam"]) );
11:        print "<p>Üdvözljük! Az Ön azonosítója".
11:            $vendeg_allapot["vendeg_azon"]."</p>\n\n";
12:        print "<p>Az Ön látogatásainak száma
12:            $vendeg_allapot["latogatas_szam"]."</p>\n\n";
13:        print "<p>Letöltések átlagos száma
13:            látogatásonként: $letoltesek</p>\n\n";
14:        print "<p>Átlagos látogatási időtartam: $idotartam
14:            másodperc</p>\n\n";
15:        print "<p>Átlagos látogatási időtartam: $idotartam
15:            másodperc</p>\n\n";
16:    }
17: ?>
```

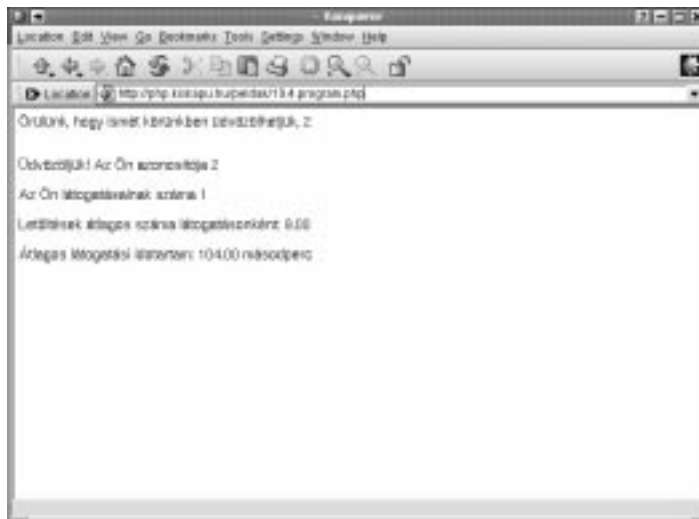
A 19.1 ábra a 19.4 program kimenetét mutatja. Az `include()` sorral biztosítottuk, hogy a felhasználókövető kód lefusson. Hasonló sorral kell bővítenünk webhelyünk minden olyan oldalát, melyet be szeretnénk vonni a statisztikába. Az `allapotMegjelenites()` függvény a `$vendeg_allapot` globális változó alapján dolgozik. Ezt vagy az `ujvendeg()`, vagy a `regivendeg()` függvény hozza létre és tölti fel a `vendeg_naplo` tábla megfelelő sorának értékeivel.

Azt, hogy egy felhasználó látogatásonként átlagosan hányszor kattintott olyan hivatkozásra, mely a webhely valamelyik oldalára mutatott, úgy számoljuk ki, hogy elosztjuk a `$vendeg_allapot["ossz_letoltes"]` változót a látogatósok számával. Ugyanezzel az értékkel a `$vendeg_allapot["ossz_ido"]`-t elosztva a látogatósok átlagidejét kaphatjuk meg. Ezeket a számokat az `sprintf()` segédletével kerekítjük két tizedesjegyre, majd mondatokká formálva megjelenítjük azokat a felhasználónak.

Természetesen a példát úgy is bővíthetnénk, hogy a program rögzítse a felhasználó érdeklődési területeit a webhelyen belül, de akár naplóztathatnánk a látogatáshoz használt böngésző típusát vagy a látogatók IP címeit is. Hogy mire lehet mind ezt használni? Könnyíthetjük felhasználóink eligazodását webhelyünkön, úgy, hogy böngészési szokásaikat kielemezve, vastag betűvel kiemeljük a számukra feltehetőleg különösképp érdekes mondanivalót.

19.1. kép

A használati statisztika megjelenítése



Lekérdező karakterláncok használata

A süti hatalmas hátránya azok felhasználó-függősége. Nem csak a felhasználó kénye-kedvének vagyunk kitéve, ha sütitet használunk, de a felhasználó böngészőjétől is függ használatuk. A felhasználó letilthatja a süti használatát, a böngészők pedig nem mindig egyformán valósítják meg a szabványt. Egyikük-másikuk a süti kezelése területén dokumentált hibákkal is rendelkezik. Ha csak egy látogatás folyamán szeretnénk állapot-adatokat tárolni, jobban járunk, ha egy sokkal hagyományosabb megoldáshoz folyamodunk.

Ha egy űrlapot a GET eljárással küldünk el, akkor annak mezőit és azok értékeit URL kódolásban hozzáillesztjük ahhoz a címhez (URL-hez), ahová az űrlapot küldjük. Az értékek ekkor elérhetőek lesznek a kiszolgáló, illetve az általa futtatott programok számára. Vegyünk példaként egy olyan űrlapot, amely egy felhasználó és egy nev mezőt tartalmaz. Ekkor a lekérés a következőképp fog festeni:

```
http://php.kiskapu.hu/proba5.php?nev=
➡ 344343&felhasznalo=Szy+Gyorgy
```

Minden mező nevét egy egyenlőségjel (=) választja el annak értékétől; ezeket a név-érték párokat pedig ÉS jelek (&) határolják. A PHP visszafejti a jeleket, a talált adatokat a `$HTTP_GET_VARS` asszociatív tömbben teszi elérhetővé a program számára, és emellett a mezők nevével azonos nevű globális változókat is létrehoz, a megfelelő tartalommal. A `felhasznalo` nevű GET változóhoz így az alábbi két módon férhetünk hozzá:

```
$HTTP_GET_VARS["felhasznalo"];  
$felhasznalo;
```

A GET lekérések szerencsére nem csak űrlapokkal kapcsolatban használhatók; viszonylag könnyen készíthetünk mi is ilyen karakterláncokat, így a fontos adatokat könnyedén továbbíthatjuk lapról lapra.

19

Lekérdező karakterlánc készítése

Alapvető feladatunk a lekérdező karakterláncok készítésekor, hogy a kulcs-érték párokat URL formára alakítsuk. Tegyük fel, hogy egy címet (URL-t) szeretnénk átadni egy lapnak paraméterként, azaz egy lekérdező karakterlánc részeként. Erre a PHP `urlencode()` függvénye használható, mely egy tetszés szerinti karakterláncot vár és annak kódolt változatával tér vissza:

```
print urlencode("http://www.kiskapu.hu");  
// azt írja ki, hogy http%3A%2F%2Fwww.kiskapu.hu
```

Az URL kódolású szövegek előállítása így nem okoz gondot, akár saját GET lekéréseket is összerakhatunk. A következő kódrészlet két változóból épít fel egy lekérdező karakterláncot:

```
<?php  
$erdeklodes = "sport";  
$honlap = "http://www.kiskapu.hu";  
$lekerdezes = "honlap=".urlencode( $honlap );  
$lekerdezes .= "&erdeklodes=".urlencode( $erdeklodes );  
?>  
<A HREF="masiklap.php?<?print $lekerdezes ?>">Gyerünk!</A>
```

Ennek hatására a böngészőhöz az URL már a kódolt lekérdező karakterlánccal bővítve jut el:

```
masiklap.php?honlap=http%3A%2F%2Fwww.kiskapu.hu&  
  ➡ erdeklodes=sport
```

A honlap és az erdeklodes paraméterek a masiklap.php programon belül ugyanilyen nevű, URL-ből visszafejtett globális változókként lesznek elérhetők.

Ez így a problémának kissé kezdetleges megoldása, mert a változónevek kódba ágyazásával megnehezítjük a kód újrahasonosítását. Ahhoz, hogy hatékonyan küldhessünk adatokat lapról-lapra, egyszerűvé kell tennünk a lekérdező karakterláncokat tartalmazó hivatkozások készítését. Ez különösen fontos, ha meg szeretnénk tartani a PHP azon hasznos tulajdonságát, miszerint a nem kifejezetten programozók is könnyedén eligazodhatnak rajta.

A 19.5. példa egy olyan `lsztring()` nevű függvény megvalósítását tartalmazza, amely a kapott asszociatív tömb alapján elkészít és visszaad egy lekérdező karakterláncot.

19.5. program Lekérdező karakterláncot készítő függvény

```
1: <html>
2: <head>
3: <title>19.5 program Lekérdező karakterláncot készítő
   függvény</title>
4: </head>
5: <body>
6: <?php
7: function lsztring( $lekerdezes )
8:     {
9:         global $QUERY_STRING;
10:         if ( ! $lekerdezes ) return $QUERY_STRING;
11:         $lsztring = "";
12:         foreach( $lekerdezes as $kulcs => $ertek )
13:             {
14:                 if ( strlen( $lsztring ) ) $lsztring .= "&";
15:                 $lsztring .= urlencode( $kulcs ) .
16:                     "=" . urlencode( $ertek );
17:             }
18:         return $lsztring;
19:     }
20: $lekerdezes = array ( "nev" => "Kis Lajos Bence",
21:                       "erdeklodes" => "Filmek (főleg
22:                                   művészfilmek)",
23:                       "honlap" => "http://www.kiskapu.hu/lajos/"
24: );
25: print lsztring( $lekerdezes );
```

19.5. program (folytatás)

```
24: // azt írja ki, hogy nev=Kis+Lajos+Bence&erdeklodes=
    Filmek+%28f%F5leg+m%FBv%E9szfilmek
25: // %29&honlap=http%3A%2F%2Fwww.kiskapu.hu%2F1ajos%2F
26: ?>
27: <p>
28: <a href="masiklap.php?<? print lsztring($lekerdezes)
    ?>">Gyerünk!</a>
29: </p>
30: </body>
31: </html>
```

Az `lsztring()` egy asszociatív tömböt vár paraméterként, amit jelen esetben a `$lekerdezes` változó képében adunk át neki. Ha a `$lekerdezes` még nem kapott volna értéket, a programot tartalmazó lap lekérdező karakterláncát adjuk vissza, amit a `$QUERY_STRING` változóval érhetünk el. Ezzel oldhatjuk meg azt, hogy a GET-tel továbbított, módosíthatatlan úrlapadatok könnyedén továbbadhatóak legyenek más lapok számára.

Ha a `$lekerdezes` tömb nem üres, a visszaadandó lekérdező karakterláncot az `$lsztring` változóba helyezzük.

A `foreach` segítségével sorra vesszük a tömb elemeit, úgy, hogy azok a `$kulcs` és `$ertek` változókon keresztül legyenek elérhetők.

A kulcs–érték párokat `&` jellel kell elválasztanunk, így ha nem a ciklus első lefutásánál tartunk – azaz az `$lsztring` változó már nem üres –, ezt a karaktert is hozzátesszük a lekérdező karakterlánchoz.

A `$kulcs` és `$ertek` változókat az `urlencode()` függvénnyel kódolva és egy egyenlőségjellel (`=`) elválasztva adjuk hozzá az `$lsztring` végéhez.

Ezek után már csak vissza kell adnunk az összeállított, kódolt karakterláncot.

E függvény alkalmazásával pár sornyi PHP kód is elég, hogy adatokat adhassunk át egyes lapjaink között.

Összefoglalás

Ezen az órán a weblapok közti információátadás két módjával ismerkedhettünk meg. Segítségükkel több oldalból álló webes alkalmazások alakíthatók ki, melyek a felhasználók igényeit is figyelembe veszik.

Tudjuk, hogyan használjuk a `setcookie()` függvényt süti létrehozására. Ennek kapcsán láthattuk, hogyan tárolhatók adatok a webhely felhasználóinak szokásairól sütik és egy adatbázis felhasználásával. Láttunk lekérdező karakterláncokat is. Tudjuk, hogyan kell azokat kódolni és egy hasznos eszközt is a magunkénak mondhatunk, amely jelentősen egyszerűsíti ezek használatát.

Kérdések és válaszok

Van-e valamiféle komoly biztonságot vagy személyiségi jogokat sértő velejárója a süti használatának?

A kiszolgáló csak a saját tartományán belül levő gépek sütijeit férhet hozzá. Azon túl, hogy a süti a felhasználó fájlrendszerében tárolódik, azzal semmiféle további kapcsolatba nem kerül. Lehetőség van azonban arra, hogy sütit állítsunk be egy kép letöltésének válaszként. Tehát ha sok webhely jelentet meg egy külső reklámszolgáltató vagy számláló programot oldalain, ezek szolgáltatója képes lehet a látogatók követésére különböző webhelyek között is.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Milyen függvénnyel vehetjük rá a weboldalunkat letöltő felhasználó böngészőjét, hogy létrehozzon egy sütit és tárolja azt?
2. Hogyan törölhetjük a sütiket?
3. Milyen függvénnyel tehetünk egy tetszőleges karakterláncot alkalmassá arra, hogy lekérdező karakterláncban szerepelhessen?
4. Melyik beépített változó tartalmazza a lekérdező karakterláncot nyers formában?
5. A lekérdező karakterlánc formájában elküldött név–érték párok globális változókként válnak elérhetővé. Létezik azonban egy tömb is, amely tartalmazza őket. Mi ennek a tömbnek a neve?

Feladatok

1. Tegyük lehetővé, hogy a webhelyünket meglátogató felhasználó beállíthassa a weboldalak háttérszínét, valamint hogy megadhassa, milyen néven köszöntsék a webhely oldalai. Oldjuk meg sütikkel, hogy minden oldal köszönjön, illetve a megfelelő háttérszínnel jelenjen meg.
2. Írjuk át az előbbi feladat programját úgy, hogy az adatokat lekérdező karakterláncban tároljuk, ne sütikben.



20. ÓRA

Állapotok tárolása munkamenet-függvényekkel

Az előző órában áttekintettük, hogy hogyan lehet süti vagy lekérdező karaktersorozat használatával menteni az oldalak állapotát. A PHP 4 ezeken kívül további lehetőségeket is nyújt, különböző függvényei vannak a felhasználó állapotának mentésére is. Ezen függvények használata nagyon hasonló az előző órában tanultakhoz, de mivel közvetlenül a nyelvbe vannak építve, az állapot mentése egyszerű függvényhívásokkal végezhető el.

Ebben az órában a következőkről tanulunk:

- Mik azok a munkamenet-változók és hogyan használhatók?
- Hogyan indítsunk el és folytassunk egy munkamenetet?
- Hogyan tartsuk nyilván a munkamenetek változóit?
- Hogyan semmisítsünk meg egy munkamenetet?
- Hogyan töröljük egy munkamenet egyes változóit?

Mik azok a munkamenet-függvények?

A munkamenet-függvényekkel egyszerűen végezhetjük el azokat a műveleteket, amelyekről az előző órában tanultunk. Ezek egy különleges felhasználói azonosítót biztosítanak, amellyel oldalról oldalra különböző információkat vihetünk át, hozzákapcsolva azokat ehhez az azonosítóhoz. Az előzőekhez képest igen nagy előny, hogy a függvények használatával kevesebb munka hárul ránk. Amikor a felhasználó lekér valamilyen munkamenetet támogató oldalt, kap egy azonosítót, vagy a régebbi látogatásakor szerzett meglévő azonosítója kerül felhasználásra. Ezt követően a munkamenethez kapcsolt összes globális változó elérhető lesz a kódban.

A munkamenet-függvények mindkét, az előző órában tanult módszert támogatják. Alapértelmezés szerint a sütiket használják, ugyanakkor az azonosítókat minden, a saját oldalunkra mutató kereszthivatkozásban elhelyezhetjük, így biztosítva, hogy az oldal a sütiket nem támogató böngészővel is használható legyen.

A munkamenet állapotát a rendszer általában egy ideiglenes fájlban tárolja, de a megfelelő modulok használatával az adatokat adatbázisba is tehetjük.

Munkamenet indítása a `session_start()` függvénnyel

Hacsak nem változtattuk meg a `php.ini` fájlt, a munkameneteket mindig saját magunknak kell elindítanunk, alapállapotban azok nem indulnak el automatikusan. A `php.ini` fájlban található a következő sor:

```
session.auto_start = 0
```

Változtassuk a `session.auto_start` értékét 1-re, így a PHP dokumentumokban a munkamenetek rögtön elindulnak. Ha nem változtatjuk meg az értéket, mindig meg kell hívunk a `session_start()` függvényt.

Miután a munkamenet elindult, a `session_id()` függvénnyel rögtön hozzáférhetünk a felhasználó munkamenet-azonosítójához. A `session_id()` függvénnyel lekérdezhetjük vagy módosíthatjuk az azonosítót. A 20.1. példában található program a munkamenet-azonosítót írja ki a böngészőbe.

20.1. program Munkamenet indítása vagy folytatása

```
1: <?php
2: session_start();
3: ?>
4: <html>
5: <head>
6: <title>20.1. program Munkamenet indítása vagy
   folytatása</title>
7: </head>
8: <body>
9: <?php
10: print "<p>Üdvözzöljük! Az Ön munkamenet-azonosítója "
    .session_id()."</p>\n\n";
11: ?>
12: </body>
13: </html>
```

A fenti program első lefutása után létrehoz egy munkamenet-azonosítót. Ha a felhasználó később újratölti az oldalt, ugyanazt az azonosítót kapja meg. Ehhez természetesen szükséges, hogy a felhasználó engedélyezze a sütitet a böngészőjében. Vizsgáljuk meg a 20.1. példa kimenetének fejléceit. A süti a következőkben lett beállítva:

```
HTTP/1.1 200 OK
Date: Sun, 07 Jan 2001 13:50:36 GMT
Server: Apache/1.3.9 (Unix) PHP/4.0.3
Set-cookie: PHPSESSID=2638864e9216fee10fcb8a61db382909; path=/
Connection: close
Content-Type: text/html
```

Mivel a `session_start()` megpróbálja beállítani a sütit, amikor munkamenetet hoz létre, el kell indítani, mielőtt bármit kiíratnánk a böngészővel. Vegyük észre, hogy nincs lejáratási idő. Ez azt jelenti, hogy a munkamenet csak addig tarthat, amíg a böngésző aktív. Amikor a felhasználó leállítja a böngészőt, a sütit nem menti, de ez a viselkedés a `php.ini` fájl `session.cookie_lifetime` beállításának megváltoztatásával módosítható. Az alapértelmezett beállítás 0, itt kell a lejáratási időt másodpercben megadni. Ezzel mindegyik munkamenetnek megadjuk a működési határidőt.

Munkamenet-változók

A munkamenet-azonosító hozzárendelése a felhasználóhoz csak az első lépés. Egy munkamenet során tetszőleges számú globális változót vezethetünk be, amelyeket később bármelyik, a munkameneteket támogató oldalunkról elérhetünk.

A változó bejegyzéséhez, bevezetéséhez a `session_register()` függvényt kell használnunk. A `session_register()` függvény paramétereiben karaktersorozatokot kell megadni, így határozva meg egy vagy több bevezetendő változó nevét. A függvény visszatérési értéke `true`, ha a bejegyzés sikeres volt. A függvény paramétereiben csak a változók nevét kell megadni, nem pedig a változók értékeit.

A 20.2. példában két változó bejegyzését láthatjuk.

20.2. program Változók bejegyzése

```
1: <?php
2: session_start();
3: ?>
4: <html>
5: <head>
6: <title>20.2. program Változók bejegyzése</title>
7: </head>
8: <body>
9: <?php
10: session_register( "termek1" );
11: session_register( "termek2" );
12: $termek1 = "Ultrahangos csavarhúzó";
13: $termek2 = "HAL 2000";
14: print session_encode();
15: print "A változókat bejegyeztük";
16: ?>
17: </body>
18: </html>
```

A 20.2. példában látható kódnak nincs túl sok értelme, amíg a felhasználó be nem tölt egy új oldalt. A 20.3. példában egy olyan PHP program látható, amely a 20.2. példában bejegyzett változókat használja.

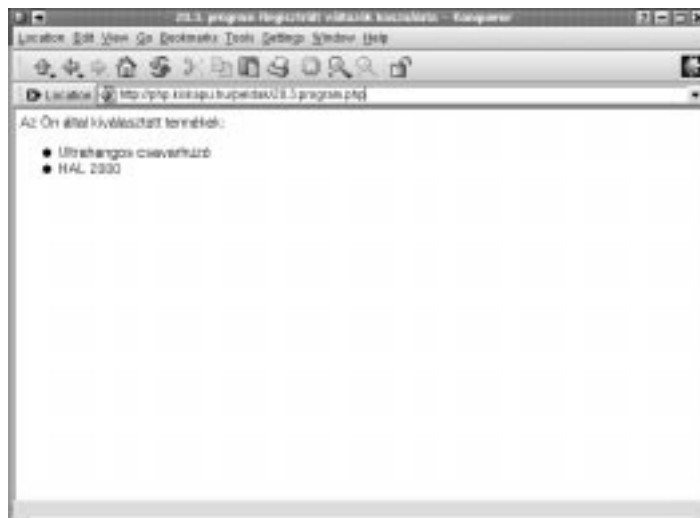
20.3. program Bejegyzett változók használata

```
1: <?php
2: session_start();
3: ?>
4: <html>
5: <head>
6: <title>20.3. program Bejegyzett változók
   használata</title>
7: </head>
8: <body>
9: <?php
10: print "Az Ön által kiválasztott termékek:\n\n";
11: print "<ul><li>$termek1\n<li>$termek2\n</ul>\n";
12: ?>
13: </body>
14: </html>
```

A 20.3. program eredménye a 20.1. ábrán látható. Látjuk, hogy az oldalról hozzáférünk a másik programban bejegyzett `$termek1` és `$termek2` változókhoz.

20.1. ábra

*Bejegyzett változók
használata*



Hogyan működik mindez? A PHP 4 a háttérben folyamatosan fenntart egy ideiglenes fájlt a munkamenet számára. A `session_save_path()` függvény használatával e fájl elérési útját deríthetjük ki, de a függvénynek egy könyvtár elérési útját is megadhatjuk. Ha ezt meghatározzuk, az ideiglenes munkamenet-fájlok ezután

ide kerülnek. Ha a függvényt paraméter nélkül futtatjuk, visszatérési értéke annak a könyvtárnak az elérési útja lesz, ahová a rendszer az ideiglenes munkamenet-fájlokat menti. A szerző rendszerén a

```
print session_save_path();
```

visszatérési értéke /tmp. A /tmp könyvtár például a következő fájlokat tartalmazhatja:

```
sess_2638864e9216fee10fcb8a61db382909  
sess_76cae8ac1231b11afa2c69935c11dd95  
sess_bb50771a769c605ab77424d59c784ea0
```

Amikor megnyitjuk azt a fájlt, melynek neve a 20.1. példában visszaadott munkamenet-azonosítót tartalmazza, láthatjuk, hogyan tárolódnak a bejegyzett változók:

```
termek1|s:22:"Ultrahangos csavarhúzó";termek2|s:8:"HAL 2000";
```

Amikor meghívjuk a `session_register()` függvényt, a PHP beírja a fájlba a változó nevét és értékét. Később innen kapjuk meg a változó értékét.

Ha a `session_register()` függvénnyel bejegyeztünk egy változót, annak a PHP program futása során módosított értéke a munkamenet-fájlban is meg fog változni.

A 20.2. példa bemutatta a `session_register()` függvény használatát, de a függvény működése nem tűnik túl rugalmasnak. Mit tehetünk, ha a felhasználó többféle terméket tartalmazó változókat szeretne bevezetni? Szerencsére a `session_register()` függvény paraméterében tömb típusú változót is megadhatunk.

A 20.4. példában a felhasználó több termék közül választhat. Ezután munkamenet-változókkal létrehozunk egy egyszerű „bevásárlókosarat”.

20.4. program Tömb típusú változó bejegyzése

```
1: <?php  
2: session_start();  
3: ?>  
4: <html>  
5: <head>  
6: <title>20.4. program Tömb típusú változó  
   bejegyzése</title>  
7: </head>
```


20.4. program (folytatás)

```
8: <body>
9: <h1>Termékböngésző lap</h1>
10: <?php
11: if ( isset( $termekek_urlap ) )
12:     {
13:         $termekek = $termekek_urlap;
14:         session_register( "termekek" );
15:         print "<p>A termékeit bejegyeztük!</p>";
16:     }
17: ?><p>
18: <form method="POST">
19: <select name="termekek_urlap[]" multiple size=3>
20: <option> Ultrahangos csavarhúzó
21: <option> HAL 2000
22: <option> Kalapács
23: <option> TV
24: <option> Targonca
25: </select>
26: </p><p>
27: <input type="submit" value="Rendben">
28: </form>
29: </p>
30: <a href="20.5.program.php">Tartalom</a>
31: </body>
32: </html>
```

Először elindítjuk a munkamenetet a `session_start()` függvénnyel. Ezzel hozzáférünk minden korábban beállított munkamenet-változóhoz. A HTML dokumentumon belül a FORM elem ACTION tulajdonságát beállítjuk a jelenlegi dokumentumra. Létrehozunk egy `termekek_urlap[]` nevű SELECT elemet, amely a termékek számával megegyező OPTION összetevőt tartalmaz. Emlékezzünk rá, hogy a HTML dokumentum azon elemeinek, amelyek engedélyezik több elem kiválasztását, szögletes zárójellel együtt kell megadni a NAME paraméterét. Így a felhasználó választása egy tömbbe kerül.

A PHP kód blokkjában megvizsgáljuk, hogy létezik-e a `$termekek_urlap` tömb. Ha a változó létezik, feltehetjük, hogy a kérdőívet kitöltötték. Ezt a változót hozzárendeljük a `$termekek` változóhoz, amit a `session_register()` függvénnyel bejegyzünk. A `$termekek_urlap` tömböt közvetlenül nem jegyezzük be, mert ez egy későbbi kitöltés során ütközést okozhat az ugyanilyen nevű, POST eljárással érkezett változóval.

A kód végén megadunk egy hivatkozást, ahol bemutatjuk, hogy valóban hozzáférünk a bejegyzett tömbhöz. Ez a kód a 20.5. példában található.

20.5. program Hozzáférés bejegyzett tömb típusú változóhoz

```
1: <?php
2: session_start();
3: print session_encode(); //kódolt karaktersorozat kiírása
4: ?>
5: <html>
6: <head>
7: <title>20.5. program Hozzáférés bejegyzett
   tömb típusú változóhoz</title>
8: </head>
9: <body>
10: <h1>Tartalom lap</h1>
11: <?php
12: if ( isset( $stermekek ) )
13:     {
14:         print "<b>A bevásárlókocsi tartalma:</b><ol>\n";
15:         foreach ( $stermekek as $egytermek )
16:             print "<li>$egytermek";
17:         print "</ol>";
18:     }
19: ?>
20: <a href="20.4.program.php">Vissza
   a termékválasztáshoz</a>
21: </body>
22: </html>
```

A `session_start()` használatával folytatjuk a munkamenetet. Ellenőrizzük, hogy létezik-e a `$stermekek` változó. Ha létezik, elemeit egyenként kiírjuk a böngészőbe.

Természetesen egy igazi „bevásárlókosár-program” megírásakor a termékeket érdemes adatbázisban, és nem a futtatandó kódban tárolni. A 20.4. és 20.5. példa csak azt mutatta be, hogyan érhető el egy tömb típusú változó egy másik oldalról.

A munkamenet és a változók bejegyzésének törlése

A munkameneteket és a bejegyzett változókat a `session_destroy()` függvénnyel törölhetjük. A `session_destroy()` függvénynek nincsenek paraméterei, futtatásához azonban szükséges egy folyamatban lévő munkamenet. A következő kódrészlet létrehoz, majd rögtön megsemmisít egy munkamenetet:

```
session_start();  
session_destroy();
```

Amikor olyan oldalt töltünk be, amely egy munkamenetet vár, az már nem látja a korábban törölt munkamenetet, ezért létre kell hoznia egy újat. Minden korábban bejegyzett változó elvész.

Valójában a `session_destroy()` függvény nem semmisíti meg a bejegyzett változókat. Abban a kódban, amelyben a `session_destroy()` függvényt futtattuk, a változók továbbra is elérhetők. A következő kódrészletben látható, hogy az 5-re állított `$proba` változó a `session_destroy()` meghívása után is elérhető marad. A munkamenet megszüntetése nem törli a bejegyzett változókat.

```
session_start();  
session_register( "proba" );  
$proba = 5;  
session_destroy();  
print $proba; // kiírja, hogy 5
```

A munkamenetből a bejegyzett változókat a `session_unset()` függvénnyel távolíthatjuk el. A `session_unset()` a munkamenet-fájlból és a programból is eltávolítja a változókat, ezért óvatosan használjuk.

```
session_start();  
session_register( "proba" );  
$proba = 5;  
session_unset();  
session_destroy();  
print $proba; // nem ír ki semmit. a $proba változó  
    ➡ már nem létezik.
```

A munkamenet megszüntetése előtt meghívjuk a `session_unset()` függvényt, amely eltávolítja a `$proba`, és minden más bejegyzett változót a memóriából.

Munkamenet-azonosítók a lekérdező karakterláncban

Áttekintettük a sütin alapuló munkamenetek kezelését. Tudnunk kell azonban, hogy ez nem a legbiztosabb módszer. Nem feltételezhetjük ugyanis, hogy a látogató böngészője támogatja a sütik használatát. A probléma elkerülése végett építsünk be programunkba egy, a munkamenet-azonosító továbbadására alkalmas lekérdező karakterláncot. A PHP létrehozza a `SID` állandót, ha a böngésző nem küldött vissza azonosítót tartalmazó sütit. A `SID` név-érték alakú állandó karakterlánc, amelyet beágyazhatunk a munkamenetet támogató oldalra mutató HTML hivatkozásokba:

```
<a href="masik_lap.php"?<? print SID; ?>">Másik lap</a>
A böngészőben ez már így jelenik meg:
<a href="masik_lap.php?PHPSESSID=
    ➔ 08ecedf79fe34561fa82591401a01da1">Másik lap</a>
```

A munkamenet-azonosító automatikusan átvételre kerül, amikor a céloldalon meghívjuk a `session_start()` függvényt, és az összes korábban bejegyzett változó a szokásos módon elérhető lesz.

Ha a PHP 4-et az `--enable-trans-sid` kapcsolóval fordítottuk le a fenti lekérdező karakterlánc automatikusan hozzáadódik minden HTML hivatkozáshoz. Ez a lehetőség alapállapotban kikapcsolt, de használatával kódjaink hordozhatóbbá válnak.

Munkamenet-változók kódolása és visszafejtése

Amikor megnéztük a munkamenet-fájl tartalmát, láttuk, hogyan menti és kódolja a PHP a bejegyzett változókat. A kódolt karaktersorozathoz a `session_encode()` függvény használatával bármikor hozzáférhetünk. Ez hasznos lehet a munkamenetek nyomkövetésénél. A következő kódrészletben láthatjuk, hogyan működik a `session_encode()` függvény.

```
session_start();
print session_encode()."<br>";
// minta eredmény: termekek|a:2:{i:0;s:8:
    ➔ "HAL 2000";i:1;s:8:"Targonca";}
```

Látható, hogy a függvény tárolási alakjukban adja vissza a bejegyzett változókat, így ellenőrizhetjük, hogy a változó valóban be lett-e jegyezve és értéke megfelelő-e. A `session_encode()` függvény használatával bejegyzett változókat tartalmazó adatbázisokat is készíthetünk.

A `session_decode()` függvény használatával a változók kódolt formájából visszanyerhetjük az értéküket. A következő kódrészlet ezt mutatja be:

```
session_start();
session_unset(); // nem szabad, hogy létezzenek
    ➔ munkamenet-változók
session_decode( "termekek|a:2:{i:0;s:8:\"HAL 2000\";
    ➔ i:1;s:8:\"Targonca\";}" );
foreach ( $termekek as $egytermek )
{
    print "$egytermek<br>\n";
}
// Kimenet:
// HAL 2000
// Targonca
```

A szokásos módon elindítjuk a munkamenetet. Hogy tiszta lappal induljunk, a `session_unset()` függvénnyel töröljük az összes korábban bejegyzett változót. Ezután átadjuk a `session_decode()` függvénynek a változók kódolt formáját. A függvény párosítja a változóneveket a megfelelő értékekkel. Ezt ellenőrizzük is: a `$termekek` tömb minden elemét kiíratjuk.

20

Munkamenet-változó bejegyzésének ellenőrzése

Mint láttuk, a bejegyzett változó jelenlétét az `isset()` függvénnyel ellenőrizhetjük, de más módon is megvizsgálhatjuk a változó létezését. Erre való a `session_is_registered()` függvény. A függvény paraméterében egy változó nevét tartalmazó karakterláncot kell megadni, a visszatérési érték pedig `true`, ha a változó bejegyzett.

```
if ( session_is_registered ( "termekek" ) )
    print "A 'termekek' változó bejegyzett!";
```

Ez akkor lehet hasznos, ha biztosak akarunk lenni a változó forrásában, azaz, hogy a változó tényleg a munkamenet bejegyzett változója vagy csak egy GET kérelem eredményeként kaptuk.

Összefoglalás

Ebben és az előző órában áttekintettük, hogyan menthetjük az állapotokat egy állapot nélküli protokollban. Különféle módszereket alkalmaztunk: sütiket és lekérdező karakterláncokat használtunk, néha ezeket fájlokkal és adatbázisokkal ötvöztük. Mindegyik megközelítési módnak megvannak a maga előnyei és hátrányai.

A süti nem igazán megbízható és nem tárolhat sok információt. Ezzel szemben hosszú ideig fennmaradhat.

Az adatok fájlban vagy adatbázisban való tárolása miatt a sebesség csökkenhet, ami gondot okoz egy népszerű webhelyen. Ugyanakkor egy egyszerű azonosítóval nagy mennyiségű adathoz férhetünk hozzá.

A lekérdező karakterlánc a süti ellentéte. Elég csúnyán néz ki, de viszonylag nagy mennyiségű információt hordoz magában. Használatáról a körülmények mérlegelése után kell dönteni.

Ebben az órában megtanultuk, hogyan indítsunk el munkamenetet a `session_start()` függvénnyel. Láttuk, hogyan jegyezhetünk be változókat a `session_register()` függvénnyel, hogyan ellenőrizhetjük a bejegyzést a `session_is_registered()` függvénnyel, valamint hogyan semmisíthetjük meg a változókat a `session_unset()` függvénnyel. A munkamenet megsemmisítésére a `session_destroy()` függvényt használtuk.

Hogy minél több felhasználó érhesse el a munkamenetet támogató környezetet, lekérdező karakterláncainkban használjuk a `SID` állandót a munkamenet-azonosító átadására.

Kérdések és válaszok

Van valamilyen veszélye a munkamenet-függvények használatának, amiről még nem tudunk?

A munkamenet-függvények alapvetően megbízhatók. De emlékezzünk rá, hogy a sütik használata nem lehetséges több tartományon keresztül. Ezért, ha több tartománynevet használunk ugyanazon a kiszolgálón (elektronikus kereskedelemi alkalmazásoknál ez gyakori), tiltsuk le a sütik használatát a `session.use_cookies` 0-ra állításával a `php.ini` fájlban.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvényt használjuk egy munkamenet elindítására?
2. Melyik függvény adja vissza a jelenlegi munkamenet azonosítóját?
3. Hogyan rendelhetünk változókat a munkamenethez?
4. Hogyan semmisíthetjük meg a munkamenetet és törölhetjük el minden nyomát?
5. Hogyan semmisíthetjük meg az összes munkamenet-változót a futó programban és az egész munkamenetben?
6. Mi a `SID` állandó értéke?
7. Hogyan ellenőrizhetjük, hogy a `$proba` nevű változó be van-e jegyezve egy munkameneten belül?

20

Feladatok

1. Az előző óra Feladatok részében készítettünk egy programot, amely sütit vagy lekérdező karakterláncot használ, hogy oldalról oldalra vigye tovább a felhasználó válaszait. Ezután a környezet minden oldala a beállított háttérszínnel indult és név szerint üdvözölte a felhasználót. Készítsük el ezt a programot a PHP 4 munkamenet-függvényeivel.
2. Készítsünk programot, amely munkamenet-függvények használatával emlékezik rá, hogy a felhasználó a környezet mely oldalait látogatta meg. Készítsük el a felhasználó lépéseit tartalmazó hivatkozások sorozatát, hogy mozgása nyomon követhető legyen.



21. ÓRA

Munka kiszolgálói környezetben

A korábbi fejezetekben áttekintettük, hogyan társaloghatunk távoli számítógépekkel és hogyan vehetünk át adatokat a felhasználótól. Ebben a fejezetben ismét tekintünk, olyan eljárásokat tanulunk meg, amelyek külső programok saját gépünkön való futtatására használhatók. A fejezet példái Linux operációs rendszerre készültek, de a legtöbb alapelv felhasználható Microsoft Windows rendszerben is.

A fejezetben a következőket tekintjük át:

- Hogyan közvetítsünk adatokat a programok között?
- A héjparancsok végrehajtására és az eredmények megjelenítésére milyen lehetőségeink vannak?
- Hogyan írhatunk biztonságosabb PHP programokat?

Folyamatok összekötése a popen() függvénnyel

Ahogy a fájlokat nyitottuk meg írásra vagy olvasásra az `fopen()` segítségével, ugyanúgy nyithatunk adatcsatornát két folyamat között a `popen()` paranccsal. A `popen()` paramétereiben meg kell adni egy parancsot elérési úttal, és azt, hogy írási vagy olvasási módban akarjuk használni a függvényt. A `popen()` visszatérési értéke az `fopen()` visszatérési értékéhez hasonló fájlazonosító. A `popen()`-nek a 'w' jelzővel adhatjuk meg, hogy írási módban, az 'r' jelzővel pedig azt, hogy olvasási módban akarjuk használni a függvényt. Ugyanazzal az adatcsatornával nem lehet egyszerre írni és olvasni is egy folyamatot. Amikor befejeztük a munkát a `popen()` által megnyitott folyamattal, a `pclose()` paranccsal le kell zárunk az adatcsatornát. A `pclose()` függvény paraméterében egy érvényes fájlazonosítót kell megadni.

A `popen()` használata akkor javasolt, amikor egy folyamat kimenetét sorról sorra szeretnénk elemezni. A 21.1-es példában a GNU `who` parancs kimenetét elemezzük, és a kapott felhasználóneveket `mailto` hivatkozásokban használjuk fel.

21.1. program A who UNIX parancs kimenetének olvasása a popen() segítségével

```
1: <html>
2: <head>
3: <title>21.1. program A who UNIX parancs kimenetének
4: olvasása a popen() segítségével</title>
5: </head>
6: <body>
7: <h2>A rendszerbe bejelentkezett felhasználók</h1>
8: <?php
9: $ph = popen( "who", "r" )
10:      or die( "A 'who' paranccsal nem vehető fel
      kapcsolat" );
11: $kiszolgalo="www.kiskapu.hu";
12: while ( ! feof( $ph ) )
13: {
14:     $sor = fgets( $ph, 1024 );
15:     if ( strlen( $sor ) <= 1 )
16:         continue;
17:     $sor = ereg_replace( "^[a-zA-Z0-9_-]+).*",
18:     "<a
      href=\"mailto:\\1@$kiszolgalo\">\\1</a><br>\n",
19:     $sor );
20:     print "$sor";
21: }
```

21.1. program (folytatás)

```
22: pclose( $ph );  
23: ?>  
24: </body>  
25: </html>
```

A `who` parancs eredményének kiolvasásához szükségünk van egy fájlmutatóra a `popen()` függvénytől, így a `while` utasítás segítségével sorról sorra elemezhetjük a kimenetet. Ha a sorban olvasható kimenet csupán egyetlen karakter, a `while` ciklus következő lépésére ugrunk, kihagyva az elemzést, különben az `ereg_replace()` függvénnyel újabb HTML hivatkozással és soremeléssel bővítjük a végeredményt. Végül a `pclose()` függvénnyel lezárjuk az adatcsatornát. A program egy lehetséges kimenete a 21.1. ábrán látható.

21.1. ábra

A `who` UNIX parancs kimenetének olvasása



A `popen()` függvény használható parancsok bemenetének írására is. Ez akkor hasznos, ha egy parancs a szabványos bemenetről vár parancssori kapcsolókat. A 21.2. példában azt láthatjuk, hogyan használhatjuk a `popen()` függvényt a `column` parancs bemenetének írására.

21.2. program A column parancs bemenetének írása a popen() függvény segítségével

```

1: <html>
2: <head>
3: <title>21.2. program A column parancs bemenetének
   írása
4: a popen() függvény segítségével</title>
5: </head>
6: <body>
7: <?php
8: $stermek = array(
9:     array( "HAL 2000", 2, "piros" ),
10:    array( "Modem", 3, "kék" ),
11:    array( "Karóra", 1, "rózsaszín" ),
12:    array( "Ultrahangos csavarhúzó", 1,
        "narancssárga" )
13: );
14: $ph = popen( "column -tc 3 -s / >
fizetve/3as_felhasznalo.txt", "w" )
15:     or die( "A 'column' paranccsal nem vehető fel
        kapcsolat" );
16: foreach ( $stermek as $stermek )
17:     fputs( $ph, join('/', $stermek)."\n");
18: pclose( $ph );
19: ?>
20: </body>
21: </html>

```

A 21.2. példában megfigyelhető, hogyan érhetők el és írhatók ASCII táblázatként fájlba a többdimenziós tömbök elemei. A `column` parancs bemenetéhez adatcsatornát kapcsolunk, amin keresztül parancssori kapcsolatokat küldünk. A `-t` kapcsolóval megadjuk, hogy táblázattá formázza a kimenetet, a `-c 3` megadja a szükséges oszlopok számát, a `-s /` a `/-t` állítja be mezőelválasztóként. Megadjuk, hogy a végeredményt a `3as_felhasznalo.txt` fájlba írja. A `fizetve` könyvtárnak léteznie kell és a megfelelő jogosultság szükséges, hogy a program írhasson bele.

Most egyetlen utasítással egyszerre több dolgot tettünk. Meghívtuk a `column` programot és kimenetét fájlba írtuk. Valójában parancsokat adtunk ki a héjnak: ez azt jelenti, hogy az adatcsatorna használatával, egy folyamat futtatásával más feladatokat is elindíthatunk. A `column` kimenetét például a `mail` parancs segítségével postázhatjuk valakinek:

```
popen( "column -tc 3 -s / | mail kiskapu@kiskapu.hu", "w" )
```

Így elérhetjük, hogy a felhasználó által beírt adatokkal rendszerparancsokat hajtsunk végre PHP függvényekkel. Ennek néhány hátrányos tulajdonságát még áttekintjük az óra további részében.

Miután rendelkezünk egy fájlazonosítóval, a \$termek **tömb** minden elemén végiglépkedünk. Minden elem maga is egy **tömb**, amit a `join()` függvény segítségével karakterlánccá alakítunk. Az üres karakter helyett mezőelválasztóként a `' '` karaktert választjuk. Ez azért szükséges, mert ha üres karakterek jelennének meg a termékek **tömbjében**, az összezavarná a `column` parancsot. Az átalakítás után a karakterláncot és egy újsor karaktert írunk ki az `fputs()` függvénnyel.

Végül lezárjuk az adatcsatornát. A `3as_felhasznalo.txt` fájlban a következők szerepelnek:

HAL 2000	2	piros
Modem	3	kék
Karóra	1	rózsaszín
Ultrahangos csavarhúzó	1	narancssárga

A kódot hordozhatóbbá tehetjük, ha a szöveg formázására a `sprintf()` függvényt használjuk.

Parancsok végrehajtása az `exec()` függvénnyel

Az `exec()` egyike azoknak a függvényeknek, amelyekkel parancsokat adhatunk ki a héjnak. A függvény paraméterében meg kell adni a futtatandó program elérési útját. Ezen kívül megadható még egy **tömb**, mely a parancs kimenetének sorait fogja tartalmazni és egy változó, amelyből a parancs visszatérési értéke tudható meg.

Ha az `exec()` függvénynek az `'ls -al .'` parancsot adjuk meg, akkor az aktuális könyvtár tartalmát jeleníti meg. Ez látható a 21.3. példában.

21.3. program Könyvtárlista betöltése a böngészőbe

```
1: <html>
2: <head>
3: <title>21.3. program Könyvtárlista betöltése
   a böngészőbe </title>
4: </head>
5: <body>
```

21.3. program (folytatás)

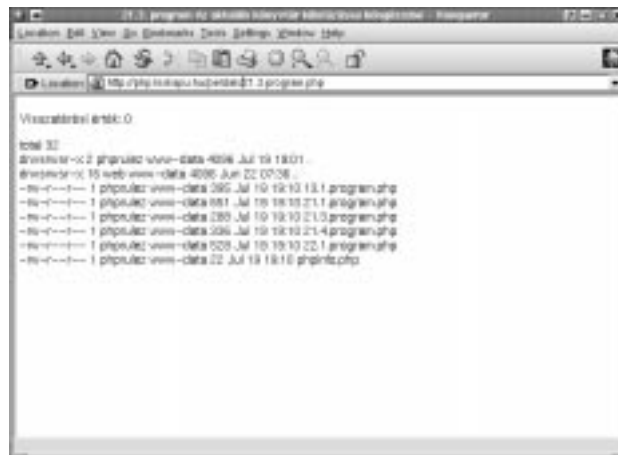
```
6: <?php
7: exec( "ls -al .", $kimenet, $vissza );
8: print "<p>Visszatérési érték: $vissza</p>";
9: foreach ( $kimenet as $fajl )
10:     print "$fajl<br>";
11: ?>
12: </body>
13: </html>
```

Ha az `ls` parancs sikeresen végrehajtódik, a visszatérési érték 0 lesz. Ha a program a megadott könyvtárat nem találja vagy az nem olvasható, a visszatérési érték 1.

A végeredmény szempontjából nem tettünk semmi újat, hiszen az `opendir()` és a `readdir()` függvényekkel ugyanezt elérhettük volna, de elképzelhető olyan eset, amikor rendszerparancsokkal vagy korábban megírt Perl programokkal sokkal gyorsabban megoldhatjuk a feladatot, mint PHP függvényekkel. Ha a PHP program kifejlesztésének sebessége fontos szempont, esetleg érdekesebb a korábban megírt Perl program meghívása mellett dönteni, mint átültetni azt PHP-be, legalábbis rövid távon. Meg kell azonban jegyeznünk, hogy rendszerparancsok használatával programjaink több memóriát fogyasztanak és többnyire futásuk is lassabb.

21.2. ábra

A könyvtárlista betöltése a böngészőbe az `exec()` függvény segítségével



Külső programok futtatása a `system()` függvénnyel vagy a ``` műveletjel segítségével

A `system()` függvény az `exec()` függvényhez hasonlóan külső programok indítására használható. A függvénynek meg kell adni a futtatandó program elérési útját, valamint megadható egy változó, amely a program visszatérési értékét tartalmazza majd. A `system()` függvény a kimenetet közvetlenül a böngészőbe írja. A következő kódrészlet a `man` parancs leírását adja meg:

```
<?php
print "<pre>";
system( "man man | col -b", $vissza );
print "</pre>";
?>
```

A PRE HTML elemet azért adtuk meg, hogy a böngésző megtartsa a kimenet eredeti formátumát. A `system()` függvényt használtuk, hogy meghívjuk a `man` parancsot, ennek kimenetét hozzákapcsoltuk a `col` parancshoz, amely ASCII-ként újraformázza a megjelenő szöveget. A visszatérési értéket a `$vissza` változóba mentjük. A `system()` közvetlenül a böngészőbe írja a kimenetet.

Ugyanezt az eredményt érhetjük el a ``` műveletjel használatával. A kiadandó parancsot egyszerűen ilyen fordított irányú aposztrófok közé kell tennünk. Az így megadott parancsot a rendszer végrehajtja és visszatérési értékként a parancs kimenetét adja, amit kiírhatunk vagy változóban tárolhatunk.

Íme az előző példa ilyenén megvalósítása:

```
print "<pre>";
print `man man | col -b`;
print "</pre>";
```

Vegyük észre, hogy ebben a megvalósításban rögtön kiírtuk a végeredményt.

Biztonsági rések megszüntetése az `escapeshellcmd()` függvény használatával

Az `escapeshellcmd()` függvény ismertetése előtt tekintsük át, mivel szemben van szükségünk védelemre. A példa erejéig tegyük fel, hogy meg szeretnénk engedni a látogatóknak, hogy különböző súgóoldalakat tekinthessenek meg. Ha bekérjük egy oldal nevét, a felhasználó bármit beírhat oda. A 21.4. példában található programot ne telepítsük, mert biztonsági rést tartalmaz!

21.4. program A man program meghívása

```

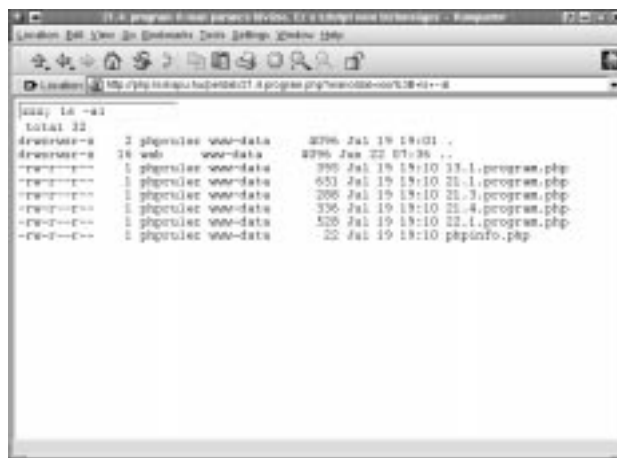
1: <html>
2: <head>
3: <title>21.4. program A man program meghívása.
4:     Ez a kód nem biztonságos</title>
5: </head>
6: <body>
7: <form>
8: <input type="text" value="<?php print $manoldal;
   ?>" name="manoldal">
9: </form>
10: <pre>
11: <?php
12: if ( isset($manoldal) )
13:     system( "man $manoldal | col -b" );
14: ?>
15: </pre>
16: </body>
17: </html>

```

Korábbi példánkat egy szöveges mezővel és a `system()` függvénnyel egészítettük ki. Megbízhatónak tűnik, UNIX rendszeren azonban a felhasználó a `manoldal` mezőhöz hozzáadhatja saját parancsait, így hozzáférhet a kiszolgálón számára tiltott részen lévő adatokhoz is. Erre láthatunk példát a 21.3. ábrán.

21.3. ábra

A man program meghívása



A felhasználó az oldalon keresztül kiadta az `xxx; ls -al` parancsot. Ezt a `$manoldal` változóban tároljuk. A program futása közben a `system()` függvénynek a következő parancsot kell végrehajtania:

```
"man xxx; ls -al | col -b"
```

Vagyis meg kell jelenítenie az `xxx` parancshoz tartozó súgóoldalt, ami természetesen nem létezik. Ebben az esetben a `col` parancs bemenete a teljes könyvtárlista lesz. Ezzel a támadó kiírathatja a rendszer bármelyik olvasható könyvtárának tartalmát. A következő utasítással például a `/etc/passwd` tartalmát kapja meg:

```
xxx; cat /etc/passwd
```

Bár a jelszavak egy titkosított fájlban, az `/etc/shadow`-ban vannak tárolva, amely csak a rendszergazda (`root`) által olvasható, ez mégis egy igen veszélyes biztonsági rés. A legegyszerűbben úgy védekezhetünk ellene, hogy soha nem engedélyezzük, hogy a felhasználók közvetlenül adjanak parancsot a rendszernek. Ennél kicsit több lehetőséget ad, ha az `escapeshellcmd()` függvénnyel fordított perjel (`\`) karaktert adunk minden metakarakterhez, amit a felhasználó kiadhat. Az `escapeshellcmd()` függvény paramétere egy karakterlánc, végeredménye pedig egy átalakított másolat. A korábbi kód biztonságosabb változata a 21.5. példában található.

21.5. program A felhasználói bemenet javítása az `escapeshellcmd()` függvény használatával

```
1: <html>
2: <head>
3: <title>21.5. program A felhasználói bemenet javítása
4:     az escapeshellcmd() függvény
   használata</title>
5: </head>
6: <body>
7: <form>
8: <input type="text" value="<?php print $manoldal; ?>"
   name="manoldal">
9: </form>
10: <pre>
11: <?php
```

21.5. program (folytatás)

```
12: if ( isset($manoldal) )
13:     {
14:         $manoldal = escapeshellcmd( $manoldal );
15:         system( "man $manoldal | col -b" );
16:     }
17: ?>
18: </pre>
19: </body>
20: </html>
```

Ha a felhasználó most kiadja az `xxx; cat etc/passwd` parancsot, akkor a `system()` függvény meghívása előtt az `escapeshellcmd()` az `xxx\; cat /etc/passwd` parancssá alakítja azt, azaz a `cat` utasítás sugóját kapjuk a jelszófájl helyett.

A rendszer biztonságát az `escapeshellcmd()` függvény segítségével tovább növelhetjük. Lehetőség szerint kerüljük azokat a helyzeteket, ahol a felhasználók közvetlenül a rendszernek adnak ki parancsokat. A programot még biztonságosabbá tehetjük, ha listát készítünk az elérhető sugóoldalakról és még mielőtt meghívnánk a `system()` függvényt, összevetjük a felhasználó által beírtakat ezzel a listával.

Külső programok futtatása a `passthru()` függvénnyel

A `passthru()` függvény a `system()`-hez hasonló, azzal a különbséggel, hogy a `passthru()`-ban kiadott parancs kimenete nem kerül átmeneti tárbá. Így olyan parancsok kiadására is lehetőség nyílik, amelyek kimenete nem szöveges, hanem bináris formátumú. A `passthru()` függvény paramétere egy rendszerparancs és egy elhagyható változó. A változóba kerül a kiadott parancs visszatérési értéke.

Lássunk egy példát! Készítsünk programot, amely a képeket kicsinyített mintaként jeleníti meg és meghívható HTML vagy PHP oldalalacról is. A feladatot külső alkalmazások használatával oldjuk meg, így a program nagyon egyszerű lesz. A 21.6. példában látható, hogyan küldhetünk a képről mintát a böngészőnek.

21.6. program A passthru() függvény használata képek megjelenítésére

```
1: <?php
2: if ( isset($kep) && file_exists( $kep ) )
3:     {
4:         header( "Content-type: image/gif" );
5:         passthru( "giftopnm $kep | pnmscale -xscale
                     .5 -yscale .5 | ppmtogif" );
6:     }
7: else
8:     print "A(z) $kep nevű kép nem található.";
9: ?>
```

Vegyük észre, hogy nem használtuk az `escapeshellcmd()` függvényt, ehelyett a felhasználó által megadott fájl létezését ellenőriztük a `file_exists()` függvénnyel. Ha a `$kep` változóban tárolt kép nem létezik, nem is próbáljuk megjeleníteni. A program még biztonságosabbá tehető, ha csak bizonyos kiterjesztésű fájlokat engedélyezünk és korlátozzuk az elérhető könyvtárakat is.

A `passthru()` hívásakor három alkalmazást indítunk el. Ha ezt a programot használni akarjuk, rendszerünkre telepítenünk kell ezen alkalmazásokat és meg kell adni azok elérési útját. Először a `giftopnm`-nek átadjuk a `$kep` változó értékét. Az beolvassa a GIF képet és hordozható formátumúra alakítja. Ezt a kimenetet rákapcsoljuk a `pnmscale` bemenetére, amely 50 százalékkal lekicsinyíti a képet. Ezt a kimenetet a `ppmtogif` bemenetére kapcsoljuk, amely visszaalakítja GIF formátumúvá, majd a kapott képet megjelenítjük a böngészőben.

A programot a következő utasítással bármelyik weboldalról meghívhatjuk:

```
">
```

Külső CGI program meghívása a virtual() függvénnyel

Ha egy oldalt HTML-ről PHP-re alakítunk, azt vesszük észre, hogy a kiszolgálóoldali beillesztések nem működnek. Ha Apache modulként futtatjuk a PHP-t, a `virtual()` függvénnyel külső CGI programokat hívhatunk meg, például Perl vagy C nyelven írt számlálókat. Minden felhasznált CGI programnak érvényes HTTP fejléccel kell kezdenie a kimenetét.

Írjunk egy Perl CGI programot! Ne aggódjunk, ha nem ismerjük a Perl-t. Ez a program egyszerűen egy HTTP fejléctet ír ki és minden rendelkezésre álló környezeti változót felsorol:

```
#!/usr/bin/perl -w
print "Content-type: text/html\n\n";
foreach ( keys %ENV ) {
    print "$_: $ENV{$_}<br>\n";
}
```

Mentsük a programot a `cgi-bin` könyvtárba `proba.pl` néven. Ezután a `virtual()` függvénnyel a következőképpen hívhatjuk meg:

```
<?php
virtual("/cgi-bin/proba.pl");
?>
```

Összefoglalás

Ebben a fejezetben áttekintettük, hogyan működhetünk együtt a héjjal és rajta keresztül a külső alkalmazásokkal. A PHP sok mindenre használható, de lehet, hogy külső programok használatával az adott probléma elegánsabb megoldásához jutunk.

Megtanultuk, hogyan kapcsolhatunk össze alkalmazásokat a `popen()` függvény segítségével. Ez akkor hasznos, amikor a programok a szabványos bemenetről várnak adatokat, de mi egy alkalmazás kimenetéből szeretnénk továbbítani azokat.

Megnéztük, hogyan használható az `exec()` és a `system()` függvény, valamint a fordított irányú aposztróf műveletjel a felhasználói utasítások közvetítésére a rendszermag felé. Láttuk, hogyan védekezzünk a betörésre használható utasítások ellen az `escapeshellcmd()` függvény segítségével. Láttuk, hogyan fogadhatók bináris adatok rendszerparancsoktól a `passthru()` függvénnyel és hogy hogyan valósíthatjuk meg a kiszolgálóoldali beillesztéseket (SSI) a `virtual()` függvénnyel.

Kérdések és válaszok

Sokat emlegettük a biztonságot ebben a fejezetben. Honnan tudhatunk meg többet a szükséges biztonsági óvintézkedésekről?

A legbővebb számítógépes biztonsággal foglalkozó forrás Lincoln Stein (a híres Perl modul, a CGI.pm szerzője) által fenntartott FAQ. Megtalálható a <http://www.w3.org/Security/Faq/> címen. Érdeemes a PHP kézikönyv biztonságról szóló részét is tanulmányozni.

Mikor használjunk külső alkalmazásokat saját PHP kódok helyett?

Három szempontot kell megvizsgálnunk: a hordozhatóságot, a fejlesztési sebességet és a hatékonyságot. Ha saját kódot használunk külső programok helyett, programunk könnyebben átvihető lesz a különböző rendszerek között. Egyszerű feladatoknál, például könyvtár tartalmának kiírásakor, saját kóddal oldjuk meg a problémát, így csökkenthetjük a futási időt, mivel nem kell minden futás alkalmával egy külső programot meghívunk. Másrésztől nagyobb feladatoknál sokat segíthetnek a kész külső programok, mivel képességeiket nehéz lenne megvalósítani PHP-ben. Ezekben az esetekben egy külön erre a célra készített külső alkalmazás használata javasolt.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

21

Kvíz

1. Melyik függvényt használjuk alkalmazások összekapcsolására?
2. Hogyan olvasunk adatokat egy folyamat kimenetéről, miután elindítottuk?
3. Hogyan írunk adatokat egy folyamat bemenetére, miután elindítottuk?
4. Az `exec()` függvény közvetlenül a böngészőbe írja a végeredményt?
5. Mit csinál a `system()` függvény a végrehajtott külső parancs kimenetével?
6. Mi a fordított irányú aposztróf visszatérési értéke?
7. Hogyan adhat ki biztonságosan a felhasználó parancsokat a rendszerhéjnak?
8. Hogyan hajthatunk végre külső CGI programot a PHP programokból?

Feladatok

1. Írjunk programot, amely a UNIX `ps` parancsának segítségével megjeleníti a böngészőben a futó folyamatokat! (Megjegyezzük, hogy nem biztonságos, ha a felhasználók futtathatják a programot!).
2. Nézzük meg a `ps` parancs súgójában a lehetséges parancssori kapcsolókat! Módosítsuk az előző programot, hogy a felhasználó a kapcsolók egy részét használhassa. Ne küldjünk ellenőrizetlen utasításokat a rendszermagnak!



22. ÓRA

Hibakeresés

E könyv írásakor a PHP 4 semmilyen hibakereső eszközt nem tartalmazott. A fejlesztők ígéretet tettek hibakereső szolgáltatások beépítésére, például hogy a verem tartalmát nyomon követhessük, ezért elképzelhető, hogy mire e könyv az Olvasó kezébe kerül, a legfrissebb kiadás már tartalmaz valamilyen fejlettebb hibakereső eszközt. Ebben a fejezetben a kódban rejlő hibák felderítésének néhány egyszerű módját mutatjuk be.

Az órában a következő témákkal foglalkozunk:

- A PHP beállításainak lekérdezése
- A PHP által automatikusan elérhetővé tett változók
- Hibaüzenetek kiírása naplófájlba
- Az adatok nyomon követése a programban
- A gyakori hibák felfedezése
- Információk a PHP-ről és adott programokról

Ha egy program nem működik megfelelően, érdemes először is a PHP beállításait megvizsgálnunk. Ezután jöhetnek a PHP által létrehozott és a saját változók, és ha még mindig nem találjuk a hibát, akkor megvizsgálhatjuk a forráskódot egy olyan eszközzel, amely színtkiemeléssel jelzi a nyelvtani elemeket – így hamar rábukkanhatunk a problémás részre. Ebben a részben két módszert is megvizsgálunk arra, hogyan szerezhetünk információkat a használt PHP-értelmezőről és magáról a futó programról.

A phpinfo()

A `phpinfo()` függvény az egyik leghasznosabb hibakereső eszköz: részletes információkkal szolgál magáról a PHP-ről, a kiszolgálói környezetről és a futó program változóiról. A függvénynek nem kell átadnunk semmilyen paramétert és csak egy logikai értéket ad vissza, viszont egy csinos HTML oldalt küld a böngészőnek. A `phpinfo()` kimenetét a 22.1. ábrán láthatjuk.

22.1. ábra

PHP információk megjelenítése



Az oldal tetején a használt PHP-változatról, a webkiszolgáló típusáról, a rendszerről és a szerzőkről találunk információkat. A következő táblázat részletezi a PHP beállításait – ezeket a `php.ini` fájlban módosíthatjuk. Tegyük fel például, hogy van egy "felhasznalo" nevű űrlapmezőnk, de a programban valamilyen okból nem jön létre a `$felhasznalo` változó. Vessünk egy pillantást a következő beállításokra:

```
track_vars      On
register_globals Off
```


Már meg is találtuk a probléma forrását. A `track_vars` hatására a GET változók a `$HTTP_GET_VARS[]` tömbben tárolódnak, a POST változók a `$HTTP_POST_VARS[]` tömbben, míg a süti a `$HTTP_COOKIE_VARS[]` tömbben. Ez eddig rendben is van, a `register_globals` kikapcsolása azonban azt jelenti, hogy a változók nem jönnek létre globális PHP változók formájában. Alapállapotban mindkét lehetőség engedélyezett. Ebben az esetben két lehetőségünk van. Keressük meg a `register_globals` bejegyzést a `php.ini` fájlban és változtassuk On-ra. Nem tudjuk, merre keressük a `php.ini` fájlt? Nos, a `phpinfo()` táblázataiban erről is kaphatunk információt. A másik lehetőségünk, hogy a "felhasznalo" mező tartalmára a program ne `$felhasznalo`-ként, hanem `$HTTP_POST_VARS["felhasznalo"]`-ként hivatkozzunk.

Egy olyan táblát is találunk kell, amely a felhasználói munkamenetek kezelésére vonatkozó beállításokat tartalmazza. Ha ez hiányzik, akkor a PHP-változatunkba nem fordítottuk bele a munkamenetek kezelésének támogatását. A táblázatban hasznos információkat találunk a munkamenetek kezelését megvalósító kód hibakereséséhez. Tegyük fel például, hogy olyan munkameneteket szeretnénk létrehozni, amelyek bizonyos ideig fennmaradnak. Ha a munkamenet elvész, amikor a felhasználó bezárja a böngésző ablakát, és a `phpinfo()` a következő beállítást mutatja:

```
session.cookie_lifetime      0
```

már meg is találtuk a probléma forrását. A `session.cookie_lifetime` értéket kell átállítanunk a `php.ini` fájlban, annak megfelelően, hogy hány másodpercig szeretnénk fenntartani a munkameneteket.

Ha a `php.ini` állomány a következő sort tartalmazza:

```
session.use_cookies          0
```

a süti nem engedélyezettek a munkamenetek kezelése során. Ebben az esetben az azonosítás során a lekérdező karakterláncra kell hagyatkoznunk vagy módosítanunk kell a beállítást a `php.ini`-ben.

A `phpinfo()` a webkiszolgálóról is rengeteg hasznos információval szolgál, különösen akkor, ha Apache fut a gépünkön. Láthatjuk például a programmal kapcsolatban forgalmazott összes kérés- és válaszfejléceket, illetve a kiszolgáló környezeti változóit is (például `HTTP_REFERER`).

Ha a PHP-t adatbázis-támogatással fordítottuk, az erre vonatkozó beállításokat is láthatjuk, például az alapértelmezett felhasználót, IP címet és kaput.

Az egyik legfontosabb információforrásunk lehet az a tábla, amelyben a PHP által létrehozott globális változók vannak felsorolva az értékeikkel együtt. Lássunk erre egy példát. A 22.1. példa egy egyszerű programot tartalmaz, amely létrehoz egy HTML űrlapot és beállít egy sütit.

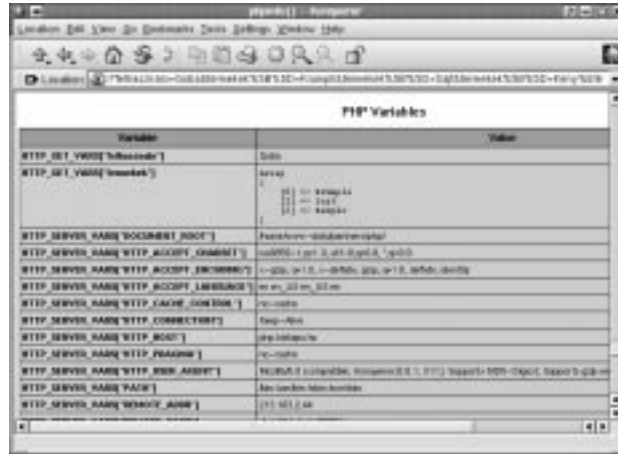
22.1. program A phpinfo() függvény kipróbálása

```
1: <?php
2: setcookie( "azonosito", "2344353463433",
           time()+3600, "/" );
3: ?>
4: <html>
5: <head>
6: <title>22.1. program A phpinfo() függvény
   kipróbálása</title>
7: </head>
8: <body>
9: <form action="<?php print "$PHP_SELF" ?>"
   METHOD="get">
10: <input type="text" name="felhasznalo">
11: <br>
12: <select name="termekek[]" multiple>
13: <option>Krumpli
14: <option>Sajt
15: <option>Kenyér
16: <option>Csirke
17: </select>
18: <br>
19: <input type="submit" value="Lássuk!">
20: </form>
21: <p></p>
22: <hr>
23: <p></p>
24: <?php
25: phpinfo();
26: ?>
27: </body>
28: </html>
```

Ha a „Lássuk!” gombra kattintunk, a program megkapja a felhasználó által megadott adatokat és a süti is beállítódik. Ha meghívjuk a `phpinfo()` függvényt, látni fogjuk ezeket a változókat – a kimenet lényeges részét a 22.2. ábrán láthatjuk.

22.2. ábra

*Globális változók
elérése*



Variable	Value
\$HTTP_GET_VARS["name"]	John
\$HTTP_GET_VARS["name"]	John
\$HTTP_SERVER_VARS["DOCUMENT_ROOT"]	/usr/local/apache/htdocs
\$HTTP_SERVER_VARS["HTTP_ACCEPT"]	*/*.*; q=0.5; q=0.5; q=0.5
\$HTTP_SERVER_VARS["HTTP_ACCEPT_ENCODING"]	gzip, deflate, q=0.5; q=0.5; q=0.5
\$HTTP_SERVER_VARS["HTTP_ACCEPT_LANGUAGE"]	en-us; q=0.5; q=0.5; q=0.5
\$HTTP_SERVER_VARS["HTTP_CACHE_CONTROL"]	no-cache
\$HTTP_SERVER_VARS["HTTP_CONNECTION"]	Keep-Alive
\$HTTP_SERVER_VARS["HTTP_HOST"]	192.168.1.1
\$HTTP_SERVER_VARS["HTTP_REFERER"]	http://192.168.1.1:80/
\$HTTP_SERVER_VARS["HTTP_USER_AGENT"]	Mozilla/5.0 (X11; Linux i686; rv:1.9.0.1) Gecko/20100101 Firefox/3.5.1
\$HTTP_SERVER_VARS["PATH"]	/usr/local/apache/htdocs
\$HTTP_SERVER_VARS["REMOTE_ADDR"]	192.168.1.1

Látható, hogy a süti és a `$HTTP_GET_VARS` változó elérhető. Az űrlap tartalmazott egy olyan listát is, amelyből több elemet is kiválaszthattunk – a változók között a teljes tömb megjelenik.

Nagyobb lélegzetű feladatoknál gyakran gondot okoz az űrlapváltozók és a sütik nyomon követése, ilyenkor a `phpinfo()` felbecsülhetetlen segítséget nyújthat.

A forráskód megjelenítése színelőemelésel

Ha nem találjuk meg a probléma forrását a `phpinfo()` függvény segítségével, talán nem is a beállításokkal van baj. Jó ötlet lehet egy újabb pillantást vetni a forráskódra. A PHP segítségével megtekinthetjük a program forráskódját, ráadásul a kulcsszavakat, a karakterláncokat, a megjegyzéseket és a HTML kódot színelőemelésel is megjeleníthetjük.

Ha Apache kiszolgálót használunk, a beállítófájlhoz (többnyire `httpd.conf`) kell hozzáadnunk a következő sort:

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

Ezután minden `.phps` kiterjesztésű fájl színelőemelésel fog megjelenni a böngészőablakban. Ha nincs jogunk megváltoztatni a kiszolgáló beállítóállományát, használjuk a `show_source()` függvényt, amely a paraméterül megadott fájl színelőemelésel jeleníti meg a böngészőablakban.

A 22.2. példaprogram segítségével megtekinthetjük programjaink forráskódját.

22.2. program Dokumentum forrásának megjelenítése

```

1: <html>
2: <head>
3: <title>22.2. program Dokumentum forrásának
   megjelenítése</title>
4: </head>
5: <body>
6: <form action="<?php print "$PHP_SELF" ?>"
   method="get">
7: Kérem a fájl nevét:
8: <input type="text" name="file" value="<?php print
   $file; ?>">
9: <p></p><hr><p></p>
10: <?php
11: if ( isset( $file ) )
12:     show_source( $file ) or print "nem lehet
        megnyitni a következő fájlt: \"$file\"";
13: ?>
14: </body>
15: </html>

```

A 22.3. ábra a 22.2. példaprogramot mutatja működés közben.

22.3. ábra

*Színtkiemelés
használata*



Miért hasznos ez a lehetőség? Hiszen megnézhetnénk a kódot megszokott szövegszerkesztőnkkel is. A legnagyobb előny a színtkiemelésben rejlik. Igen könnyű észrevenni például az elgépeléseket, hiszen ezeket nem kulcsszóként értelmezi a megjelenítő, ezért más színnel jelöli.

Egy másik gyakori probléma, hogy elég nehéz nyomon követni egy félig nyitott idézőjel-párt. Mivel a PHP az idézőjel utáni szöveget karakterláncként értelmezi, gyakran rossz helyen jelzi a hibát. Az idézőjelben lévő karakterláncok szintén más színnel jelöltek, így ez a típusú hiba nagyon könnyen észrevehető.



Ne tegyük ezt a programot elérhetővé webhelyünkön, mert így bárki belenézhet forrásainkba. Csak fejlesztéskor használjuk ezt a kódot!

A 22.1. táblázat a szíнкиemelések jelentését tartalmazza.

22.1. táblázat Szíнкиemelések

<i>php.ini bejegyzés</i>	<i>Alapértelmezett szín</i>	<i>Kód</i>	<i>Jelentés</i>
<code>highlight.string</code>	Vörös	#DD0000	Idézőjelek és karakterláncok
<code>highlight.comment</code>	Narancs	#FF8000	PHP megjegyzések
<code>highlight.keyword</code>	Zöld	#007700	Műveletjelek, nyelvi elemek és a legtöbb beépített függvény
<code>highlight.default</code>	Kék	#0000BB	Minden egyéb PHP kód
<code>highlight.html</code>	Fekete	#000000	HTML kód

PHP hibaüzenetek

Miközben e könyv segítségével elsajátítottuk a PHP programozás alapjait, bizonyára előfordult, hogy hibaüzeneteket kaptunk a várt eredmény helyett. Például elfelejtettünk zárójelbe tenni egy kifejezést vagy elgépeltek egy függvény nevét. A hibaüzenetek nagyon fontosak a hibakeresés során. Állítsuk a `php.ini` fájlban a `display_errors` bejegyzést "On"-ra, ezzel biztosíthatjuk, hogy a PHP elküldje a hibaüzeneteket a böngészőnek. Ne feledjük, hogy a `phpinfo()` függvény segítségével megnézhetjük, hogy be van-e kapcsolva ez a lehetőség.

Miután meggyőződünk arról, hogy programunk tudatni fogja velünk az esetleges hibákat, meg kell adnunk, hogy mennyire legyenek „szigorúak” az üzenetek. Ha be akarunk állítani egy alapértelmezett szintet, rendeljük a `php.ini` fájlban az `error_reporting` bejegyzéshez egy számot. Szerencsére nem kell fejben

tartanunk az összes számot, mivel rendelkezésünkre állnak azok az állandók, amelyeket a hibakezelés szintjének beállításához használhatunk. A különböző értékeket a 22.2. táblázat tartalmazza.

22.2. táblázat Hibakezelési szintek

<i>Állandó</i>	<i>Név</i>	<i>Leírás</i>	<i>Mi történik?</i>
E_ALL	Mind	Mindenféle hiba	Függ a hiba típusától
E_ERROR	Hibák	Végzetes hibák (például memória- kiosztási problémák)	Felhasználó értesítése és a program futásának megállítása
E_WARNING	Figyelmeztetések	Nem végzetes hibák (például nem szabá- lyos paraméterátadás)	Értesíti a felhasználót, de nem szakítja meg a program futását
E_PARSE	Értelmező hiba	Az értelmező nem érti az utasítást	Felhasználó értesítése és a program futásának megállítása
E_NOTICE	Megjegyzések	Lehetséges problé- maforrás (például egy kezdeti értékkel el nem látott változó)	Értesíti a felhasználót és folytatja a program futtatását
E_CORE_ERROR	Belső hiba az értel- mező indításakor	Az értelmező indítása- kor fellépő végzetes hibák	Megszakítja az értelmező indítását
E_CORE_WARNING	Figyelmeztető üzenet az értel- mező indításakor	Az értelmező indítása- kor fellépő nem végzetes hibák	Értesíti a felhasználót és folytatja a program futtatását

Ezekkel a beállításokkal nem változtathatjuk meg, mi történjen a hiba felbukkanásakor, csak abba van beleszólásunk, hogy a hibaüzenetet el akarjuk-e küldeni a böngészőnek vagy sem.

Természetesen egyszerre több hibakezelési szintet is engedélyezhetünk, ekkor az egyes szinteket a VAGY (|) jellel elválasztva kell megadnunk. A következő sor például engedélyezi a hibaüzenetek és a figyelmeztetések megjelenítését is:

```
error_reporting = E_ERROR|E_WARNING
```

Ha a hibakezelést az összes hibatípusra ki akarjuk terjeszteni, használjuk az `E_ALL` állandót. Mi a teendő akkor, ha az összes típust engedélyezni szeretnénk, kivéve egyet? A következő sor éppen ezt valósítja meg:

```
error_reporting = E_ALL & ~E_NOTICE
```

Az értelmező a megjegyzéseken kívül minden felmerülő üzenetet elküld a böngészőnek.

Az `E_ERROR|E_WARNING` és az `E_ALL&~E_NOTICE` tulajdonképpen kettes számrendszerbeli aritmetikai műveletek, melyek eredménye egy új hibakezelési szintet megadó szám. A kettes számrendszerbeli aritmetikával ebben a könyvben nem foglalkozunk, de a módszer ettől még remélhetőleg érthető marad.

A `php.ini` beállítását az `error_reporting()` függvénnyel bírálhatjuk felül, melynek bemenő paramétere a hibakezelési szintet megadó egész szám, visszatérési értéke pedig a megelőző hibakezelési beállítás. Természetesen ennél a függvényenél is használhatjuk az előzőekben megismert állandókat. Lássunk egy példát egy olyan esetre, ahol a hibakezelési szint módosítása a segítségünkre lehet. Lássuk, észrevesszük-e a szándékolt hibát a 22.3. programban.

22.3. program Egy szándékos hiba

```
1: <?php
2: error_reporting( E_ERROR|E_WARNING|E_PARSE );
3: $flag = 45;
4: if ( $flag == 45 ) {
5:     print "Tudom, hogy a \$flag változó értéke 45";
6: } else {
7:     print "Tudom, hogy a \$flag változó értéke NEM 45";
8: };
9: ?>
```

Mint látható, a `$flag` változó értékét akarjuk vizsgálni, de elgépeztük. Nincs végzetes hiba a programban, így minden további nélkül lefut és az `E_ERROR|E_WARNING|E_PARSE` hibakezelési beállítás mellett még csak üzenetet sem küld a böngészőnek. A kód bekerül a programba és a lehető legrosszabb pillanatban működésbe lép. Ha azonban az `error_reporting()` függvénynek az `E_ERROR|E_WARNING|E_PARSE|E_NOTICE` értéket adnánk át (ez magában foglalja a megjegyzések elküldését is), a program a következő kimenetet küldené:

```
Warning: Undefined variable: flg in /home/httpd/htdocs/  
    ➔ peldak/22.3.program.php on line 4  
Tudom, hogy a $flag változó értéke NEM 45
```

Azaz:

```
Figyelem: Nem meghatározott változó: flg a /home/httpd/htdocs/  
    ➔ peldak/22.3.program.php fájlban a 4. sorban  
Tudom, hogy a $flag változó értéke NEM 45
```

Az üzenetből nem csak az derül ki számunkra, hogy valami nincs rendben, de még a problémás sor számát is megtudtuk. Ugyanezt a hatást úgy is elérhetjük, ha az `error_reporting()` függvénynek az `E_ALL` állandót adjuk át paraméterként, ennek azonban lehetnek nem kívánt mellékhatásai is. Elképzelhető az is, hogy szándékosan alkalmazunk nem meghatározott változókat. Vegyük például a következő kódrészletet:

```
<INPUT TYPE="text" NAME="felhasznalo" VALUE="<?"  
    ➔ print $felhasznalo; ?>">
```

Itt azt használjuk ki, hogy ha egy nem meghatározott változó értékét írjuk ki, akkor annak hasonló a hatása ahhoz, mintha egy üres karakterláncot jelenítenénk meg (tulajdonképpen semmilyen hatása nincs). A "felhasznalo" mező egy előzőleg elküldött értéket vagy semmit sem tartalmaz. Ha a megjegyzések elküldését is engedélyeztük, hibaüzenetet kapunk.

Hibaüzenetek kiírása naplófájlba

Azok a hibák, amelyekre eddig vadásztunk, nagyobb részben azonnali, fejlesztés közben keletkező hibák voltak. Vannak azonban olyan problémák is, amelyek később jönnek elő, például azért, mert megváltozik a futtató környezet. Tegyük fel például, hogy szükségünk van egy programra, amely a felhasználó által kitöltött űrlap adatait írja fájlba. Elkészítjük a programot, kipróbáljuk, mérjük a teljesítményét éles körülmények között, majd magára hagyjuk, hogy végezze a feladatát. Néhány héttel később azonban véletlenül töröljük azt a könyvtárat, amely az adott fájlt tartalmazza.

A PHP `error_log()` függvénye pontosan az ilyen esetekre kifejlesztett hibakereső eszköz. Ez egy beépített függvény, amellyel a kiszolgáló naplófájljába vagy egy általunk megadott fájlba naplózhatjuk a felmerülő hibákat. Az `error_log()` függvénynek két bemenő paramétere van: egy karakterlánc, amely a hiba szövegét tartalmazza és egy egész szám, amely a hiba típusát írja le. A hiba típusától függően egy további paramétert is meg kell adnunk az `error_log()` függvénynek: egy elérési utat vagy egy e-mail címet.

A 22.3. táblázat az elérhető hibatípusokat sorolja fel.

22.3. táblázat Az `error_log()` függvény hibatípusai

Érték	Leírás
0	A hibaüzenetet a <code>php.ini</code> fájl <code>error_log</code> bejegyzésénél megadott fájlba kell kiírni.
1	A hibaüzenetet a harmadik paraméterben megadott e-mail címre kell küldeni.
3	A hibaüzenetet a harmadik paraméterben megadott fájlba kell kiírni.

Ha hibanaplózást szeretnénk, adjunk meg egy naplófájl a `php.ini` `error_log` bejegyzésénél. Természetesen a `phpinfo()` függvénnyel megnézhetjük, van-e már ilyen beállítás a `php.ini` fájlban. Ha nincs, adjuk hozzá a fájlhoz például a következő bejegyzést:

```
error_log = /home/httpd/logs/php_errors
```

Ha NT vagy Unix operációs rendszert használunk, a "syslog" karakterláncot is hozzárendelhetjük az `error_log` bejegyzéshez. Ekkor a 0-ás hibatípussal meghívott `error_log()` hibaüzenetek a rendszernaplóba (Unix), illetve az eseménynaplóba (NT) kerülnek. A következő kódrészlet egy hibaüzenetet hoz létre, ha nem tudunk megnyitni egy fájlt:

```
fopen( "./fontos.txt", "a" )
    or error_log( __FILE__, "__LINE__". sor:
    ➡ Nem lehet megnyitni a következő fájlt: ./fontos.txt", 0 );
```

A hibaüzenet a `__FILE__` és `__LINE__` állandókat használja, amelyek az éppen futó program elérési útját, illetve az aktuális sor számát tartalmazzák. A 0-ás hibatípussal azt adjuk meg, hogy a hibaüzenet a `php.ini` fájlban megadott helyre kerüljön. Ezek alapján valami ilyesmi bejegyzés kerül a naplófájlba:

```
/home/httpd/htdocs/proba5.php, 4. sor:
    ➡ Nem lehet megnyitni a következő fájlt: ./fontos.txt
```

Ha a hibaüzenetet adott fájlba akarjuk kiküldeni, használjuk a 3-as hibatípus paramétert:

```

if ( ! $megvan = mysql_connect( "localhost", "jozsi",
    "valami" ) )
{
    error_log( date("d/m/Y H I")." Nem lehet csatlakozni
        az adatbázishoz",
        3, "dbhiba.txt" );
    return false;
}

```

Ha 3-as típusú hibaüzenetet hozunk létre, egy harmadik paraméterben azt is meg kell adnunk az `error_log()` függvénynek, hogy hová kerüljön a hibaüzenet.

Az `error_log()` függvénynek azt is megadhatjuk, hogy a hibaüzenetet egy e-mail címre postázza. Ehhez az 1-es típuskódot kell használnunk:

```

if ( ! file_exists( "kritikus.txt" ) )
    or error_log( "Ó, nem! kritikus.txt-nek vége van!",
    ➔ 1, "veszeset@kritikus.hu" );

```

Ha 1-es típusú hibaüzenetet küldünk, az `error_log()` harmadik paraméterében meg kell adnunk az e-mail címet is.

A hibaüzenet elfogása

Ha hibaüzeneteket írunk egy fájlba vagy elektronikus levélben küldjük el azokat, hasznos lehet, ha hozzáférünk a PHP által előállított hibaüzenethez.

Ehhez a `php.ini` fájlban a `track_errors` bejegyzést "On"-ra kell állítanunk – ennek hatására a legutolsó PHP hibaüzenet bekerül a `$php_errormsg` változóba (hasnólóan a Perl `$!` változójához).

A `$php_errormsg` változó felhasználásával a hibaüzeneteket és naplóbejegyzéseket beszédesebbé tehetjük.

A következő kódrészlet által létrehozott hibaüzenet már sokkal használhatóbb, mint az előzőek:

```

fopen( "./fontos.txt", "a" )
    or error_log( __FILE__."", ".__LINE__". sor:
        ".$php_errormsg, 0 );

```

Az üzenet így fog festeni:

```

/home/httpd/htdocs/proba.php, 21. sor:
fopen("./fontos.txt","a") - Permission denied

```


22.4. program (folytatás)

```

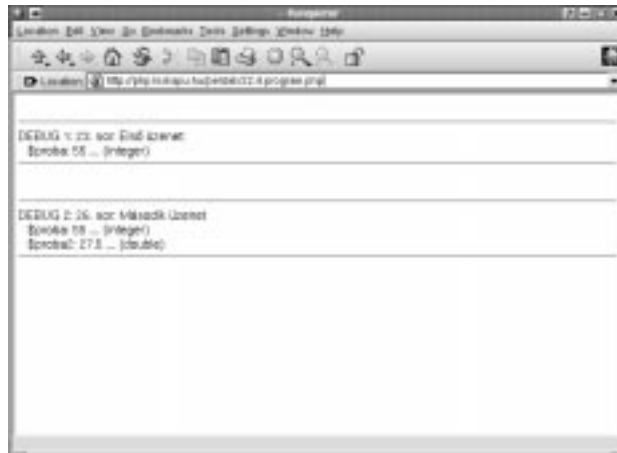
15:             print " .... (".gettype(
                    $argumentumok[$x+1] ).")<BR>\n";
16:         }
17:     }
18:     print "<hr><p></p>\n";
19:     $hivasok++;
20: }
21: $proba = 55;
22: debug( __LINE__, "Első üzenet:", "proba", $proba );
23: $teszt = 66;
24: $proba2 = $proba/2;
25: debug( __LINE__, "Második üzenet", "proba", $proba,
        "proba2", $proba2 );
26: ?>

```

A `debug()` függvénynek egy sorszámot, egy üzenetet, illetve tetszőleges számú név–érték párt adhatunk át. A sor számát és az üzenetet kiírni a feladat könnyebbik része. Ezután ki kell használnunk a PHP 4 azon tulajdonságát, hogy egy függvényt változó számú paraméterrel is meghívhatunk. A `func_get_args()` által visszaadott tömb függvényünk összes paraméterét tartalmazza – ezeket a változókat töltjük be az `$argumentumok` tömbbe. Mivel név–érték párokat várunk, hibaüzenetet kell adnunk, ha páratlan számú elem érkezik a tömbben. Egyébként végiglépkedünk a tömb összes elemén (a harmadikkal kezdve) és kiírjuk az összes párt a változó adattípusával együtt. A 22.4. ábra a 22.4. program kimenetét mutatja.

22.4. ábra

A debug() függvény használata



Gyakori hibák

Mindannyian követünk el hibákat, különösen akkor, ha vészesen közeledik a leadási határidő. Vannak olyan csapdák, amelyekbe előbb-utóbb minden programozó beleesik.

Észrevesszük a hibát a következő kódrészletben?

```
$valtozo=0;
while ( $valtozo < 42 );
{
    print "$valtozo<BR>";
    $valtozo++;
}
```

A kapkodás sokszor vezet hibához – például az utasítássort jelölő pontosvesszőt gyakran kitesszük olyan helyeken is, ahol nem kellene. Így kerülhetett a fenti while kifejezés mögé is pontosvessző, ami azt jelzi az értelmezőnek, hogy a ciklus törzse üres – a \$valtozo értéke soha nem fog nőni, ezért a program végtelen ciklusba kerül.

Lássunk még egy példát:

```
$ertek = 1;
while ( $ertek != 50 )
{
    print $ertek;
    $ertek+=2;
}
```

Az előletesztelő ismétléses vezérlési szerkezet ciklustörzsének végrehajtása a zárójelben található kifejezés logikai értékétől függ. A fenti példában ez a kifejezés csak akkor lesz hamis, ha az \$ertek változó értéke 50. Vagyis a ciklustörzs addig ismétlődik, amíg az \$ertek értéke nem 50. Csakhogy ez soha nem fog bekövetkezni, mivel a változó értéke 1-ről indul és minden lépésben kettővel növekszik – vagyis csak páratlan értékeket vesz fel. Ismét sikerült végtelen ciklusba ejtenünk a PHP-t.

A következő kódrészletben rejlő hiba elég alattomos:

```
if ( $proba = 4 )
{
    print "<P>A \"$proba\" értéke 4</P>\n";
}
```

A végrehajtást vezérlő logikai kifejezésben nyilvánvalóan ellenőrizni akartuk a `$proba` változó értékét. A kód azonban ehelyett 4-et rendel a `$proba`-hoz. Mivel a hozzárendelés mint kifejezés értéke a jobb oldali operandussal egyenlő (jelen esetben 4-gyel), a logikai kifejezés értéke a `$proba`-tól függetlenül mindig igaz lesz. Ezt a típusú hibát azért nehéz felfedezni, mert semmilyen üzenetet nem hoz létre, a program egyszerűen nem úgy fog működni, ahogyan azt elvárjuk.

A karakterláncok kezelése során szintén elkövethetünk típushibákat. Az alábbi elég gyakori:

```
$datum = "A mai dátum: . date('d/m/Y H I');  
print $datum;
```

Mivel nem zártuk le az idézőjelet, az értelmezőnek fogalma sincs, hol végződik a karakterlánc. Ez szinte mindig hibajelzéssel jár, de nehézkes visszakeresni, mivel gyakran rossz sorszámot kapunk. A zárójelpárok hibás kezelése szintén hasonló jelenséghez vezet.

A következő hibát viszont elég könnyű megtalálni:

```
print "Ön a $startomany-ról "$felhasznalo"-kent  
    ➡ jelentkezett be";
```

Ha idézőjelet akarunk kiíratni, jeleznünk kell az értelmezőnek, hogy itt nem különleges karakterként kell kezelnie. Az előbbi kódrészlet a következőképpen fest helyesen:

```
print "Ön a $startomany-ról \"$felhasznalo\"-kent  
    ➡ jelentkezett be";
```

Összefoglalva, ebben a részben a következő gyakori hibatípusokkal foglalkoztunk:

- Üres ciklustörzs létrehozása hibásan alkalmazott pontosvesszővel.
- Végtelen ciklus létrehozása hibás vezérlőfeltétellel.
- Hozzárendelés használata az „egyenlő” logikai műveletjel helyett.
- Levédetlen idézőjelek használata miatt elcsúszott karakterlánc-határok.

Összefoglalás

A fejlesztés talán egyik leghálátlanabb része a hibakeresés. A megfelelő módszerekkel azonban megkönnyíthetjük az életünket.

Ebben az órában tanultunk arról, hogy a `phpinfo()` függvény segítségével hogyan nyerhetünk információkat a rendszer beállításairól és a környezeti változókról. Az `error_log()` függvénnyel hibaüzeneteket írtunk ki fájlba és küldtünk el elektronikus levélben. Tanultunk a kézi hibakeresésről a fontosabb változók értékének kiírásával és írtunk egy függvényt a feladat megkönnyítésére. Ha ismerjük az adatokat, amelyekkel programunk dolgozik, viszonylag könnyen és gyorsan megtalálhatjuk a hibákat. Végül megnéztünk néhány olyan típushibát, amelyek még a tapasztaltabb programozók életét is megkeserítik.

Kérdések és válaszok

Létezik módszer, amellyel csökkenthetjük a kódba kerülő hibákat?

Legyünk szemfülesek! Tulajdonképpen szinte lehetetlen egy olyan nagyobb lélegzetű programot létrehozni, amely elsőre tökéletesen működik. Használjunk moduláris tervezést, hogy gyorsan elkülöníthessük a hibát tartalmazó kódrészletet. Olyan gyakran ellenőrizzük a kódot, amilyen gyakran csak lehetséges. Rossz megközelítés, ha megírjuk a programot, elindítjuk, azután várjuk, mi történik. Oldjunk meg egy részfeladatot (lehetőleg függvények formájában), majd próbáljuk ki az elkészült kódot. Ha tökéletesen működik, mehetünk a következő részfeladatra. Próbáljunk „bolondbiztos” kódot készíteni és ellenőrizzük működését szélsőséges körülmények között. Ha az egyik leglényegesebb lépés például az adatok kiírása egy megadott fájlba, nézzük meg, mi történik, ha a fájl hiányzik.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvénnyel kaphatunk hasznos információkat a PHP-ről és a rendszerünkről?
2. Melyik függvénnyel jeleníthetjük meg a színekkel kiemelt forráskódot a böngészőablakban?

3. Melyik `php.ini` bejegyzéssel szabályozhatjuk a hibaüzenetek kijelzésének szintjét?
4. Melyik függvénnyel bírálhatjuk felül ezt a beállítást?
5. Melyik függvénnyel naplózhatjuk a hibákat?
6. Melyik beépített változó tartalmazza a hibaüzenetet, ha a `track_errors` bejegyzés engedélyezett a `php.ini` fájlban?

Feladatok

1. Vizsgáljuk meg eddig elkészült programjaink kódját és felépítését az ebben az órában tanult módszerek segítségével.



IV. RÉSZ

Összefoglaló példa

23. óra Teljes példa (első rész)

24. óra Teljes példa (második rész)



23. ÓRA

Teljes példa (első rész)

Ha figyelmesen követtük az egyes órák anyagát, jó alapokkal rendelkezünk a PHP programozáshoz. Ebben és a következő órában egy teljes, működő programot készítünk, amely az előző órákban tárgyalt eljárásokból építkezik.

Az órában a következőket tanuljuk meg:

- Hogyan készítsünk tervet?
- Hogyan használjuk az `include()` nyelvi szerkezetet, hogy függvénykönyvtárakat és újrahasznosítható navigációs elemeket készítsünk?
- Hogyan tartsuk nyilván az állapotokat GET típusú lekérdezésekkel, adatbázisokkal és munkamenet-függvényekkel?
- Hogyan válasszuk el a HTML és a PHP kódot, hogy a programozásban járatlan grafikus is dolgozhasson az oldalon?
- Hogyan védjük meg szolgáltatásunk oldalait felhasználó-azonosítással?

A feladat rövid leírása

Tegyük fel, hogy egy közösségi webhely tulajdonosai felkértek bennünket, hogy készítsünk el egy kis interaktív eseménynaptárt az általuk kiszolgált kisváros számára. Klubok és együttesek jegyeztethetik be magukat, hogy reklámozzák rendezvényeiket. A webhely felhasználói ezután különböző formákban kiíráthatják majd az adatbázist, hogy lássák, milyen rendezvények lesznek a városban. A felhasználók képesek lesznek a lista szűkítésére a klubok típusa vagy akár a szervezett rendezvény helye szerint is.

Az oldalak felépítése

Mielőtt akár egy sor kódot is íránk, el kell döntenünk, hogyan fog működni programunk. Milyen módon fogják elérni a felhasználók a rendszer különböző elemeit? Milyen oldalakra van szükségünk?

A program természetesen két részre tagolódik. Az első a tagok számára kialakított terület, amely a klubok információinak kezelésére, új események hozzáadására szolgál majd. A második a felhasználók területe, ahol az adatbázison lekérdezéseket kell majd végrehajtani.

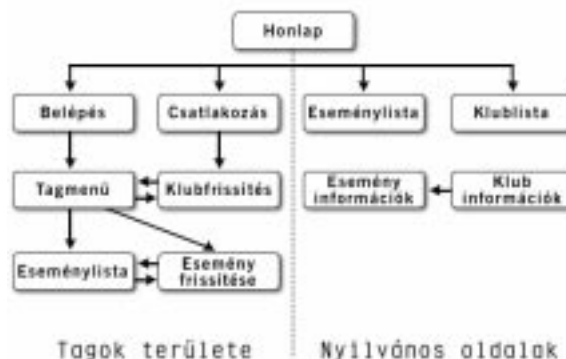


A továbbiakban tagoknak nevezzük azokat a személyeket, akik klubjukat bejegyezve felügyeleti feladatokat látnak el, és felhasználóknak azokat az érdeklődőket, akik a listákat böngészve barangolnak a klubok és események között.

A 23.1. ábra az alkalmazás felépítését mutatja.

23.1. ábra

Az alkalmazás felépítése



Az új tagok a `csatlakozas.php` oldalon csatlakozhatnak a rendszerhez (itt jegyeztethetik be magukat a tagok közé), egy név–jelszó pár megadásával. Ha a választott név még nem foglalt, a leendő tag a `klubfrissites.php` oldalra kerül, ahol egy űrlapon meg kell adnia a klubról a szükséges információkat. Amíg ki nem tölti ezt az űrlapot, nem vihet be új rendezvényeket a rendszerbe. Ha a tag a hozzá tartozó klub adatait sikeresen bevitte, a tagok menüjéhez kerül (`tagmenu.php`), ahonnan a tagok részére készített valamennyi oldal elérhető.

A már bejegyzett tagok a belépő oldalról indulnak (`belepes.php`). Ha a megadott név és jelszó helyesnek bizonyult, egyenesen a `tagmenu.php` oldalra kerülnek.

A menü oldalról indulva a tagok új rendezvényeket adhatnak a rendszerhez (`esemenyfrissites.php`) és megtekinthetik az adatbázisban levő rendezvények listáját (`esemenylista.php`). A klub adatait a `klubfrissites.php` oldalon bármikor módosíthatják.

Minden felhasználó képes lesz az események hónapok, napok, típusok és területek alapján történő rendezett kiíratására, egyetlen PHP oldal segítségével (`esemenyekinfo.php`). Lehetőség lesz a klubok felsoroltatására is, terület vagy típus alapján (`klubokinfo.php`). Végül a felhasználók egy klubra vagy eseményre kattintva bővebb információkat tudhatnak meg (`esemenyinfo.php`, `klubinfo.php`).

Az adatbázis kialakítása

Az alkalmazás adatainak tárolásához létre kell hoznunk a `szervezo` adatbázist és ebben négy táblát: `klubok`, `esemenyek`, `teruletek`, `tipusok`. Nyilvánvaló, hogy a klubok és rendezvények adatait külön táblákban kell tárolnunk, hiszen egy klubhoz több rendezvény is tartozhat. A területek és típusok számára kialakított táblák jelentősen megkönnyítik a listázáshoz és adatbevitelhez szükséges lenyíló menük kialakítását. A táblákat SQL parancsokkal hozzuk létre, „kézi úton”. A klubok tábla a következőképpen hozható létre:

```
CREATE TABLE klubok (  
    azonosito INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    klubnev VARCHAR(50),  
    tipus CHAR(3),  
    terulet CHAR(3),  
    email VARCHAR(50),  
    ismerteto BLOB,  
    nev VARCHAR(8),  
    jelszo VARCHAR(8)  
);
```

A tagok a klubnev, email, ismerteto, nev és jelszo mezők adatait fogják megadni. A típus és terület mezők értelemszerűen a típusok és területek megfelelő soraira vonatkozó azonosítókat tartalmazzák majd.

Az események tábla az eseményekre vonatkozó valamennyi információt tartalmazza:

```
CREATE TABLE esemenyek (
    eazonosito INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    etipus CHAR(3),
    eterulet CHAR(3),
    edatum INT,
    enev VARCHAR(100),
    ehelyszin VARCHAR(100),
    ecim VARCHAR(255),
    eirsz CHAR(4),
    eismerteto BLOB,
    eklub INT NOT NULL
);
```

Vegyük észre, hogy ez a tábla is tartalmaz etipus és eterulet mezőket, hiszen előfordulhat, hogy egy klub a város északi részén található, de valamelyik rendezvényét a déli városrészben tartja. Egy társadalmi csoport tarthat oktatási szemináriumot, de politikai találkozót is. Az eklub mező annak a klubnak az azonosítószámát tartalmazza, amelyik az adott esemény szervezője. A kapcsolat arra használható fel, hogy egy klub összes rendezvényét felsoroljuk, illetve hogy elérjük az eseményhez köthető klub adatait.

A területek és típusok táblák igen egyszerűek:

```
CREATE TABLE teruletek ( azonosito CHAR(3),
    ➡ terület VARCHAR(30));
CREATE TABLE tipusok ( azonosito CHAR(3),
    ➡ típus VARCHAR(30));
```

A tagok számára nem adunk lehetőséget ezen táblák módosítására, inkább előre meghatározott csoportokat alakítunk ki, melyeket INSERT SQL parancsokkal adunk a táblákhoz. A tagok ezekből a listákból választhatnak majd.

```
INSERT INTO tipusok (azonosito, típus) VALUES
    ➡ ("CSA", "Családi");
```

A 23.1. és 23.2. táblázatban az említett táblákba illesztendő adatok láthatók.

23.1. táblázat A területek táblához adott adatok

azonosito	terulet
ESZ	Észak
DEL	Dél
KEL	Kelet
NYU	Nyugat

23.2. táblázat A típusok táblához adott adatok

azonosito	tipus
ZEN	Zenei
CSA	Családi
TRS	Társadalmi
KZS	Közösségi

Tervezési döntésünk

Az előző bekezdésekben már láttuk, milyen szerkezetű alkalmazást készítünk. Úgy döntöttünk, hogy a különböző szolgáltatásokhoz különböző PHP programokat készítünk, ahelyett, hogy egyetlen hatalmas programot építenénk fel, amely a körülményeknek megfelelően más-más oldalakat szolgáltat. Ennek a döntésnek természetesen vannak előnyei és hátrányai is.

Ha egy ilyen dinamikus környezetet több oldalból építünk fel, a kódok ismétlésének hibájába eshetünk és a program növekedésével megnehezíthetjük a fejlesztők dolgát. Másrészt viszont a befejezett prototípust átadhatjuk grafikusainknak, akik majdnem úgy kezelhetik azt, mintha pusztán HTML kódot tartalmazna.

A tagoknak szánt oldalak

Most már itt az ideje, hogy megkezdjük a kód írását. Az óra további részében az alkalmazás tagoknak szánt oldalait készítjük el. Jó ötlet ezekkel kezdeni, mivel így könnyebben vihetünk be próbaadatokat a rendszerbe. Mielőtt azonban az adatbevitelt megkezdhetnénk, képesnek kell lennünk a tagok felvételére.

csatlakozas.php és adatbazis.inc

A `csatlakozas.php` tartalmazza azt az űrlapot, amelyen keresztül az új tagok egy név és jelszó segítségével bejegyeztethetik klubjukat a rendszerbe. Ahhoz, hogy felvehessük az adatokat és kiszűrjessük az ismétlődéseket, először meg kell nyitnunk egy kapcsolatot az adatbázis felé. Ez az ilyen alkalmazásoknak annyira jellemző eleme, hogy célszerű külön függvényt készíteni erre a célra, melyet jelen esetben egy külső állományban fogunk tárolni. Ezt a külső fájlt később minden oldalon az `include()` nyelvi elemmel illesztjük a kódba. Tulajdonképpen minden adatbázissal kapcsolatos függvényt ebben tárolunk majd, ezért az `adatbazis.inc` nevet kapja, szerepe és a beillesztés módja után. Ezzel a fájllal a további PHP oldalakat mindennemű SQL parancstól mentesítjük.

Az adatbázissal kapcsolatot tartó kódok külön fájlba helyezése könnyebbé teszi a program későbbi módosítását vagy más adatbázismotorra való átültetését. Előfordulhat, hogy később újra kell írunk a függvényeket, de a hívó kódok módosítására nem lesz szükség.

Hozzuk létre a függvényt, amely végrehajtja a csatlakozást:

23.1. program Részlet az adatbazis.inc fájlból

```
1: $kapcsolat;  
2: dbCsatlakozas();  
3: function dbCsatlakozas()  
4:     {  
5:         global $kapcsolat;  
6:         $kapcsolat = mysql_connect( "localhost",  
                                     "felhasznalo", "jelszo" );  
7:         if ( ! $kapcsolat )  
8:             die( "Nem lehet csatlakozni a MySQL-hez" );  
9:         mysql_select_db( "szervezo", $kapcsolat )  
10:        or die ( "Nem lehet megnyitni az adatbázist:  
                ".mysql_error() );  
11:    }
```

A `dbCsatlakozas()` függvény a `$kapcsolat` globális változót használja arra, hogy a `mysql_connect()` által visszaadott kapcsolatazonosítót tárolja. Mivel a változó globális, a többi adatbázisfüggvény is elérheti. Nem csupán a MySQL kiszolgálóhoz csatlakozunk ebben a függvényben, hanem megpróbáljuk kiválasztani a `szervezo` adatbázist is. Mivel ezen műveletek sikere döntő fontosságú a teljes alkalmazás működése szempontjából, a `mysql_connect()` vagy `mysql_select_db()` végrehajtásának kudarca esetén befejezzük a program futását.

A munkamenetek követése és az azonosítással kapcsolatos feladatokat ellátó függvények számára egy újabb külső állományt használunk. A munkamenet-függvényeket arra használjuk, hogy egy `$munkamenet` nevű asszociatív tömböt őrizzünk meg kérésről kérésre. A `kozsofv.inc` állomány `session_start()` függvénye arra szolgál, hogy megkezdjünk vagy folytassunk egy munkamenetet, a változónkat pedig a `session_register()` függvénnyel rendeljük ehhez:

23.2. program Részlet a kozsofv.inc fájlból

```
1: session_start();  
2: session_register( "munkamenet" );
```

Emlékezzünk rá, hogy minden PHP kód, amit ezen külső fájlokba írunk, PHP blokk kezdő (`<?php`) és záró (`?>`) elemek között kell, hogy legyen. Értelmetlen bonyolításnak tűnhet, hogy a különböző szolgáltatásokat külső állományokba helyezzük, de ez rengeteg kódismétléstől kímél majd meg bennünket, amikor oldalainkat elkészítjük. Most már készen állunk a `csatlakozas.php` oldal megírására.

23.3. program csatlakozas.php

```
1: <?php  
2:     include("adatbazis.inc");  
3:     include("kozsofv.inc");  
4:  
5:     $uzenet="";  
6:  
7:     if ( isset( $mitkelltenni ) &&  
8:         $mitkelltenni=="csatlakozas")  
9:     {  
10:         if ( empty( $urlap["nev"] ) ||  
11:             empty( $urlap["jelszo"] ) ||  
12:             empty( $urlap["jelszo2"] ) )  
13:             $uzenet .= "Ki kell töltenie minden  
14:                 mezőt!<br>\n";  
15:  
16:         if ( $urlap["jelszo"] != $urlap["jelszo2"] )  
17:             $uzenet .= "A jelszavak nem  
18:                 egyeznek!<br>\n";
```

23.3. program (folytatás)

```
17:         if ( strlen( $urlap["jelszo"] ) > 8 )
18:             $uzenet .= "A jelszó hossza legfeljebb
19:                         8 karakter lehet!<br>\n";
20:
21:         if ( strlen( $urlap["nev"] ) > 8 )
22:             $uzenet .= "A tagsági név hossza
23:                         legfeljebb
24:                         8 karakter lehet!<br>\n";
25:         if ( sorLekeres( "klubok", "nev",
26:                         $urlap["nev"] ) )
27:             $uzenet .= "A megadott tagsági néven.\"\"
28:                         $urlap["nev"] .\"\"
29:                         már van bejegyzett tagunk.
30:                         Kérjük, adjon meg más
31:                         nevet!<br>\n";
32:
33:         if ( $uzenet == "" ) // nem találtunk hibát
34:             {
35:                 $azon = ujTag( $urlap["nev"],
36:                               $urlap["jelszo"] );
37:                 munkamenetFeltoltes( $azon, $urlap["nev"],
38:                                     $urlap["jelszo"] );
39:
40:                 header( "Location:
41:                         klubfrissites.php?".SID );
42:                 exit;
43:             }
44:
45:     ?>
46:
47: <html>
48: <head>
49: <title>Csatlakozás</title>
50: </head>
51:
52: <body>
53: <?php
54: include("kozosnav.inc");
55: ?>
56: <p>
57: <h1>Csatlakozás</h1>
```

23.3. program (folytatás)

```
52:
53:     <?php
54:     if ( $uzenet != " " )
55:     {
56:         print "<b>$uzenet</b><p>";
57:     }
58:     ?>
59:
60:     <p>
61:     <form action="<?php print $PHP_SELF;?>"
62:     <input type="hidden" name="mitkelltenni"
        value="csatlakozas">
63:     <input type="hidden" name="<?php print
        session_name() ?>"
64:         value="<?php print session_id() ?>">
65:     Tagsági név: <br>
66:     <input type="text" name="urlap[nev]"
67:         value="<?php print $urlap["nev"] ?>"
        maxlength=8>
68:     </p>
69:     <p>
70:     Jelszó: <br>
71:     <input type="password" name="urlap[jelszo]"
        value="" maxlength=8>
72:     </p>
73:     <p>
74:     Jelszó megerősítés: <br>
75:     <input type="password" name="urlap[jelszo2]"
        value="" maxlength=8>
76:     </p>
77:     <p>
78:     <input type="submit" value="Rendben">
79:     </p>
80:     </form>
81:
82:     </body>
83:     </html>
```

Először az `include()` használatával beillesztjük a külső állományokat, tehát azonnal rendelkezésünkre áll egy adatbáziskapcsolat és egy aktív munkamenet. Létrehozunk egy `$uzenet` nevű változót. Ezzel a kettős célú változóval még

számos más oldalon találkozni fogunk. Az \$uzenet célja egyrészt, hogy tartalmazza a hibaüzenetet, amit később kiírhatunk a böngésző számára, másrészt használhatjuk feltételes kifejezésekben, hogy ellenőrizzük, volt-e hibaüzenet vagy sem. Ha a változó üres marad, feltételezhetjük, hogy minden ellenőrzés sikeresen végrehajtódott.

Ezután a \$mitkelltenni változó létezését és tartalmát vizsgáljuk. Ez is olyan elem, amely többször felbukkan majd az alkalmazásban. Minden általunk készített űrlapon beállítunk egy mitkelltenni nevű rejtett elemet és egy, az adott szolgáltatásra vonatkozó értéket adunk neki. Ha PHP programunkban a \$mitkelltenni változó létezik és a várt érték található benne, biztosak lehetünk abban, hogy az űrlapot kitöltötte valaki, így folytathatjuk a programot az űrlap adatainak ellenőrzésével. Ha a változó nem létezik, tudhatjuk, hogy a látogató egy hivatkozáson keresztül vagy a böngészőjében lévő könyvjelző révén érkezett az oldalra és nem töltötte ki az űrlapot.

Egyelőre ugorjunk a HTML rész tárgyalására. A body HTML elemen belül először egy újabb külső fájlt illesztünk be, amely a különböző oldalakra mutató hivatkozásokat tartalmazza. Célszerű már a kezdetektől beilleszteni a navigációs sávot, mivel ez megkönnyíti az alkalmazás ellenőrzését, ahogy folyamatosan fejlesztjük azt. A navigációs elemeket a kozosnav.inc nevű állományban helyezzük el. Egyelőre ez a fájl csak a látogatók számára is elérhető oldalakra mutató hivatkozásokat tartalmazza.

23.4. program Részlet a kozosnav.inc fájlból

```
1: <p>
2: <a href="klubokinfo.php"?<?php print SID
   ?>">Klubinformációk</a> |
3: <a href="esemenyekinfo.php"?<?php print SID
   ?>">Eseményinformációk</a> |
4: <a href="csatlakozas.php"?<?php print SID
   ?>">Csatlakozás</a> |
5: <a href="belepes.php"?<?php print SID ?>">Belépés</a> |
6: <a href="index.php"?<?php print SID ?>">Honlap</a>
7: </p>
```

Azzal, hogy a navigációs elemeket külön állományba helyezzük, egyetlen mozdulattal módosíthatjuk a hivatkozásokat, illetve az elemek kinézetét a teljes alkalmazásra vonatkozóan.

Visszatérve a `csatlakozas.php` oldalra, miután kiírjuk az oldal címsorát, ellenőrizzük az `$uzenet` változót. Ha nem üres, kiírjuk a tartalmát a böngésző számára. Az összegyűjtött hibaüzenetek kiírásával jelezhetjük a tagnak, hogy a bevitt adatok nem dolgozhatók fel.

A HTML űrlap elemei között három látható mezőt hoztunk létre, az `urlap[nev]`, `urlap[jelszo]` és `urlap[jelszo2]` mezőket. Azért használjuk ezeket az elsőre esetleg furcsának látszó elnevezéseket, mert a PHP az így megadott nevekkal érkező értékeket egy asszociatív tömbbe rendezi, amit `$urlap` néven érhetünk majd el. Ez megóv bennünket attól, hogy a globális változók környezetét „beszennyezzük”, azaz feleslegesen sok globális változót hozzunk létre. Programjainkban így minden munkamenet-érték a `$munkamenet` asszociatív tömbben, az űrlapértékek pedig az `$urlap` asszociatív tömbben érhetők el. Ez megvédi bennünket a változók ütközésétől. Célszerű a lehető legkevesebbre csökkenteni a globális változók számát egy ilyen méretű programban, hogy megelőzzük a kavarodást. A továbbiakban minden űrlapot tartalmazó oldalon az `$urlap` tömbbel fogunk találkozni.

Az űrlapban létrehozuk a már korábban említett `mitkelltenni` rejtett elemet is, valamint egy másikat, amely a munkamenet nevét és értékét tartalmazza majd. Az alkalmazás elkészítése során mindig oldalról-oldalra fogjuk adni a munkamenet-azonosítót, mivel nem kívánjuk elveszteni azokat a látogatókat, akik nem tudnak vagy nem akarnak sütitet fogadni.



Ha munkameneteket használó programokat ellenőrzünk, célszerű kikapcsolni a böngészőben a süтик elfogadását. Így pontosan tudni fogjuk, hogy a munkamenet azonosítóját sikeresen át tudjuk-e adni oldalról oldalra ezen szolgáltatás nélkül is vagy sem.

Most, hogy megnéztük a HTML űrlapot, rátérhetünk arra a kódra, amely a bemenő adatok ellenőrzését végzi. Először meg kell bizonyosodnunk róla, hogy a `leendő` tag valóban kitöltötte-e az összes mezőt és hogy egyik mező hossza sem haladja meg a nyolc karaktert. Ezután meghívjuk az új `sorLekeres()` függvényt. Ez azon függvények egyike, amelyek az `adatbazis.inc` fájlban találhatók. Első paramétere egy táblanév, a második egy mezőnév, a harmadik egy érték. Ezen adatok alapján a függvény egy illeszkedő sort keres az adatbázisban, visszaadva azt egy tömbben.

23.5. program Részlet az adatbazis.inc fájlból

```
1: function sorLekeres( $tabla, $mezonev, $mezoertek )
2:     {
3:         global $kapcsolat;
4:         $eredmeny = mysql_query( "SELECT * FROM $tabla
                                   WHERE $mezonev='$mezoertek'",
                                   $kapcsolat );
5:         if ( ! $eredmeny )
6:             die ( "sorLekeres hiba: ".mysql_error() );
7:         return mysql_fetch_array( $eredmeny );
8:     }
```

A függvénynek a klubok táblanevet, a nev mezőnevet és a látogatók által bevitt \$urlap["nev"] mezőértéket adjuk át. Ha a `mysql_fetch_array()` nem üres tömböt ad vissza, tudhatjuk, hogy egy ilyen belépési névvel rendelkező tag már van a rendszerben, ezért hibaüzenetet kell adnunk.

Ha az ellenőrzések után az `$uzenet` változó még mindig üres, létrehozhatjuk az új tagot a bevitt adatokkal. Ez két lépésből tevődik össze. Először is frissíteni kell az adatbázist. Ehhez egy új függvény tartozik az `adatbazis.inc` állományból, `ujTag()` néven:

23.6. program Részlet az adatbazis.inc fájlból

```
1: function ujTag( $nev, $jelszo )
2:     {
3:         global $kapcsolat;
4:         $eredmeny = mysql_query( "INSERT INTO klubok
                                   (nev, jelszo)
5:                                   VALUES('$nev', '$jelszo')",
                                   $kapcsolat);
6:         return mysql_insert_id( $kapcsolat );
7:     }
```

A függvény egy név és egy jelszó paramétert vár, majd ezeket új sorként felveszi a klubok táblájába. A függvény a `mysql_insert_id()`-t használja, hogy megkapja a felvett új tag azonosítóját.

Mivel már rendelkezünk az új tag azonosítójával, meghívhatjuk a felvételéhez szükséges újabb függvényt, ami a `kozofsv.inc` fájlban található. A `munkamenetFeltoltes()` függvény egy azonosítót, egy belépési nevet és egy jelszót vár paraméterül és hozzáadja azokat a `$munkamenet` tömbhöz. Így ezek az értékek már elérhetőek lesznek minden munkamenetet kezelő oldalunk számára. Ezzel képesek leszünk a tag azonosítására a további oldalakon is.

23.7. program Részlet a kozofsv.inc fájlból

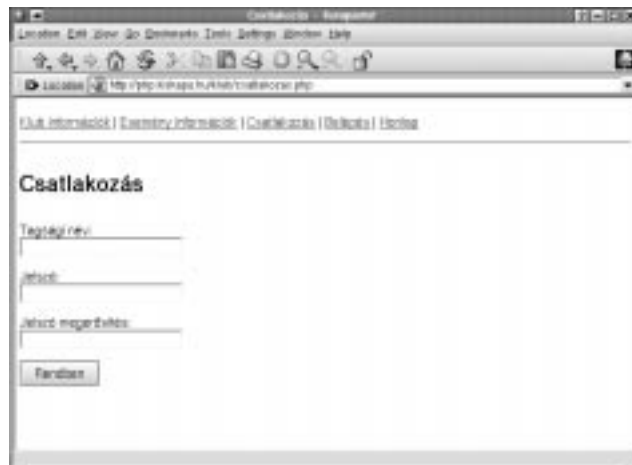
```
1: function munkamenetFeltoltes( $azonosito, $nev,
                                $jelszo )
2:     {
3:         global $munkamenet;
4:         $munkamenet["azonosito"] = $azonosito;
5:         $munkamenet["nev"] = $nev;
6:         $munkamenet["jelszo"] = $jelszo;
7:         $munkamenet["belepett"] = true;
8:     }
```

Az azonosító, név és jelszó megadása mellett a belépési állapotot jelző értéket (`belepett`) is beállítjuk a munkameneteket kezelő oldalaink számára. Végül, miután a `csatlakozas.php` oldal frissítette az adatbázist és a munkamenethez rendelt értékeket, útjára engedhetjük új tagunkat. Meghívjuk a `header()` függvényt, átirányítva a böngészőt a `klubfrissites.php` oldalra, ahol a tagnak be kell majd állítania az újonnan bejegyzett klub részletes adatait.

A `csatlakozas.php` kimenetét a 23.2. ábrán láthatjuk.

23.2. ábra

A csatlakozas.php kimenete



klubfrissites.php

Ez az oldal kettős célt szolgál. Egyrészt az új tagoknak itt kell megadniuk a klub részletes adatait, a bejegyzett tagok pedig itt módosíthatják ezeket.

23.8. program klubfrissites.php

```
1: <?php
2:     include("adatbazis.inc");
3:     include(" kozosfv.inc");
4:
5:     $klub_sor = azonositas();
6:
7:     $uzenet = "";
8:
9:     if ( isset( $mitkelltenni ) &&
        $mitkelltenni=="frissites" )
10:    {
11:        if ( empty( $urlap["klubnev"] ) )
12:            $uzenet .="A klubnév mező nincs
                kitöltve!<br>\n";
13:
14:        if ( ! sorLekeres( "teruletek", "azonosito",
            $urlap["terulet"] ) )
15:            $uzenet .="KRITIKUS HIBA: A terület kód-
                ja nem található!<br>";
16:
17:        if ( ! sorLekeres( "tipusok", "azonosito",
            $urlap["tipus"] ) )
18:            $uzenet .="KRITIKUS HIBA: A típus kódja
                nem található!<br>";
19:
20:        if ( $uzenet == "" )
21:        {
22:            klubFrissites( $munkamenet["azonosito"],
                $urlap["klubnev"],
                $urlap["terulet"],
23:                $urlap["tipus"],
                $urlap["email"],
                $urlap["ismerteto"] );
24:            header("Location: tagmenu.php?".SID);
25:            exit;
26:        }
27:    }
```


23.8. program (folytatás)

```
28:     else
29:     {
30:         $urlap = $klub_sor;
31:     }
32:     ?>
33:
34:     <html>
35:     <head>
36:     <title>Klubinformációk frissítése</title>
37:     </head>
38:
39:     <body>
40:     <?php
41:     include("kozosnav.inc");
42:     ?>
43:     <h1>Klubinformációk frissítése</h1>
44:     <?php
45:     if ( $uzenet != " " )
46:     {
47:         print "<b>$uzenet</b><p>";
48:     }
49:     ?>
50:
51:     <form action="<?php print $PHP_SELF;?>">
52:     <input type="hidden" name="mitkelltenni"
53:         value="frissites">
54:     <input type="hidden" name="<?php print
55:         session_name() ?>"
56:         value="<?php print session_id() ?>">
57:     <p>
58:     Klub neve: <br>
59:     <input type="text" name="urlap[klubnev]"
60:         value="<?php print
61:             stripslashes($urlap["klubnev"]) ?>">
62:     </p>
63:     Klub helye: <br>
64:     <select name="urlap[terulet]">
65:     <?php optionLista( "teruletek", $urlap["terulet"] ) ?>
```

23.8. program (folytatás)

```
65:      </select>
66:      </p>
67:      <p>
68:      Klub típusa: <br>
69:      <select name="urlap[tipus]">
70:      <?php optionLista( "tipusok", $urlap["tipus"])?>
71:      </select>
72:      </p>
73:      <p>
74:      Kapcsolattartó e-mail címe: <br>
75:      <input type="text" name="urlap[email]"
76:      value="<?php print
      stripslashes($urlap["email"]) ?>">
77:      </p>
78:      <p>
79:      Klub ismertetője: <br>
80:      <textarea name="urlap[ismerteto]" rows=5 cols=30
      wrap="virtual">
81:      <?php print stripslashes($urlap["ismerteto"]) ?>
82:      </textarea>
83:      </p>
84:      <p>
85:      <input type="submit" value="Rendben">
86:      </p>
87:      </form>
88:
89:      </body>
90:      </html>
```

Beillesztjük az `adatbazis.inc` és `kozosfv.inc` nevű külső állományainkat, így rögtön kész adatbáziskapcsolattal és munkamenettel kezdjük az oldalt. Ezután meghívjuk az új `azonositas()` nevű függvényt, ami a `kozosfv.inc` fájlban található. Ez összehasonlítja a munkamenet adatait az adatbázissal.

23.9. program Részlet a `kozosfv.inc` fájlból

```
1: function azonositas( )
2:     {
3:         global $munkamenet, $belepett;
4:         $munkamenet["belepett"] = false;
```

23.9. program (folytatás)

```
5:      $klub_sor = sorLekeres( "klubok", "azonosito",
                                $munkamenet["azonosito"] );
6:      if ( ! $klub_sor ||
7:          $klub_sor["nev"] != $munkamenet["nev"] ||
8:          $klub_sor["jelszo"] != $munkamenet["jelszo"] )
9:      {
10:         header( "Location: belepes.php" );
11:         exit;
12:      }
13:      $munkamenet["belepett"] = true;
14:      return $klub_sor;
15:  }
```

Az `azonositas()` jól átlátható függvény. A `$munkamenet["azonosito"]` elemét használjuk arra, hogy a `sorLekeres()` függvény segítségével lekérjük a klubhoz tartozó sort az adatbázisból. Ezt a `$klub_sor` asszociatív tömbben tároljuk és ellenőrizzük a `nev` és `jelszo` elemek egyezését a `$munkamenet` megfelelő tömb-elemeivel. Ha nem egyeznek, a `belepes.php` oldalra küldjük a tagot.

Miért vállaljuk fel az adatbázis lekérdezésének költségét az azonosításhoz? Miért nem egyszerűsítjük a problémát a `belepett` elem ellenőrzésére? Ez visszaéléseknek engedne teret, mivel egy rosszindulatú látogató egyszerűen hozzáadhatna az oldal címéhez egy ilyen részt:

```
munkamenet%5Bbelepett%5D=1&munkamenet%5Bazonosito%5D=1
```

Így átvéve az 1-es azonosítóval rendelkező tag fölött az irányítást, a PHP ugyanis ezt a `$munkamenet` tömbbé alakítja, amely egy felületesebb ellenőrzésen túlmutathatna. A rosszindulatú látogató most is alkalmazhat egy hasonló trükköt, hogy belépést nyerjen, de mivel az adatbázisban lévő névvel és jelszóval történő egyezést vizsgáljuk, a „hamisított” `$munkamenet` változónak helyes tartalommal kell rendelkeznie, így pedig felesleges a csalás.

Az `azonositas()` függvény mellékterméke a visszaadott tömb, amely a kérdéses klubról az adatbázisban található összes adatot tartalmazza. Később az oldalak ezt az információt saját céljaikra használhatják fel. Miután az `azonositas()` függvény visszatér a klub adatait tartalmazó tömbbel, ezt a `$klub_sor` változónak adjuk értékül.

Ismételten ugorjunk a HTML részre. Itt először a `kozosnav.inc` állományt illesztjük be. Ebben a fájlban azonban továbblépünk az eddigieknél és szerepeltetjük a tagok menüjét is:

23.10. program kozosnav.inc

```
1: <p>
2: <a href="klubokinfo.php?<?php print SID ?>">
   Klubinformációk</a> |
3: <a href="esemenyekinfo.php?<?php print SID ?>">
   Eseményinformációk</a> |
4: <a href="csatlakozas.php?<?php print SID ?>">
   Csatlakozás</a> |
5: <a href="belepes.php?<?php print SID ?>">Belépés</a> |
6: <a href="index.php?<?php print SID ?>">Honlap</a>
7: </p>
8: <?php
9: if ( $munkamenet["belepett"] )
10: {
11:   ?>
12:   <p>
13:     <A HREF="klubfrissites.php?<?php print SID ?>">
       Klub részletei</A> |
14:     <A HREF="esemenylista.php?<?php print SID ?>">
       Bejegyzett események</A> |
15:     <A HREF="esemenyfrissites.php?<?php print SID ?>">
       Új esemény</A> |
16:     <A HREF="tagmenu.php?<?php print SID ?>">
       Tagok honlapja</A>
17:   </p>
18:   <?
19:   }
20: ?>
21: <hr>
```

Ha a `$munkamenet["belepett"]` igazra van állítva, a csak tagok számára megjelenítendő hivatkozásokat is kiírjuk. Vegyük észre, hogy minden hivatkozásban megadjuk a `SID` állandót. Ez tartalmazza a munkamenet nevét és azonosítóját, így biztosítjuk, hogy az azonosító oldalról-oldalra átvadódjon, még akkor is, ha a sütik nincsenek bekapcsolva.

A `klubfrissites.php` oldal űrlapja igazán figyelemreméltó. Az eddigieknek megfelelően a `mitkellteni` és a munkamenetet azonosító rejtett mezőket is felvesszük az űrlapba. Szövegmezőket biztosítunk a klub neve, ismertetője és a kapcsolattartó elektronikus levélcíme számára. A klubtípusok és területek lenyíló menüinek előállítására egy új függvény, az `optionLista()` szolgál.

23.11. program Részlet az adatbazis.inc fájlból

```
1: function optionLista( $tabla, $azon )
2:     {
3:         global $kapcsolat;
4:         $eredmeny = mysql_query( "SELECT * FROM $tabla",
                                   $kapcsolat );
5:         if ( ! $eredmeny )
6:             {
7:                 print "Nem lehet megnyitni: $tabla<p>";
8:                 return false;
9:             }
10:        while ( $egy_sor = mysql_fetch_row( $eredmeny ) ){
11:            print "<option value=\"\$egy_sor[0]\"";
12:            if ( $azon == $egy_sor[0] )
13:                print "SELECTED";
14:            print ">$egy_sor[1]\"n\"";
15:        }
16:    }
```

A függvény a `$tabla` paramétert használja, hogy kiválasszon minden sort a területek vagy típusok táblából. Ezután végiglépked minden soron az eredménytáblában, kiírva egy HTML `option` elemet a böngésző számára.

Ha a `mysql_fetch_array()` által visszaadott tömb első eleme egyezik az `$azon` paraméterben megadott azonosítóval, a `SELECTED` karakterláncot is kiírja. Ezzel biztosítjuk, hogy a megfelelő elem kerül alapbeállításban kiválasztásra az űrlapon.

A PHP kódban, ahol ellenőrizzük a bevitt adatokat, első feltételünk az `urlap["klubnev"]` kitöltöttségének tényét vizsgálja. Ezután az `urlap["terulet"]` és `urlap["típus"]` értékek helyességét ellenőrizzük.

Ha az `$uzenet` változó ezeket követően üres marad, tudjuk, hogy az adatok a meghatározott feltételeknek eleget tesznek, ezért átadjuk azokat egy új függvénynek, a `klubFrissites()`-nek, amely végrehajtja a szükséges módosítást az adatbázisban. A függvény egy klubazonosítót, egy klubnevet, egy területet, egy típust, egy e-mail címet és egy ismertetőt vár a klubok tábla számára.

23.12. program Részlet az adatbazis.inc fájlból

```

1: function klubFrissites( $azonosito, $klubnev,
                        $terulet, $tipus, $email,
                        $ismerteto )
2:     {
3:     global $kapcsolat;
4:     $lekeres = "UPDATE klubok set klubnev='$klubnev',
                    terulet='$terulet', tipus='$tipus',
5:     email='$email', ismerteto='$ismerteto' WHERE
                    azonosito='$azonosito'";
6:     $eredmeny = mysql_query( $lekeres, $kapcsolat );
7:     if ( ! $eredmeny )
8:         die ( "klubFrissites hiba: ".mysql_error() );
9:     }

```

Miután a sort frissítettük, a tagot a tagmenu.php oldalra küldhetjük.

Ha a tag az oldalt egy hivatkozásra kattintva vagy a böngészőjében lévő könyvjelzővel érte el, a \$mitkellteni változó nem kerül beállításra és az ellenőrzést és frissítést végző kód nem hajtódik végre. Még mindig van azonban egy feladatunk. A \$klub_sor asszociatív tömb tartalmazza a klub adatait, amit az azonositas() függvénnyel töltöttünk fel. Ebben találhatók az adott klubhoz tartozó mezőnevek és értékek az adatbázisból. Ezt a tömböt adjuk értékül az \$urlap tömbnek, így biztosítva, hogy az űrlap alapbeállításban az adatbázisban található érvényes információkat tartalmazza.

A 23.3. ábra a klubfrissites.php kimenetét mutatja.

23.3. ábra

A klubfrissites.php kimenete



tagmenu.php

A tagmenu.php a tagok területének központja. Alapvetően hivatkozások listájából áll, de a tagok számára fontos hírek és ajánlatok is helyet foglalhatnak az oldalon.

23.13. program tagmenu.php

```
1: <?php
2:     include("adatbazis.inc");
3:     include("kozosfv.inc");
4:
5:     $klub_sor = azonositas();
6:     klubAdatEllenorzes( $klub_sor );
7:     ?>
8:
9:     <html>
10:    <head>
11:    <title>Üdvözllet!</title>
12:    </head>
13:
14:    <body>
15:    <?php
16:    include("kozosnav.inc");
17:    ?>
18:
19:    <h1>Tagok honlapja</h1>
20:
21:    <a href="klubfrissites.php?<?php print SID ?>">
      Klub részletei</a><br>
22:    <a href="esemenylista.php?<?php print SID ?>">
      Bejegyzett események</a><br>
23:    <a href="esemenyfrissites.php?<?php print SID ?>">
      Új esemény</a><br>
24:
25:    </body>
26:    </html>
```

Az oldalon egyetlen újdonság található. Miután meghívjuk az `azonositas()` függvényt, hogy meggyőződjünk róla, hogy a látogatónk tag, az általa visszaadott tömböt egy új függvénynek adjuk át. Ez a függvény, a `klubAdatEllenorzes()`, a `kozosfv.inc` állományban található és azt ellenőrzi, hogy a tag megadta-e már a klub adatait. Amíg egy tag ki nem tölti legalább a klub nevét, nem adhat rendezvényeket a rendszerhez.

23.14. program Részlet a kozosfv.inc fájlból

```
1: function klubAdatEllenorzes( $klubtomb )
2:     {
3:         if ( ! isset( $klubtomb["klubnev"] ) )
4:             {
5:                 header( "Location: klubfrissites.php?".SID );
6:                 exit;
7:             }
8:     }
```

belepes.php

Mielőtt továbblépnünk a rendezvényeket kezelő oldalakra, el kell készítenünk a belepes.php programot. Ez az oldal ad lehetőséget a bejegyzett tagoknak, hogy a későbbiekben belépjenek a rendszerbe.

23.15. program belepes.php

```
1: <?php
2:     include("adatbazis.inc");
3:     include("kozosfv.inc");
4:
5:     $uzenet="";
6:
7:     if ( isset( $mitkelltenni ) && $mitkelltenni ==
        "belepes" )
8:     {
9:         if ( empty( $urlap["nev"] ) || empty(
            $urlap["jelszo"] ) )
10:            $uzenet .= "Ki kell töltenie minden
                mezőt!<br>\n";
11:
12:         if ( ! ( $sor_tomb =
13:                 jelszoEllenorzes( $urlap["nev"],
                    $urlap["jelszo"] ) ) )
14:            $uzenet .= "Hibás név vagy jelszó, próbál-
                kozzon újra!<br>\n";
15:
```


23.15. program (folytatás)

```
16:         if ( $uzenet == " " ) // nem találtunk hibát
17:         {
18:             munkamenetFeltoltes( $sor_tomb["azonosito"],
                                   $sor_tomb["nev"],
19:                                   $sor_tomb["jelszo"] );
20:             header( "Location: tagmenu.php?".SID );
21:         }
22:     }
23: ?>
24:
25: <html>
26: <head>
27: <title>Belépés</title>
28: </head>
29:
30: <body>
31:
32: <?php
33: include("kozosnav.inc");
34: ?>
35:
36: <h1>Belépés</h1>
37:
38: <?php
39: if ( $uzenet != " " )
40:     {
41:         print "<p><b>$uzenet</b></P>";
42:     }
43: ?>
44:
45: <p>
46: <form action="<?php print $PHP_SELF;?>">
47: <input type="hidden" name="mitkelltenni"
         value="belepes">
48: <input type="hidden" name="<?php
         print session_name() ?>"
49:         value="<?php print session_id() ?>">
50: </p><p>
51: Tagsági név: <br>
52: <input type="text" name="urlap[nev]"
53:         value="<?php print $urlap["nev"] ?>">
```

23.15. program (folytatás)

```
54:      </p><p>
55:      Jelszó: <br>
56:      <input type="password" name="urlap[jelszo]"
          value="">
57:      </p><p>
58:      <input type="submit" value="Rendben">
59:      </form>
60:
61:      </body>
62:      </html>
```

Az oldal szerkezete már ismerős kell, hogy legyen. Az `adatbazis.inc` és `kozosfv.inc` állományokat használjuk, hogy adatbáziskapcsolatot létesítsünk és aktív munkamenettel rendelkezünk.

Ha a `$mitkellteni` változó be van állítva és a várt "belepes" értéket tartalmazza, ellenőrizzük az űrlapról érkezett adatokat. Az új `adatbazis.inc` függvényt használjuk arra, hogy az `urlap["nev"]` és `urlap["jelszo"]` értékeket vizsgáljuk.

23.16. program Részlet az adatbazis.inc fájlból

```
1: function jelszoEllenorzes( $nev, $jelszo )
2:     {
3:         global $kapcsolat;
4:         $eredmeny = mysql_query( "SELECT azonosito, nev,
                                   jelszo FROM klubok
5:                                   WHERE nev='$nev' and
                                   jelszo='$jelszo'",
6:                                   $kapcsolat );
7:         if ( ! $eredmeny )
8:             die ( "jelszoEllenorzes hiba: "
                   .mysql_error() );
9:         if ( mysql_num_rows( $eredmeny ) )
10:            return mysql_fetch_array( $eredmeny );
11:         return false;
12:     }
```

A `jelszoEllenorzes()` egy belépési nevet és egy jelszót vár. Ezek felhasználásával egy egyszerű `SELECT` lekérdezést küld az adatbázisnak a klubok táblájára vonatkozóan.

Visszatérve a `belepes.php` oldalra, ha nem találunk hibát, meghívjuk a `munkamenetFeltoltes()` függvényt, amely beállítja az `azonosito`, `nev`, `jelszo` és `belepett` elemeket a `$munkamenet` tömbben. Ezután a tagot átirányítjuk a `tagmenu.php` oldalra.

esemenyfrissites.php

Most, hogy a tagok már képesek csatlakozni és belépni a rendszerbe, valamint módosítani is tudják az adataikat, lehetőséget kell adnunk a rendezvények bevitelére és módosítására. A teljes `esemenyfrissites.php` oldal a 23.17. programban látható.

23.17. program esemenyfrissites.php

```
1: <?php
2:     include("adatbazis.inc");
3:     include("kozosfv.inc");
4:     include("datum.inc");
5:
6:     $klub_sor = azonositas();
7:     klubAdatEllenorzes( $klub_sor );
8:
9:     $edatum = time();
10:    $uzenet = "";
11:
12:    if ( ! empty( $eazonosito ) )
13:        $esemeny_sor = sorLekeres( "esemenyek",
                                   "eazonosito", $eazonosito );
14:    else
15:        $eazonosito = false;
16:
17:    if ( isset( $mitkelltenni ) &&
        $mitkelltenni=="esemenyFrissites" )
18:    {
19:        if ( empty( $urlap["enev"] ) )
20:            $uzenet .="Az eseménynek névvel kell
                                   rendelkeznie!<br>\n";
21:
```

23.17. program (folytatás)

```
22:         if ( ! sorLekeres( "teruletek", "azonosito",  
                $urlap["eterulet"] ) )  
23:             $uzenet .= "KRITIKUS HIBA: A terület  
                        kódja nem található!<br>";  
24:  
25:         if ( ! sorLekeres( "tipusok", "azonosito",  
                $urlap["etipus"] ) )  
26:             $uzenet .= "KRITIKUS HIBA: A típus  
                        kódja nem található!<br>";  
27:  
28:         foreach ( array( "ehonap", "eev", "enap",  
                            "eperc" )  
29:                 as $datum_egyseg )  
30:             {  
31:                 if ( ! isset( $urlap[$datum_egyseg] ) )  
32:                     {  
33:                         $uzenet .= "KRITIKUS HIBA: A dátum  
                                    nem értelmezhető!";  
34:                         break;  
35:                     }  
36:             }  
37:         $datum = mktime( $urlap["eora"],  
                            $urlap["eperc"], 0,  
                            $urlap["ehonap"],  
38:                            $urlap["enap"], $urlap["eev"] );  
39:  
40:         if ( $datum < time() )  
41:             $uzenet .= "Múltbeli dátum nem  
                        fogadható el!";  
42:  
43:         if ( $uzenet == "" )  
44:             {  
45:                 esemenyModositas( $urlap["enev"],  
                                    $urlap["ehelyszin"],  
                                    $urlap["eterulet"],  
46:                                    $urlap["etipus"],  
                                    $urlap["ecim"], $urlap["eirsz"],  
47:                                    $urlap["eismerteto"],  
                                    $munkamenet["azonosito"],  
                                    $datum,  
48:                                    $eazonosito );  
49:                 header( "Location:  
                        esemenylista.php?".SID );  
50:             }
```

23.17. program (folytatás)

```
51:         }
52:     elseif ( $eazonosito )
53:     {
54:         //foreach( $esemeny_sor as $kulcs=>$ertek )
55:         //    $urlap[$kulcs] = $ertek;
56:         $urlap = $esemeny_sor;
57:         $edatum = $esemeny_sor["edatum"];
58:     }
59:     else
60:     {
61:         $urlap["eterulet"] = $klub_sor["terulet"];
62:         $urlap["etipus"] = $klub_sor["tipus"];
63:     }
64:     ?>
65:
66:     <html>
67:     <head>
68:     <title>Esemény hozzáadása/frissítése</title>
69:     </head>
70:     <body>
71:
72:     <?php
73:     include("kozosnav.inc");
74:     ?>
75:
76:     <h1>Esemény hozzáadása/frissítése</h1>
77:
78:     <?php
79:     if ( $suzenet != " " )
80:     {
81:         print "<b>$suzenet</b>";
82:     }
83:     ?>
84:     <p>
85:     <form action="<?php print $PHP_SELF;?>">
86:     <input type="hidden" name="mitkelltenni"
87:         value="esemenyFrissites">
88:     <input type="hidden" name="<?php
89:         print session_name() ?>"
90:         value="<?php print session_id() ?>">
```

23.17. program (folytatás)

```
89:      <input type="hidden" name="eazonosito"
90:          value="<?php print $eazonosito ?>">
91:      Esemény neve: <br>
92:      <input type="text" name="urlap[enev]"
93:          value="<?php print
          stripslashes($urlap["enev"]) ?>">
94:      </p>
95:      <p>
96:      Dátum és idő: <br>
97:      <select name="urlap[ehonap]">
98:      <?php honapLehetosegek( $edatum ) ?>
99:      </select>
100:
101:      <select name="urlap[enap]">
102:      <?php napLehetosegek( $edatum ) ?>
103:      </select>
104:
105:      <select name="urlap[eev]">
106:      <?php evLehetosegek( $edatum ) ?>
107:      </select>
108:
109:      <select name="urlap[eora]">
110:      <? oraLehetosegek( $edatum ) ?>
111:      </select>
112:
113:      <select name="urlap[eperc]">
114:      <? percLehetosegek( $edatum ) ?>
115:      </select>
116:      </p>
117:      <p>
118:      Esemény helye: <br>
119:      <select name="urlap[eterulet]">
120:      <?php optionLista( "teruletek",
          $urlap["eterulet"] ) ?>
121:      </select>
122:      </p>
123:      <p>
124:      Esemény típusa: <br>
125:      <select name="urlap[etipus]">
126:      <?php optionLista( "tipusok", $urlap["etipus"])?>
127:      </select>
```

23.17. program (folytatás)

```
128:    </p>
129:    <p>
130:    Esemény ismertetője: <br>
131:    <textarea name="urlap[eismerteto]" wrap="virtual"
132:              rows=5 cols=30>
133:    <?php print stripslashes($urlap["eismerteto"]) ?>
134:    </textarea>
135:    </p>
136:    <p>
137:    Helyszín: <br>
138:    <input type="text" name="urlap[ehelyszin]"
139:           value="<?php print
140:                 stripslashes($urlap["ehelyszin"])?>">
141:    </p>
142:    <p>
143:    Helyszín címe: <br>
144:    <textarea name="urlap[ecim]" wrap="virtual"
145:              rows=5 cols=30>
146:    <?php print stripslashes($urlap["ecim"]) ?>
147:    </textarea>
148:    </p>
149:    <p>
150:    Helyszín irányítószáma: <br>
151:    <input type="text" name="urlap[eirsz]"
152:           value="<?php print
153:                 stripslashes($urlap["eirsz"])?>">
154:    </p>
155:    <p>
156:    <input type="submit" value="Rendben">
157:    </p>
158:    </form>
159:    </body>
160:    </html>
```

Ezen az oldalon az események táblának megfelelő elemekkel készítünk egy űrlapot. Szokás szerint ellenőriznünk kell a beérkező értékeket és frissítenünk kell az adatbázist. Ha azonban az oldal meghívása alapján úgy ítéljük, hogy nem esemény hozzáadása a cél, hanem módosítás, akkor le kell kérdeznünk az adatbázisból a módosítandó adatokat. Az oldal meghívása során az `$seasonosito` változót kapja,

ha egy meglévő esemény módosítását kell elvégezni. Ha a változó nem üres, a `sorLekeres()` függvényt alkalmazva kapjuk meg az `$esemeny_sor` tömbben a rendezvény adatait. Ha az `$eazonosito` nincs megadva vagy üres, akkor hamis értékre állítjuk.

Ha az oldal űrlapkitöltés hatására hívódott meg, ellenőriznünk kell a beérkezett adatokat. A dátummal kapcsolatban meg kell vizsgálnunk, hogy az nem egy múltbeli érték-e. Ennek érdekében beállítunk egy `$edatum` nevű globális változót a beadott adatok függvényében.

Ha az adatok megfelelnek az általunk támasztott követelményeknek, az `adatbazis.inc` egy új függvényét hívjuk meg, melynek neve `esemenyModositas()`. Ez a függvény minden olyan értéket vár, amit az események táblában el kell helyeznünk. Az utolsó paraméter a rendezvény azonosítószáma. Ezt az `esemenyModositas()` függvény arra használja, hogy megállapítsa, új eseményt kell felvennie vagy módosítania kell egy már létezőt. Figyeljük meg, hogy a klub azonosítószáma a `$munkamenet["azonosito"]` változóban található, így ezt helyezzük az események tábla `eklub` mezőjébe. A dátum adatbázisban történő tárolására időbélyeget használunk.

23.18. program Részlet az `adatbazis.inc` fájlból

```

1: function esemenyModositas( $enev, $helyyszin,
    $eterulet, $etipus, $ecim, $eirs,
    $eismerteto, $eklub, $edatum, $eazonosito )
2:     {
3:         global $kapcsolat;
4:         if ( ! $eazonosito )
5:             {
6:                 $lekeres = "INSERT INTO esemenyek (enev,
                    ehelyyszin, eterulet, etipus,
7:                     ecim, eirs, eismerteto, eklub,
                        edatum )
8:                     VALUES( '$enev', '$helyyszin',
                                '$eterulet', '$etipus',
                                '$ecim',
9:                                '$eirs', '$eismerteto', '$eklub',
                                '$edatum')";
10:            }
11:         else
12:            {

```


23.18. program (folytatás)

```

13:             $lekeres = "UPDATE esemenyek SET enev='$enev',
                                ehelyszin='$ehelyszin',
14:             eterulet='$eterulet',
                                etipus='$etipus',
                                ecim='$ecim',
                                eirsz='$eirsz',

15: eismerteto='$eismerteto', eklub='$eklub', edatum='$edatum'
16:             WHERE eazonosito='$eazonosito'";
17:         }
18:         $eredmeny = mysql_query( $lekeres, $kapcsolat );
19:         if ( ! $eredmeny )
20:             die ( "esemenyModositas hiba: "
                    .mysql_error() );
21:     }

```

Látható, hogy a függvény az `$eazonosito` paraméter értékétől függően működik. Ha hamis értékű, egy INSERT SQL parancs hajtódik végre a megadott adatokkal. Egyéb esetben az `$eazonosito` egy UPDATE kérés feltételében szerepel. Miután az adatbázist frissítettük, a tagot az `esemenylista.php` oldalra irányítjuk, ahol a teljes rendezvénynaptárt láthatja.

Ha a tag még nem töltötte ki az űrlapot, átugorjuk az ellenőrző és adatbázis-frissítő kódokat. Ha azonban rendelkezünk az `$eazonosito` változóval, már lekértük az esemény információit az adatbázisból és az űrlapra írhatjuk azokat. Ezt úgy tehetjük meg, hogy az `$esemeny_sor` változót értékül adjuk az `$urlap` változónak, az `$edatum` változót pedig az `$esemeny_sor["edatum"]` értékével töltjük fel.

Ha nem kaptunk űrlapértékeket és `$eazonosito` változónk sincs, az `$urlap["terulet"]` és `$urlap["tipus"]` elemeknek a klubnak megfelelő adatokat adjuk értékül a `$klub_sor` tömbből. Ezzel a megfelelő lenyíló menükben a klubokhoz tartozó értékek lesznek alapbeállításban kiválasztva.

A HTML űrlapon az egyetlen említésre méltó kód a dátum és idő számára készített lenyíló menüket állítja elő. Ezeket a menüket elég egyszerűen beépíthettük volna a PHP kódba, de szükségünk van arra, hogy alapbeállításban az aktuális időt, a tag által korábban választott időt, vagy az `esemenyek` táblában tárolt időt jelezzék. Az alapbeállításban kiválasztandó értéket már elhelyeztük az `$edatum` változóban. Ezt a program elején az aktuális időre állítottuk be. Ha beérkező adatokat észlelünk, annak megfelelően állítjuk be ezt az értéket. Egyéb esetben, ha már meglévő eseményt módosítunk, az `esemenyek` tábla `edatum` mezőjének értékét adjuk az `$edatum` változónak.

Minden lenyíló menühöz az `$datum` változóban lévő időbélyeget adjuk át a megfelelő függvénynek. Ezek az új `datum.inc` nevű külső állományban találhatók.

23.19. program Részlet a `datum.inc` fájlból

```

1:  function honapLehetosegek( $datum )
2:      {
3:          $datum_tomb = getDate( $datum );
4:          $honapok = array( "Jan","Feb","Már",
5:                           "Ápr","Máj","Jún",
6:                           "Júl","Aug","Szep",
7:                           "Okt","Nov","Dec" );
8:          foreach ( $honapok as $kulcs=>$ertek )
9:              {
10:                 print "<OPTION
11:                     VALUE=\"\".($kulcs+1).\"\"";
12:                 print ( ( $datum_tomb["mon"] ==
13:                     ($kulcs+1) )?"SELECTED":"" );
14:                 print ">$ertek\n";
15:             }
16:     }
17:     function napLehetosegek( $datum )
18:     {
19:         $datum_tomb = getDate( $datum );
20:         for ( $x = 1; $x<=31; $x++ )
21:         {
22:             print "<OPTION VALUE=\"\"$x\"\"";
23:             print ( ( $datum_tomb["mday"] ==
24:                 $x )?"SELECTED":"" );
25:             print ">$x\n";
26:         }
27:     }
28:     function evLehetosegek( $datum )
29:     {
30:         $datum_tomb = getDate( $datum );
31:         $most_tomb = getDate(time());
32:         for ( $x = $most_tomb["year"];
33:             $x <= ($most_tomb["year"]+5); $x++ )
34:         {
35:             print "<OPTION VALUE=\"\"$x\"\"";

```

23.19. program (folytatás)

```

31:             print ( ( $datum_tomb["year"] ==
                      $x )?"SELECTED":"" );
32:             print ">$x\n";
33:         }
34:     }
35:
36:     function oraLehetosegek( $datum )
37:     {
38:         $datum_tomb = getDate( $datum );
39:         for ( $x = 0; $x< 24; $x++ )
40:         {
41:             print "<OPTION VALUE=\" $x\"";
42:             print ( ( $datum_tomb["hours"] ==
                      $x )?"SELECTED":"" );
43:             print ">".sprintf("%'02d", $x)."\n";
44:         }
45:     }
46:
47:     function percLehetosegek( $datum )
48:     {
49:         $datum_tomb = getDate( $datum );
50:         for ( $x = 0; $x<= 59; $x++ )
51:         {
52:             print "<OPTION VALUE=\" $x\"";
53:             print ( ( $datum_tomb["minutes"] ==
                      $x )?"SELECTED":"" );
54:             print ">".sprintf("%'02d", $x)."\n";
55:         }
56:     }

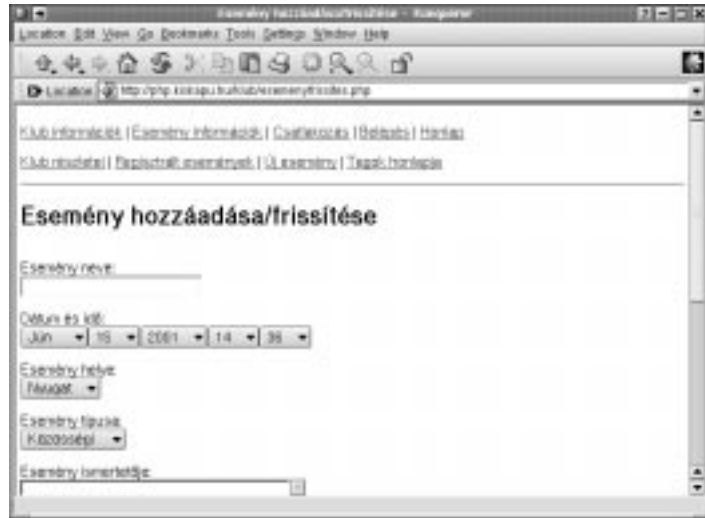
```

Minden függvény egy időbélyeget vár és HTML OPTION elemek listáját írja ki. A getDate() PHP függvényt használják arra, hogy az időbélyeg megfelelő elemének indexét megállapítsák (év, hónap, hónap napja, óra, perc). Ezután a kapott számot összehasonlítják az alkalmazott tartomány értékeivel (a tartomány a hónap napjainál 1-től 31-ig, a perceknél 0-tól 59-ig terjed). Ha egyezést észlelnek, a SELECTED jelzőt adják az OPTION elemhez, így választják ki a megfelelő sort.

A 23.4. ábra az esemenyfrissites.php kimenetét mutatja.

23.4. ábra

Az esemenyfrissites.php kimenete



esemenylista.php

Végül biztosítanunk kell a tagoknak egy oldalt, ahol minden általuk bevitt eseményt végigtekinthetnek. Ennek az oldalnak lehetőséget kell adnia arra is, hogy a tagok az eseményeket szerkeszthessék vagy törölhessenek egyet. Az esemenylista.php az eddigiek ismeretében egyszerű program.

23.20. program esemenylista.php

```

1: <?php
2:     include("adatbazis.inc");
3:     include("kozosfv.inc");
4:     $klub_sor = azonositas();
5:     klubAdatEllenorzes( $klub_sor );
6:
7:     function esemenyKiiras()
8:     {
9:         global $klub_sor;
10:        $esemenyek = esemenyekLekeres
11:                    ( $klub_sor["azonosito"] );
12:        if ( ! $esemenyek )
13:        {
14:            print "Nincs esemény a listában<P>";
15:            return;
16:        }

```

23.20. program (folytatás)

```

16:         print "<table border=1>\n";
17:         print "<td><b>Dátum</b></td>\n<td>
           <b>Név</b></td>\n
18:           <td><b>&nbsp;</b></td>\n";
19:         foreach ( $esemenyek as $sor )
20:         {
21:             print "<tr>\n";
22:             print "<td>".date("j M Y H.i",
           $sor["edatum"])."</td>\n";
23:             print "<td><a href=
           \"esemenyfrissites.php?eazonosito=\".
           $sor["eazonosito"]."\">".
24:               html($sor["enev"])."</a></td>\n";
25:             print "<td><a href=\"\".
           $GLOBALS["PHP_SELF"]."\"?eazonosito=\".
           $sor["eazonosito"];
26:             print "&mitkelltenni=
           esemenyTorles\".SID.\"\" ";
27:             print "onClick=\"return
           window.confirm('Biztos benne, hogy
           törli ezt az eseményt?')\">";
28:             print "törlés</a><br></td>\n";
29:             print "</tr>\n";
30:         }
31:         print "</table>\n";
32:     }
33:     $uzenet="";
34:
35:     if ( isset( $mitkelltenni ) &&
36:         $mitkelltenni == "esemenyTorles" &&
           isset( $eazonosito ) )
37:     {
38:         esemenyTorles( $eazonosito );
39:         $uzenet .= "Az esemény törlése
           megtörtént!<br>";
40:     }
41:
42:     ?>
43:     <html>
44:     <head>
45:     <title>Események listája</title>
46:     </head>

```

23.20. program (folytatás)

```
47:      <body>
48:      <?php
49:      include("kozosnav.inc");
50:      ?>
51:      <h1>Események listája</h1>
52:      <?php
53:      if ( $uzenet != " " )
54:      {
55:          print "<b>$uzenet</b>";
56:      }
57:      ?>
58:
59:      <?php
60:      esemenyKiiras();
61:      ?>
62:
63:      </body>
64:      </html>
```

Szokás szerint az `adatbazis.inc` fájlt használjuk, hogy kész adatbáziskapcsolattal kezdjük az oldalt és a `kozosfv.inc` fájlt, hogy aktív munkamenetünk legyen. Úgyszintén a megszokott módon ellenőrizzük, hogy érvényes tag kérte-e le az oldalt. Ezután létrehozunk egy új `esemenyKiiras()` nevű függvényt, amely közvetlenül a böngészőbe írja az eseményről elérhető információkat. Emlékeztünk rá, hogy az `azonositas()` függvény által visszaadott `$klub_sor` tömb tartalmazza a klubadatokat. A `$klub_sor["azonosito"]` értéket használhatjuk arra, hogy a klubhoz tartozó összes rendezvényt listába írjuk. Ehhez az `adatbazis.inc` `esemenyekLekeres()` függvényét használjuk. Ezt a függvényt alaposabban a következő órában tárgyaljuk, egyelőre elegendő annyit tudnunk, hogy egy klubazonosítót vár első paraméteréül, így egy kétdimenziós tömbben adja vissza a klubhoz tartozó összes esemény minden adatát.

A függvény visszatérési értékét az `$esemenyek` tömbben tároljuk. Valójában több adatot kérünk le, mint amennyire szükségünk van, de az `esemenyekLekeres()` függvény olyan rugalmas, hogy nyugodtan használhatjuk, hacsak nem érzékelünk teljesítménybeli problémákat a program futása során. Ha az `$esemenyek` hamis értékű, figyelmeztető üzenetet küldünk a kimenetre és befejezzük a függvény futását. Egyéb esetben egy HTML táblázatot építünk fel: végiglépkedve az `$esemenyek` tömbön kiírunk minden eseményre vonatkozóan néhány információt.

A `date()` PHP függvénynek átadva minden tömb edatum elemét, formázott dátumokat írunk ki. A dátum mellett minden esemény számára egy hivatkozást hozunk létre, felhasználva az esemény azonosítóját és az enev értéket. Az átadandó `eazonosito` változó mellett szerepeltetjük a `SID` állandót, így biztosítva, hogy a munkamenet érvényes marad az új oldalon is. A hivatkozás az `esemenyfrissites.php` oldalra mutat, amely így meg tudja jeleníteni az esemény adatait a szerkesztés számára. Végül minden rendezvény sorában egy erre az oldalra visszamutató hivatkozást is szerepeltetünk. Ehhez az `eazonosito` mellett egy `esemenyTorles` értékű `mitkellteni` paramétert is meghatározunk. Ez az adott esemény törlésére szolgál majd, ezért egy JavaScript eseménykezelőt is adunk a hivatkozáshoz, hogy ne lehessen véletlenül törölni egy rendezvényt sem.

Miután elkészítettük az `esemenyKiiras()` függvényt, kezelnünk kell azt az esetet is, amikor a tag törölni kíván egy eseményt. Ezt a `$mitkellteni` és az `$eazonosito` változók ellenőrzésével tehetjük meg. Ha rendben megkaptuk ezeket, egy új `adatbazis.inc` függvényt, az `esemenyTorles()`-t hívjuk meg, átadva az `$eazonosito` értéket.

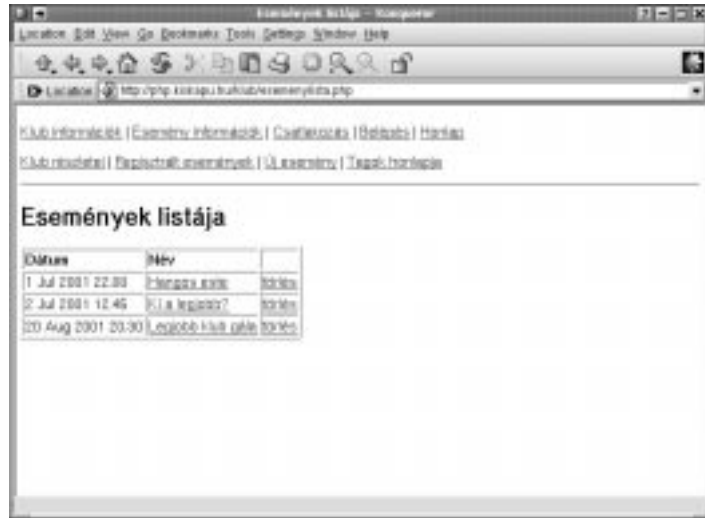
23.21. program Részlet az `adatbazis.inc` fájlból

```
1: function esemenyTorles( $eazonosito )
2:     {
3:         global $kapcsolat;
4:         $lekeres = "DELETE FROM esemenyek WHERE
                    eazonosito='$eazonosito'";
5:         $eredmeny = mysql_query( $lekeres, $kapcsolat );
6:         if ( ! $eredmeny )
7:             die ( "esemenyTorles hiba: ".mysql_error() );
8:         return ( mysql_affected_rows($kapcsolat) );
9:     }
```

Az `esemenylista.php` egy lehetséges kimenetét a 23.5. ábra mutatja.

23.5. ábra

*Az esemenylista.php
kimenete*



Összefoglalás

Az óra végére elkészültünk a teljesen működő, tagok számára készült környezettel a rendezvényeket nyilvántartó rendszerhez. Lehetőséget teremtettünk grafikusainknak az oldalak jó kialakítására, azzal, hogy a lehető legtöbb kódot külső állományokban helyeztük el. Gyakran használtuk a `header()` függvényt, hogy új oldalra irányítsuk a tagot, ha adatokat várunk tőle. A belépési információk tárolásához munkamenet-kezelő függvényeket, a tagok azonosításához adatbázis-lekérdezéseket használtunk minden oldalon.

A következő órában hasonló megoldásokat fogunk használni a program nyilvános felületének kialakításához.

Kérdések és válaszok

A több oldalból álló alkalmazások megtervezése és karbantartása bonyolultnak tűnik. Van valami titok emögött?

A példaalkalmazás elkészítése során majdnem minden oldalon egyszerű mintát követtünk. Ahogy alkalmazásaink nagyobbá válnak, a teljes környezetről egyetlen programként kell gondolkodnunk. A különálló oldalak csak a látogató csatlakozási pontjai az egész alkalmazáshoz. Tartsuk az összes egynél több oldalon ismétlődő kódot külön állományban!

Ha esetleg úgy döntünk, hogy egyetlen központi programot készítünk a teljes alkalmazás számára, egy egyszerű, alapvető HTML elemeket tartalmazó oldalt kell készítenünk. Minden más tartalom dinamikusan áll majd elő a program futása során. Szükségünk lesz egy `mitkelltenni` típusú változóra, amely mindig a megfelelő szolgáltatást választja ki a teljes alkalmazásból. Ez sokkal formásabbá teheti a kódot és segít elkerülni a nem túl elegáns `Location` fejléctrükköt. Másrészt ez azzal jár, hogy sokkal több HTML-t kell beépítenünk a PHP kódok közé.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik PHP függvényt használjuk arra, hogy MySQL adatbázishoz csatlakozzunk?
2. Melyik függvényt alkalmazzuk a munkamenet elkezdésére vagy folytatására?
3. Melyik függvény alkalmas arra, hogy külső állományokat illesszünk a PHP kódba?
4. Melyik PHP függvénnyel küldünk SQL kéréseket egy MySQL adatbázis számára?
5. Melyik állandót alkalmazhatjuk arra, hogy a munkafolyamat-azonosítót eljuttassuk egyik oldalról a másikra egy HTML hivatkozásban?
6. Hogyan küldhetjük tovább a látogatót egy másik oldalra?
7. Melyik függvény alkalmas dátumok formázására?

Feladatok

1. Tekintsünk végig az órában tárgyalt forráskódokon. Találunk olyan megoldásokat, amelyeket saját programjainkban is tudunk majd hasznosítani?



24. ÓRA

Teljes példa (második rész)

Az előző órában felépítettük a klubok számára a bejegyző és karbantartó felületet, amivel új klubok és rendezvények kerülhetnek a rendszerbe. Most az átlagos felhasználók számára elérhető programokat kell elkészítenünk, melyek lehetővé teszik a listák böngészését.

Ebben az órában a következőket tanuljuk meg:

- Hogyan készítsünk olyan függvényeket, amelyek egyetlen lekérdezéssel több táblából hívnak le adatokat?
- Hogyan készíthetünk a paramétereiktől függő SQL parancsokat végrehajtó függvényeket?
- Hogyan menthetjük a felhasználó régebbi beállításait munkamenet-függvényekkel?
- Hogyan kell átalakítanunk az egyszerű szöveges adatokat, hogy a HTML kódba írassuk azokat?

Az eseménynaptár nyilvános oldalai

Miután a tagok bejegyeztethetik klubjaikat és az eseményeket, itt az ideje, hogy az átlagos felhasználó számára is elérhető oldalak készítésébe fogjunk. Ezek az oldalak a listák böngészésére és nem a bennük való keresésre szolgálnak majd, bár nem lenne túl bonyolult ez utóbbit sem megvalósítani.

Ebben az órában négy oldalt készítünk el, amelyek lehetőséget adnak a felhasználónak, hogy kiírassa egy adott terület vagy téma rendezvényeit, bármilyen időpont szerinti szűréssel párosítva.

esemenyekinfo.php

Az `esemenyekinfo.php` oldal lehetőséget ad a rendszerben lévő rendezvények böngészésére. Bizonyos szempontból hasonlít az `esemenylista.php` oldalra, amit az előző órában tárgyaltunk, de nagyobb rugalmasságot és több információt nyújt a felhasználónak.

24.1. program `esemenyekinfo.php`

```
1: <?php
2:   include("adatbazis.inc");
3:   include("datum.inc");
4:   include("kozosfv.inc");
5:
6:   if ( isset($mitkellltenni) && $mitkellltenni ==
       "esemenyLista" )
7:       $munkamenet["esemenyek"] = $urlap;
8:   elseif ( $munkamenet["esemenyek"] )
9:       $urlap = $munkamenet["esemenyek"];
10:  else
11:  {
12:      $datum_tomb = getDate( time() );
13:      $munkamenet["esemenyek"]["terulet"] = "BÁRMELY";
14:      $munkamenet["esemenyek"]["tipus"] = "BÁRMELY";
15:      $munkamenet["esemenyek"]["honap"] = $datum_tomb["mon"];
16:      $munkamenet["esemenyek"]["ev"] = $datum_tomb["year"];
17:  }
18:
19:  $idoszak = datumIdoszak(
       $munkamenet["esemenyek"]["honap"],
20:      $munkamenet["esemenyek"]["ev"] );
```

24.1. program (folytatás)

```

21: function esemenyLista( )
22:     {
23:         global $idoszak, $munkamenet;
24:
25:         $esemenyek = esemenyekLekeres( 0, $idoszak,
                                         $munkamenet["esemenyek"]["terulet"],
26:                                         $munkamenet["esemenyek"]["tipus"] );
27:         if ( ! $esemenyek )
28:             {
29:                 print "Nincs megjeleníthető esemény<p>";
30:                 return;
31:             }
32:         print "<table border=1>\n";
33:         print "<td><b>Dátum</b></td>\n";
34:         print "<td><b>Esemény</b></td>\n";
35:         print "<td><b>Klub</b></td>\n";
36:         print "<td><b>Terület</b></td>\n";
37:         print "<td><b>Típus</b></td>\n";
38:         foreach ( $esemenyek as $sor )
39:             {
40:                 print "<tr>\n";
41:                 print "<td>".date("j M Y H.i",
                                     $sor["edatum"])."</td>\n";
42:                 print "<td><a
                                     href=\"esemenyinfo.php?eazonosito=\".
                                     $sor["eazonosito"]."\">".
43:                                     html($sor["enev"])."</a></td>\n";
44:                 print "<td><a href=\"klubinfo.php?kazonosito=\".
                                     $sor["eklub"]."\">".
45:                                     html($sor["klubnev"])."</a></td>\n";
46:                 print "<td>".$sor["teruletnev"]."</td>\n";
47:                 print "<td>".$sor["tipusnev"]."</td>\n";
48:                 print "</tr>\n";
49:             }
50:         print "</table>\n";
51:     }
52: ?>
53:
54: <html>
55: <head>
56: <title>Esemény információk</title>

```

24.1. program (folytatás)

```
57:  </head>
58:  <body>
59:  <?php
60:  include("kozosnav.inc");
61:  ?>
62:  <h1>Esemény információk</h1>
63:  <p>
64:  <form action="<?php print $PHP_SELF;?>"
65:  <input type="hidden" name="mitkelltenni"
        value="esemenyLista">
66:  <input type="hidden"
        name="<?php print session_name() ?>"
67:        value="<?php print session_id() ?>">
68:  <select name=urlap[honap]>
69:  <?php honapLehetosegek( $idoszak[0] ); ?>
70:  </select>
71:
72:  <select name=urlap[ev]>
73:  <?php evLehetosegek( $idoszak[0] ); ?>
74:  </select>
75:
76:  <select name=urlap[terulet]>
77:  <option value="BÁRMELY">Bármely terület
78:  <?php optionLista( "teruletek", $urlap["terulet"]) ?>
79:  </select>
80:
81:  <select name=urlap[tipus]>
82:  <option value="BÁRMELY">Bármely típus
83:  <?php optionLista( "tipusok", $urlap["tipus"] ) ?>
84:  </select>
85:
86:  <input type = "submit" value="Listázás">
87:  </form>
88:  </p>
89:
90:  <?php
91:  esemenyLista( );
92:  ?>
93:
94:  </body>
95:  </html>
```

Szokás szerint az `adatbazis.inc` és a `kozosfv.inc` külső állományok beillesztésével kezdjük. Amellett, hogy elérhetjük majd az ezekben tárolt függvényeket, biztosak lehetünk abban, hogy élő adatbáziskapcsolattal rendelkezünk és aktív munkamenettel dolgozhatunk.

Ezután ellenőrizzük, hogy érkezett-e űrlapadat. Ezt a már ismert `mitkelltenni` űrlapelemből származó `$mitkelltenni` változó vizsgálatával tehetjük meg. Ha kitöltött űrlap adatai érkeztek, azok előnyt élveznek minden korábban elraktározott adattal szemben. Az előző órában láthattuk, hogyan rendeltünk egy asszociatív tömböt a munkamenethez, hogy a belépett tag adatait nyilvántartsuk. Ebben az órában új elemeket adunk ehhez, amelyek a felhasználó beállításait tartalmazzák majd.

A `$munkamenet` változót a `kozosfv.inc` állományban rendeltük a munkamenethez (mint azt már az előző órában láttuk):

```
session_start();  
session_register( "munkamenet" );
```

Most ezt a `$munkamenet` tömböt többdimenziós tömbbé alakítjuk, mivel a `$munkamenet["esemenyek"]` tömbelemnek egy asszociatív tömböt adunk értékül. Ebben a tömbben fogjuk tárolni a csak ehhez az oldalhoz tartozó beállításokat. Ezek a terület, típus, hónap és év értékek. Ha űrlap adat érkezett, a korábbi beállításokat ezzel írjuk felül.

Lehetséges, hogy a program futásakor éppen nem érkezett űrlapinformáció, de korábban már tároltuk a felhasználó által előnyben részesített beállításokat. Ebben az esetben a `$mitkelltenni` változó nincs beállítva, viszont a `$munkamenet["esemenyek"]` tömb már tartalmaz elemeket. Ebben az esetben az `$urlap` tömböt írjuk felül a korábban tárolt értékekkel. Ez biztosítja, hogy az űrlapban a megfelelő beállítások látsszanak.

Ha a felhasználó nem küldött be űrlap adatokat és korábban sem tároltuk a beállításait, nekünk kell alapértékekkel feltöltenünk a `$munkamenet["esemenyek"]` tömböt. A `getDate()` beépített PHP függvényt használjuk arra, hogy elemeire bontsuk a dátumot. A visszaadott tömb alapján állítjuk be a `$munkamenet["esemenyek"]["honap"]` és `$munkamenet["esemenyek"]["ev"]` elemeket az aktuális hónap és év értékekre. Már tudjuk, hogy az év és hónap értékek érvényes adatot tartalmaznak, akár űrlapértékekből, akár korábbi beállításból, akár az aktuális dátumból származnak. Ezeket az értékeket a `datum.inc` egy új függvényének, a `datumIdoszak()` függvénynek adjuk át. Ez a függvény hónap és év paramétereket vár és két időbélyeggel tér vissza, megjelölve a hónap kezdetét és végét.

24.2. program Részlet a datum.inc fájlból

```

1: function datumIdoszak( $honap, $ev )
2:     {
3:         $eleje = mktime( 0, 0, 0, $honap, 1, $ev );
4:         $vege = mktime( 0, 0, 0, $honap+1, 1, $ev );
5:         $vege--;
6:         return array( $eleje, $vege );
7:     }

```

A függvény által visszaadott tömböt az \$idoszak globális változónak adjuk értékül.

Létrehozunk egy új esemenyLista() függvényt, amely a megfelelő eseményekről ír majd információkat a böngészőbe. Ezt később a HTML törzs részében hívjuk meg.

Ahhoz, hogy az események listáját megkapjuk, az esemenyekLekeres() függvényt használjuk, amely az adatbazis.inc állományban található. Ezzel a függvénnyel érintőlegesen már az előző órában is találkoztunk, de kevés jó tulajdonságát ismerhettük meg.

24.3 program Részlet az adatbazis.inc fájlból

```

1: function esemenyekLekeres( $kazonosito=0, $idoszak=0,
                             $terulet=0, $tipus=0 )
2:     {
3:         global $kapcsolat;
4:         $lekeres = "SELECT klubok.klubnev, esemenyek.*,
                             teruletek.terulet as teruletnev,
                             tipusok.tipus as tipusnev ";
5:         $lekeres .= "FROM klubok, esemenyek, teruletek,
                             tipusok WHERE ";
6:         $lekeres .= "klubok.azonosito=esemenyek.eklub
7:                     AND esemenyek.eterulet=teruletek.azonosito
8:                     AND esemenyek.etipus=tipusok.azonosito ";
9:         if ( ! empty( $kazonosito ) && $kazonosito
              != "BÁRMELY" )
10:            $lekeres .= "AND esemenyek.eklub=
                          '$kazonosito' ";
11:         if ( ! empty($idoszak) )

```

24.3 program (folytatás)

```

12:          $lekeres .= "AND esemenyek.edatum
                    >= '$idoszak[0]' AND
                    esemenyek.edatum
                    <=' $idoszak[1]' ";
13:      if ( ! empty($terulet) && $terulet != "BÁRMELY" )
14:          $lekeres .= "AND
                    esemenyek.eterulet='$terulet' ";
15:      if ( ! empty($tipus) && $tipus != "BÁRMELY" )
16:          $lekeres .= "AND esemenyek.etipus='$tipus' ";
17:      $lekeres .= "ORDER BY esemenyek.edatum";
18:      $eredmeny = mysql_query( $lekeres, $kapcsolat );
19:      if ( ! $eredmeny )
20:          die ( "esemenyLekeres hiba: ".mysql_error() );
21:      $vissza = array();
22:      while ( $sor = mysql_fetch_array( $eredmeny ) )
23:          array_push( $vissza, $sor );
24:      return $vissza;
25:  }

```

A függvény neve nem igazán fejezi ki sokoldalúságát. Négy paramétert vár: egy klubazonosítót, egy két időbélyeget tartalmazó tömbből álló időszakot, egy terület- és egy típuskódot. Minden paramétere elhagyható vagy hamis, illetve üres értékkel helyettesíthető.

A függvény lényegében egy SQL kérést állít össze a paraméterek alapján. A lekérés alapvetően összekapcsolja az adatbázisban lévő táblákat, ezzel biztosítva, hogy a klubnév, területnév és típusnév mezők is elérhetők legyenek az eredménytáblában.

```

$lekeres = "SELECT klubok.klubnev, esemenyek.*,
                teruletek.terulet as teruletnev,
                tipusok.tipus as tipusnev ";
$lekeres .= "FROM klubok, esemenyek, teruletek,
                tipusok WHERE ";
$lekeres .= "klubok.azonosito=esemenyek.eklub
                AND esemenyek.eterulet=teruletek.azonosito
                AND esemenyek.etipus=tipusok.azonosito ";

```

Ezután a függvény további feltételeket ad ehhez, attól függően, hogy a megfelelő függvényparaméter be van-e állítva. A \$tipus és \$terulet paraméterek akkor is figyelmen kívül maradnak, ha a "BÁRMELY" karakterláncot tartalmazzák.

A gyakorlatban ez azt jelenti, hogy minél több paramétert kap a függvény, annál szűkebb eredménytáblát fogunk kapni. Ha nem adunk át semmilyen paramétert, az összes esemény információit megkapjuk. Ha csupán egy `$kazonosito` paramétert adunk meg, csak az adott klubhoz tartozó eseményeket kapjuk vissza. Ha csak a második `$idoszak` paramétert adjuk meg, csak az adott időszakra eső rendezvényeket láthatjuk és így tovább.

Végül a függvény az SQL parancsot végrehajtja és egy kétdimenziós tömbbel tér vissza, amely a kiválasztott események részleteit tartalmazza.

Miután a függvény által visszaadott kétdimenziós tömböt az `$esemenyek` változóba helyeztük, végig kell lépkednünk annak elemein. Az `edatum` elemeket felhasználva kiírjuk a formázott dátumokat a `date()` beépített függvény segítségével. Ezután egy hivatkozást készítünk az `esemenyinfo.php` oldalra, ahol több információ érhető el egy rendezvényről. Ehhez az esemény azonosítóját és a `SID` állandót kell átadnunk. Ebben a ciklusban egy újabb HTML hivatkozást is készítünk a `klubinfo.php` oldalra, a klub azonosítójával és a `SID` állandóval, végül kiírjuk a terület- és típusneveket a böngészőbe.

Talán feltűnt az előző órában, hogy egy `html()` nevű függvényt használtunk arra, hogy szöveges adatokat írjunk ki a böngészőbe. Az `esemenyLista()` függvényben, ahogy az események információin végiglépkedünk, ismét a `html()` függvényt hívjuk meg. Ez az általunk írt függvény a `kozsofv.inc` állományban található. Egy karakterláncot vár és annak böngészőbe íráshoz alkalmassá tett, átalakított változatával tér vissza. A különleges karakterek HTML elemekké alakulnak, az újsor karakterek például `
`-é.

24.4. program Részlet a kozsofv.inc fájlból

```
1: function html( $szoveg )
2:     {
3:         if ( is_array( $szoveg ) )
4:             {
5:                 foreach ( $szoveg as $kulcs=>$ertek )
6:                     $szoveg[$kulcs] = htmlspecialchars( $ertek );
7:                 return $szoveg;
8:             }
9:         return htmlspecialchars( $szoveg );
10:    }
11:
```

24.4. program (folytatás)

```

12: function htmlspecialchars( $szoveg )
13:     {
14:         $szoveg = htmlspecialchars( $szoveg );
15:         $szoveg = nl2br( $szoveg );
16:         return $szoveg;
17:     }

```

Látható, hogy ez valójában nem egy, hanem két függvény. A `html()` egy karakterláncot vagy egy tömböt vár paraméterül. Ha tömböt kap, végiglépked rajta, átalakítva minden elemet, egyéb esetben magát a kapott karakterláncot alakítja át. A tényleges átalakítás egy másik függvényben, a `htmlspecialchars()`-ben valósul meg, amely két beépített függvényt alkalmaz a kapott karakterláncra. A `htmlspecialchars()` minden karaktert, ami nem jelenne meg helyesen a HTML kódban, a hozzá tartozó HTML elemre cserél. Az `nl2br()` minden újsor karaktert `
` sortöréssel helyettesít.

Visszatérve az `esemenyekinfo.php` oldalra, miután beállítottuk az alapadatokat és létrehoztuk az események kiírására szolgáló függvényt, már csak annyi van hátra, hogy egy űrlapot biztosítsunk a felhasználónak, amely lehetőséget ad a lista szűkítésére, illetve bővítésére.

A felhasználónak választási lehetőséget adó űrlap egyszerűen elkészíthető az előző órában megismert függvényekkel, melyek közvetlenül `OPTION` HTML elemeket írnak ki a böngészőbe.

Az `esemenyekinfo.php` egy lehetséges kimenetét a 24.1. ábra mutatja.

24.1. ábra

Az `esemenyekinfo.php` kimenete



klubokinfo.php

A felhasználó jogos igénnyel nem csak rendezvények, hanem klubok szerint is ki szeretné íratni az adatbázis adatait, típus vagy terület szerint szűkítve azokat. A klubokinfo.php oldal ezt az igényt elégíti ki.

24.5. program klubokinfo.php

```
1: <?php
2:   include("adatbazis.inc");
3:   include("datum.inc");
4:   include("kozofv.inc");
5:   if ( isset($mitkelltenni) &&
        $mitkelltenni == "klubLista" )
6:       $munkamenet["klubok"] = $urlap;
7:   elseif ( $munkamenet["klubok"] )
8:       $urlap = $munkamenet["klubok"];
9:   else
10:      {
11:          $munkamenet["klubok"]["terulet"] = "BÁRMELY";
12:          $munkamenet["klubok"]["tipus"] = "BÁRMELY";
13:      }
14:   function klubLista( )
15:   {
16:       global $munkamenet;
17:       $klubok = klubokLekeres
           ( $munkamenet["klubok"]["terulet"],
18:           $munkamenet["klubok"]["tipus"] );
19:       if ( ! $klubok )
20:       {
21:           print "Nincs megjeleníthető klub<p>\n";
22:           return;
23:       }
24:       print "<table border=1>\n";
25:       print "<td><b>Klub</b></td>\n";
26:       print "<td><b>Terület</b></td>\n";
27:       print "<td><b>Típus</b></td>\n";
28:       foreach ( $klubok as $sor )
29:       {
30:           print "<tr>\n";
```

24.5. program (folytatás)

```
31:             print "<td><a  
                href=\"klubinfo.php?[kazonosito]=\".  
                $sor[\"azonosito\"] . \"&\" . SID . \"\">\".  
32:             html($sor[\"klubnev\"] . \"</a></td>\\n\";  
33:             print "<td>$sor[\"teruletnev\"]</td>\\n\";  
34:             print "<td>$sor[\"tipusnev\"]</td>\\n\";  
35:             print "</tr>\\n\";  
36:             }  
37:             print "</table>\\n\";  
38:             }  
39:     ?>  
40: <html>  
41: <head>  
42: <title>Klub információk</title>  
43: </head>  
44: <body>  
45: <?php  
46: include("kozosnav.inc");  
47: ?>  
48: <P>  
49: <h1>Klub információk</h1>  
50: <p>  
51: <form action="<?php print $PHP_SELF;?>">  
52: <input type="hidden" name="mitkelltenni"  
    value="klubLista">  
53: <input type="hidden" name="<?php print  
    session_name() ?>"  
54:         value="<?php print session_id() ?>">  
55: <select name=urlap["terulet"]>  
56: <option value="BÁRMELY">Bármely terület  
57: <?php optionLista( "teruletek",  
    $urlap[terulet] ) ?>  
58: </select>  
59: <select name=urlap[tipus]>  
60: <option value="BÁRMELY">Bármely típus  
61: <?php optionLista( "tipusok", $urlap["tipus"] ) ?>  
62: </select>  
63: <input type = "submit" value="Rendben">  
64: </form>
```

24.5. program (folytatás)

```

65:  </p>
66:  <?php
67:  klubLista( );
68:  ?>
69:  </body>
70:  </html>

```

Láthatjuk, hogy ez a program szerkezetében és logikájában igencsak hasonlít az előző példában szereplőre. Az ehhez az oldalhoz tartozó „memória” azonban a `$munkamenet["klubok"]` tömbbe kerül. Ha űrlapadatok érkeznek, azokkal írjuk felül a korábbi tartalmát, ha nem érkezett űrlap és már be van állítva, akkor az `$urlap` tömböt írjuk felül a `$munkamenet["klubok"]` változóval. Ha egyik feltétel sem teljesül, alapértékekkel töltjük fel.

Egy `klubLista()` függvényt hozunk létre, amelyben az `adatbazis.inc` egy új függvényét, a `klubokLekeres()`-t hívjuk meg. Ez a függvény a klub területére, illetve típusára vonatkozó elhagyható paramétereket vár.

24.6. program Részlet az `adatbazis.inc` fájlból

```

1: function klubokLekeres( $terulet="", $tipus="" )
2:     {
3:         global $kapcsolat;
4:         $lekeres = "SELECT klubok.*, teruletek.terulet
                    as területnev, tipusok.tipus
                    as tipusnev ";
5:         $lekeres .= "FROM klubok, teruletek,
                    tipusok WHERE ";
6:         $lekeres .= "klubok.terulet=teruletek.azonosito
                    AND klubok.tipus=tipusok.azonosito ";
7:         if ( $terulet != "BÁRMELY" &&
            ! empty( $terulet ) )
8:             $lekeres .= "AND klubok.terulet='\$terulet' ";
9:         if ( $tipus != "BÁRMELY" && ! empty( $tipus ) )
10:            $lekeres .= "AND klubok.tipus='\$tipus' ";
11:         $lekeres .= "ORDER BY klubok.terulet,
                    klubok.tipus, klubok.klubnev";
12:         $eredmeny = mysql_query( $lekeres, $kapcsolat );
13:         if ( ! $eredmeny )
14:             die ( "klubokLekeres hiba: ".mysql_error() );

```

24.6. program (folytatás)

```
15:     $vissza = array();
16:     while ( $sor = mysql_fetch_array( $eredmeny ) )
17:         array_push( $vissza, $sor );
18:     return $vissza;
19: }
```

A `klubokLekeres()` függvény paramétereit függvényében dinamikusan állít elő egy SQL utasítást. Alap esetben csupán összekapcsolja a klubok, területek és típusok táblákat. Ha üres karakterláncra és a "BÁRMELY" karaktersorozatra kívül bármi mást kap a `$terulet` vagy `$típus` paraméterekben, tovább szűkíti az eredménytáblát a `WHERE` feltételhez adott újabb elemekkel. Végül egy kétmenziós tömbbel tér vissza.

A `klubokinfo.php` oldalon végiglépünk ezen a tömbön, kiírva a `területnev` és `típusnev` értékeket is a böngésző számára. Mint eddig, a klub neve most is egy hivatkozásban szerepel, amely a `klubinfo.php` oldalra mutat.

klubinfo.php

A `klubinfo.php` oldal lehetőséget ad egyetlen klub minden adatának megtekintésére. Hivatkozások segítségével juthat ide a felhasználó, akár az `esemenyekinfo.php`, az `esemenyinfo.php` vagy a `klubokinfo.php` oldalakról is. Az oldalt az eddig megismert megoldások és függvények segítségével építhetjük fel, amint az alábbi kód mutatja.

24.7. program klubinfo.php

```
1: <?php
2: include("adatbazis.inc");
3: include(" kozosfv.inc");
4: if ( ! isset($kazonosito) )
5:     header( "Location: klubokinfo.php?".SID );
6: $klub = klubLekeres( $kazonosito );
7:
8: $klub = html( $klub );
9: if ( $klub["email"] != "" )
10:     $klub["email"] = "<A HREF=\"mailto:$klub[email]\">".
                        $klub["email"]."</A>";
```

24.7. program (folytatás)

```

11: function klubEsemenyei ()
12:     {
13:         global $kazonosito;
14:         $esemenyek = esemenyekLekeres( $kazonosito );
15:         if ( ! $esemenyek )
16:             {
17:                 print "Nincs megjeleníthető esemény<P>";
18:                 return;
19:             }
20:         print "<table border=1>\n";
21:         print "<td><b>Dátum</b></td>\n";
22:         print "<td><b>Esemény</b></td>\n";
23:         print "<td><b>Terület</b></td>\n";
24:         print "<td><b>Típus</b></td>\n";
25:         foreach ( $esemenyek as $sor )
26:             {
27:                 print "<tr>\n";
28:                 print "<td>".date("j M Y H.i",
29:                     $sor["edatum"])."</td>\n";
30:                 print "<td><a
31:                     href=\"esemenyinfo.php?eazonosito= ".
32:                     $sor["eazonosito"]." & ".SID."">".htm
33:                     l($sor["enev"])."</a></td>\n";
34:                 print "<td>".$sor["teruletnev"]."</td>\n";
35:                 print "<td>".$sor["tipusnev"]."</td>\n";
36:                 print "</tr>\n";
37:             }
38:         print "</table>\n";
39:     }
40: ?>
41: <html>
42: <head>
43: <title>Klub részletek</title>
44: </head>
45: <body>
46: <?php
47: include("kozosnav.inc");
48: ?>
49: <p>
50: <h1>Klub részletek</h1>
51: <h4><?php print $klub["klubnev"] ?></h4>

```


24.7. program (folytatás)

```
48:  <p>
49:  Terület: <b><?php print $klub["teruletnev"] ?></b>
50:  <br>
51:  Típus: <b><?php print $klub["tipusnev"] ?></b>
52:  <br>
53:  Email: <b><?php print $klub["email"] ?></b>
54:  </p>
55:  Ismertető:<br>
56:  <b><?php print $klub["ismerteto"] ?></b>
57:  <hr>
58:  <?php
59:  klubEsemenyei();
60:  ?>
61:  </body>
62:  </html>
```

A program egy \$azonosito paramétert vár, amely egy klub azonosítószámát kell, hogy tartalmazza. Ha az oldal nem kapott ilyen paramétert, a felhasználót a klubokinfo.php oldalra küldjük. A klub adatainak kiderítésére egy új adatbázis.inc függvényt, a klubLekeres()-t alkalmazzuk. Ez a függvény egy klubazonosítót vár és egy tömböt ad vissza:

24.8. program Részlet az adatbázis.inc fájlból

```
1: function klubLekeres( $kazonosito )
2: {
3:     global $kapcsolat;
4:     $lekeres = "SELECT klubok.*, teruletek.terulet
                as teruletnev, tipusok.tipus
                as tipusnev ";
5:     $lekeres .= "FROM klubok, esemenyek, teruletek,
                tipusok WHERE ";
6:     $lekeres .= "klubok.terulet=teruletek.azonosito
7:                 AND klubok.tipus=tipusok.azonosito
8:                 AND klubok.azonosito='$kazonosito'";
9:     $eredmeny = mysql_query( $lekeres, $kapcsolat );
10:    if ( ! $eredmeny )
11:        die ( "klubLekeres hiba: ".mysql_error() );
12:    return mysql_fetch_array( $eredmeny );
13: }
```

A klub adatainak megszerzésére a `sorLekeres()` függvényt is használhattuk volna. Azért készítettünk mégis egy célfüggvényt erre a feladatra, mivel így az eredménytábla a területnev és típusnev elemeket is tartalmazza majd. Ezt a klubok, területek és típusok táblák összekapcsolásával érhetjük el.

A `klubLekeres()` által visszaadott tömböt a `$klub` változóban tároljuk, tartalmát pedig a HTML törzsrészában írjuk ki a böngésző számára. Az oldalon létrehozunk még egy `klubEseményei()` nevű függvényt is, amely a klubhoz tartozó események kiírását végzi, felhasználva az `esemenyekLekeres()` néven korábban létrehozott függvényünket. A `klubinfo.php` egy lehetséges kimenetét a 24.2. ábra mutatja.

24.2. ábra

A `klubinfo.php` kimenete



esemenyinfo.php

Az `esemenyinfo.php` az utolsó oldal, amit az alkalmazáshoz el kell készítenünk. Erre az oldalra bármely eseményinformációt adó lapról el lehet jutni egy hivatkozás segítségével. Az oldal minden információt megad az `$kazonosito` segítségével kiválasztott eseményről.

24.9. program esemenyinfo.php

```
1: <?php
2:   include("adatbazis.inc");
3:   include("kozosfv.inc");
4:   if ( ! isset($eazonosito) )
5:       header( "Location: esemenyeklista.php?".SID );
6:   $esemeny = esemenyLekeres( $eazonosito );
7:   html( $esemeny );
8:   ?>
9:   <html>
10:  <head>
11:  <title>Esemény részletei</title>
12:  </head>
13:  <body>
14:  <?php
15:    include("kozosnav.inc");
16:    ?>
17:    <P>
18:    <h1>Esemény részletei</h1>
19:    <h4><?php print $esemeny["enev"] ?></h4>
20:    <p>
21:    Klub:
22:    <b>
23:    <?php print "<a href=\"klubinfo.php?kazonosito=\".
24:                $esemeny["eklub"]."\"&\".SID.\"\">
25:    $esemeny["klubnev"]</a>"
26:    ?>
27:    </b>
28:    <br>
29:    Terület: <b><?php print $esemeny["teruletnev"]?></b>
30:    <br>
31:    Típus: <b><?php print $esemeny["tipusnev"] ?></b>
32:    </p>
33:    Ismertető:<br>
34:    <?php print $esemeny["eismerteto"] ?>
35:    </body>
36:  </html>
```

Az eddigiek után ez az oldal már elég egyszerűnek látszik. Egyetlen új függvénnyel találkozhatunk, az `adatbazis.inc` állomány `esemenyLekeres()` függvényével. Ennek segítségével kapjuk meg az adott `$eazonosito`-hoz tartozó esemény részleteit. Miután ezek egy tömbben rendelkezésre állnak, már csak ki kell írunk azokat a böngészőbe.

Az `esemenyLekeres()` függvényt a 24.10. példában láthatjuk. Tulajdonképpen csupán egy egyszerű SQL lekérésből áll, amely a klubok és az események táblákat kapcsolja össze.

24.10. program Részlet az adatbazis.inc fájlból

```
1: function esemenyLekeres( $azonosito )
2:     {
3:         global $kapcsolat;
4:         $lekeres = "SELECT klubok.klubnev as klubnev,
                       esemenyek.*, teruletek.terulet
                       as teruletnev, tipusok.tipus
                       as tipusnev ";
5:         $lekeres .= "FROM klubok, esemenyek, teruletek,
                       tipusok WHERE ";
6:         $lekeres .= "klubok.azonosito=esemenyek.eklub
7:                     AND esemenyek.eterulet=teruletek.azonosito
8:                     AND esemenyek.etipus=tipusok.azonosito
9:                     AND esemenyek.eazonosito='$azonosito'";
10:        $eredmeny = mysql_query( $lekeres, $kapcsolat );
11:        if ( ! $eredmeny )
12:            die ( "esemenyLekeres hiba: ".mysql_error() );
13:        return mysql_fetch_array( $eredmeny );
14:    }
```

A jövő

Mostanra elkészültünk a teljes rendezvénynaptárral. Remélem sikerült érzékeltetni ezzel a valós életbeli kisalkalmazások dinamikus kialakításának lehetőségeit és a PHP szerteágazó képességeit.

Különösen érdemes megfigyelni, mennyire megkönnyítik munkánkat a PHP 4 munkameneteket kezelő függvényei, mivel ezek használatával igen egyszerű az adatok megőrzése kérésről kérésre. Ha a felhasználónk a munkamenet során később visszatér mondjuk az `esemenyekinfo.php` oldalra, pontosan azokat a beállításokat fogja találni, mint amelyeket legutóbb otthagytott. A munkamenet-függvények nélkül feltehetően sokkal több információt kellett volna kérésről-kérésre átadnunk oldalainknak az URL-ekben.

Bár az eseménynaptár bizonyos megközelítésből teljesnek tekinthető, mindazonáltal csak egy jó prototípus, amit meg tudunk mutatni a megrendelőnek, ahogy haladunk a munkával. Az alkalmazás hasznára válna néhány további szolgáltatás, különösképpen egy kulcsszavas keresés. Kellemes lenne lehetőséget adni a látogatóknak, hogy megjegyzéseket fűzhessenek a rendezvényekhez. Ez új dimenziót nyitna az alkalmazás számára, igazán érdekes környezetté alakítva azt.

Lehetőséget adhatnánk a tagok számára, hogy képekre mutató hivatkozásokat helyezzenek el a klubok ismertetőiben. Akár azt is megoldhatnánk, hogy képeket tölthessenek fel böngészőjük segítségével.

A tagok feltehetően örülnének, ha az eseményeket lemásolhatnák vagy a rendezvények ismétlődő jellegét is kezelhetnék.

Mielőtt az alkalmazást átnyújtanánk a megbízónak, szükségünk lesz egy karbantartó felületre, amely alkalmas arra, hogy egy technikailag képzetlen adminisztrátor is módosíthasson vagy törölhessen tagokat, illetve az eseményeket, valamint a terület- és típuskategóriákat megváltoztathassa.

Valószínűleg feltűnt már, hogy programjaink kimenete eléggé spártai. Végül át kell adnunk a munkát egy grafikus tervezőnek, aki ügyes vezérlőssávet, komoly grafikát és más elemeket ad az oldalhoz. Szerencsére a legtöbb PHP kód külső állományokban található, a HTML-től függetlenül, de elképzelhető, hogy szükség lesz ránk is a ciklusok újraírásánál.

Összefoglalás

Ebben és az előző órában elkészítettünk egy teljesen működő többoldalas PHP alkalmazást. Gyakoroltuk az állapot mentésének, a felhasználói azonosításnak, az adatbázis-adatok módosításának és megjelenítésének módjait és sok más kérdést is érintettünk.

Egy többoldalas alkalmazás végigvitele egy könyvben nem könnyű dolog, de megéri a kitartást. Olyan kérdésekre adtunk válaszokat, olyan problémákat oldottunk meg, amelyekkel munkánk során időről időre találkozni fogunk. Majdnem minden programnak, amit írunk, egynél több felhasználó kéréseit kell majd egyidőben kiszolgáltatnia, tehát egyes megoldásokra van szükség az állapotok mentésére.

Kérdések és válaszok

Ennyi volt. Hogyan tovább?

Ez már nem a könyvön múlik. Elég információt találhatunk ebben a könyvben ahhoz, hogy saját kifinomult alkalmazásainkat el tudjuk készíteni. Ezzel és a Világhálón elérhető rengeteg információval nincs, ami megállíthatna bárkit is a fejlődésben!

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvény használható új elem tömb végére való beszúrásához?
2. Lehetséges egy új elem felvétele a tömb végére függvény használata nélkül is?
3. Melyik függvény alkalmazható a különleges karakterek átalakítására HTML-kiírás céljából?
4. Melyik függvény alakítja át az újsor karaktereket `
` elemekké?
5. A `SID` állandó a munkamenet-azonosító oldalról-oldalra küldéséhez hivatkozások készítésénél használható. Hogyan érhető el ugyanez a hatás úrlap készítésekor?

Feladatok

1. Tekintsünk végig az órában tárgyalt forráskódokon. Találunk olyan megoldásokat, amiket fel tudunk használni saját programjainkban?
2. Lapozzuk át a könyvet és jegyzeteinket, ha írtunk ilyeneket. Jusson később eszünkbe, hogy érdemes kis idő elteltével újra átnézni a jegyzeteket, hogy minél több hasznunk legyen a befektetett munkából.