

A PHP nyelv

(gyakorlati jegyzet)

Rigó Ernő

Tartalom

A PHP nyelv.....	1
Bevezető.....	10
Webes architektúrák és a PHP.....	10
A klasszikus webalkalmazások működése.....	10
A modern webalkalmazások működése.....	11
Munkakörnyezet kialakítása és alapszintű használata.....	11
A WAMP5 telepítése.....	12
Az EasyEclipse for PHP telepítése.....	13
Órai jegyzetek.....	14
A PHP futtatása.....	14
CLI.....	14
CGI.....	14
MOD_PHP.....	14
A PHP bemenete.....	15
GET.....	15
POST.....	15
COOKIE.....	15
A PHP kimenete.....	16
Előnyök, felhasználási terület.....	16
Hátrányok.....	16
A PHP Safe Mode.....	16
WAMP5.....	17
EasyEclipse for PHP.....	17
Szoftver architektúra.....	17
Projektkezelés.....	17
Hello World!.....	17
A PHP nyelv alapjai.....	18
Alapvető szintaxis.....	18
Típusok.....	18
Változók.....	19
Kifejezések.....	20
Operátorok.....	20
Vezérlési szerkezetek.....	21
Függvények.....	21
PHP5 OO alapok.....	21
Referenciák.....	22
Biztonság.....	22
A PHP API áttekintése.....	23
Tömbök.....	23
Dátum és idő.....	23
Fájlok és könyvtárak.....	23
SMTP.....	23
SQL gyorstalpaló.....	23
SQL.....	24
Stringkezelés.....	24
Reguláris kifejezések.....	24
PCRE gyorstalpaló.....	24
PHP tervezési minták.....	24
HTML FORM.....	24
Formkezelés és ellenőrzés.....	25

Formok (újra)feltöltése.....	26
Többnyelvűség.....	26
Fájlfeltöltés kezelése.....	26
Munkamenetek.....	26
Cookie.....	26
HTTP Hitelesítés.....	26
Cookie alapú hitelesítés.....	27
Template rendszerek.....	27
Adatbázis alapú funkciók.....	28
Dinamikus képek.....	28
Melléklet: forráskódok.....	29
_feladatok/alap2 - 010 fibonacci.php.....	29
_feladatok/api1 - 010 oszthatosagi szures.php.....	30
Melléklet: a hivatalos PHP kézikönyv.....	77
PHP kézikönyv.....	77
Előszó.....	79
Szerzők és más segítők.....	79
Szerzők és szerkesztők.....	79
Felahsználói megjegyzések kezelői.....	80
I. Első lépések.....	80
1. Fejezet. Bevezetés a PHP-be.....	80
Mi az a PHP?.....	80
Mit tud a PHP?.....	81
2. Fejezet. A simple tutorial.....	82
What do I need?.....	83
Your first PHP-enabled page.....	83
Something Useful.....	85
Dealing with Forms.....	86
Using old code with new versions of PHP.....	87
What's next?.....	87
II. Telepítés és beállítás.....	88
3. Fejezet. Általános telepítési szempontok.....	88
4. Fejezet. Telepítés Unix rendszerre.....	89
Apache 1.3.x Unix rendszereken.....	89
Apache 2.0 Unix rendszereken.....	93
Caudium.....	95
fhttpd megjegyzések.....	96
Sun, iPlanet and Netscape servers on Sun Solaris.....	96
CGI environment and recommended modifications in php.ini.....	98
Special use for error pages or self-made directory listings (PHP >= 4.3.3).....	98
Note about nsapi_virtual() and subrequests (PHP >= 4.3.3).....	99
CGI és parancssori verzió.....	99
Tesztelés.....	99
Szintmérés (benchmarking).....	99
Változók használata.....	100
Telepítés HP-UX rendszerre.....	100
Telepítés OpenBSD rendszerre.....	101
Bináris csomagok használata.....	101
Port-ok használata.....	102
Általános problémák.....	102
Régebbi kiadások.....	102
Telepítés Solaris rendszerre.....	102

Szükséges programok.....	102
Csomagok használata.....	103
Telepítés Gentoo rendszerre.....	103
A Portage (emerge) használata.....	103
A konfiguráció beállítása.....	104
Általános problémák.....	104
5. Fejezet. Telepítés Mac OS X rendszerre.....	105
Csonmagok használata.....	105
Fordítás OS X szerveren.....	105
Fordítás MacOS X kliensre.....	106
6. Fejezet. Telepítés Windows rendszerekre.....	107
A Windows telepítő.....	107
A kézi telepítés lépései.....	107
ActiveScript.....	112
Microsoft IIS / PWS.....	112
Általános telepítési szempontok IIS-hez.....	113
Windows NT/200x/XP és IIS 4 vagy újabb.....	113
Windows és PWS 4.....	114
Windows és PWS/IIS 3.....	115
Apache 1.3.x Microsoft Windows-on.....	116
Telepítés Apache modulként.....	116
Telepítés CGI binárisként.....	117
Apache 2.0.x on Microsoft Windows.....	117
Installing as a CGI binary.....	118
Installing as an Apache module.....	119
Sun, iPlanet and Netscape servers on Microsoft Windows.....	119
CGI setup on Sun, iPlanet and Netscape servers.....	120
NSAPI setup on Sun, iPlanet and Netscape servers.....	120
CGI environment and recommended modifications in php.ini.....	121
Special use for error pages or self-made directory listings (PHP >= 4.3.3).....	121
Note about nsapi_virtual() and subrequests (PHP >= 4.3.3).....	122
OmniHTTPd Server.....	122
Sambar Server on Microsoft Windows.....	123
Xitami on Microsoft Windows.....	123
Fordítás forrásból.....	124
Követelmények.....	124
Az egész összerakása.....	124
Az MVC ++ konfigurálása.....	125
Build resolv.lib.....	125
Fordítás.....	126
Kiterjesztések telepítése Windows-on.....	127
7. Fejezet. PECL kiterjesztések telepítése.....	130
Bevezetés a PECL telepítésébe.....	130
PECL kiterjesztések letöltése.....	130
PECL Windows felhasználók részére.....	131
Megosztott PECL kiterjesztések fordítása PEAR-rel.....	131
Megosztott PECL kiterjesztések fordítása phpize-vel.....	131
PECL kiterjesztések fordítása PHP-be statikusan.....	132
8. Fejezet. Problémák?.....	132
Olvasd el a FAQ-t.....	132
Egyéb problémák.....	132
Hibajelentések.....	133

9. Fejezet. Futásidéjű beállítások.....	133
A konfigurációs állomány.....	133
Konfigurációs beállítások megváltoztatása.....	134
PHP Apache modulként.....	134
A PHP konfiguráció megváltoztatása a Windows registry segítségével.....	135
Egyéb interfések a PHP-hez.....	135
III. A nyelv alapjai.....	136
10. Fejezet. Alapvető szintaxis.....	136
Escape szekvencia HTML-ben.....	136
Utasítások elválasztása.....	137
Megjegyzések - kommentek.....	138
11. Fejezet. Típusok.....	138
Bevezető.....	138
Logikai adattípus.....	139
Szintaxis.....	139
Logikai értékké alakítás.....	140
Egész számok.....	141
Szintaxis.....	141
Egészek értelmezési határának túllépése.....	141
Egész értékké alakítás.....	142
Átalakítás boolean (logikai) értékekről.....	142
Átalakítás lebegőpontos értékekről.....	142
Átalakítás karakterláncokról.....	143
Átalakítás más típusokról.....	143
Lebegőpontos számok.....	143
Konverzió float-ra.....	144
Stringek.....	144
Szintaxis.....	144
String létrehozása aposztróffal.....	144
String létrehozása idézőjellel.....	145
String létrehozása heredoc szintaxissal.....	145
Változók behelyettesítése.....	147
Egyszerű szintaxis.....	147
Komplex (kapcsos zárójeles) szintaxis.....	148
String karaktereinek elérése és módosítása.....	149
Hasznos függvények és operátorok.....	149
Konverzió stringgé.....	149
String konvertálása számmá.....	150
Tömbök.....	151
Szintaxis.....	151
Tömb létrehozása az array() nyelvi elemmel.....	151
Létrehozás/módosítás a szögletes zárójeles formával.....	152
Hasznos függvények.....	154
Mit lehetünk, és mit nem a tömbökkel.....	154
Miért nem jó az \$size[bigyo] forma?.....	154
De miért nem jó ez?.....	156
Konverzió tömbbé.....	157
Tömbök összehasonlítása.....	157
Példák.....	157
Objektumok.....	160
Objektumok létrehozása.....	160
Konverzió objektummá.....	160

Erőforrások.....	160
Konverzió erőforrásra.....	161
Erőforrások felszabadítása.....	161
NULL.....	161
Szintaxis.....	161
E dokumentációban szereplő pszeudo-típusok.....	162
mixed.....	162
number.....	162
callback.....	162
Bűvészkedés a típusokkal.....	163
Típuskonverziók.....	163
12. Fejezet. Változók.....	164
Alapok.....	164
Előre definiált változók.....	166
Változók hatásköre.....	168
A global kulcsszó.....	168
Statikus változók használata.....	169
Referenciák globális és statikus változókkal.....	170
Változó változók.....	172
Változók a PHP-n kívülről.....	172
HTML űrlapok (GET és POST).....	172
IMAGE SUBMIT változónevek.....	174
HTTP sütek (cookie).....	174
Pontok a bejövő változónevekben.....	175
Változótípusok meghatározása.....	175
13. Fejezet. Állandók.....	175
Szintakszis.....	176
Mágikus konstansok.....	177
14. Fejezet. Kifejezések.....	177
15. Fejezet. Operátorok.....	180
Operátorok precedenciája.....	180
Aritmetikai operátorok.....	181
Hozzárendelő operátorok.....	182
Bitorientált operátorok.....	182
Összehasonlító operátorok.....	183
A ternáris operátor.....	185
Hibakezelő operátorok.....	185
Végrehajtó operátorok.....	186
Növelő/csökkentő operátorok.....	186
Logikai operátorok.....	187
String operátorok.....	188
Tömb operátorok.....	188
Típus operátorok.....	189
16. Fejezet. Vezérlési szerkezetek.....	190
if.....	190
else.....	190
elseif.....	191
Vezérlési szerkezetek alternatív szintaxisa.....	191
while.....	192
do-while.....	193
for.....	193
foreach.....	195

break.....	197
continue.....	197
switch.....	198
declare.....	201
Tick-ek.....	201
return.....	202
require().....	202
include().....	203
require_once().....	207
include_once().....	208
17. Fejezet. Függvények.....	208
Felhasználó által definiált függvények.....	208
Függvényargumentumok.....	210
Referencia szerinti argumentumfeltöltés.....	210
Argumentumok kezdőértékei.....	211
Változó hosszúságú argumentumlista.....	212
Visszatérési értékek.....	212
Függvényváltozók.....	213
Belső (beépített) függvények.....	214
18. Fejezet. Osztályok, objektumok (PHP 4-ben).....	215
class.....	215
extends.....	218
Konstruktur.....	219
Hatókör megadó operátor (::).....	220
parent.....	221
Objektumok szerializációja, objektumok session-ökben.....	222
A speciális __sleep és __wakeup metódusok.....	223
Referenciák a konstruktorkban.....	223
Objektuk összehasonlítása.....	225
19. Fejezet. Osztályok és objektumok (PHP 5).....	228
Bevezetés.....	228
Az alapok.....	228
class.....	228
new.....	229
extends.....	230
Automatikusan betöltődő objektumok.....	231
Konstruktork és destruktork.....	231
Konstruktur.....	231
Destruktor.....	232
Láthatóság.....	232
Adattagok láthatósága.....	233
Metódus láthatóság.....	234
Scope Resolution Operator (::).....	235
A static kulcsszó.....	236
Osztály konstansok.....	237
Elvont osztályok (abstract).....	237
Objektum interfészek.....	238
implements.....	239
Példák.....	239
Overloading.....	240
Member overloading.....	240
Method overloading.....	242

Object Iteration.....	242
Patterns.....	246
Factory.....	246
Singleton.....	246
Magic Methods.....	247
__sleep and __wakeup.....	247
__toString.....	248
__set_state.....	249
A final kulcsszó.....	249
Objektum klónozás.....	250
Comparing objects.....	251
Reflection.....	253
Introduction.....	253
ReflectionException.....	254
ReflectionFunction.....	254
ReflectionParameter.....	256
ReflectionClass.....	257
ReflectionObject.....	259
ReflectionMethod.....	260
ReflectionProperty.....	261
ReflectionExtension.....	263
Extending the reflection classes.....	263
Type Hinting.....	264
20. Fejezet. Kivételek (Exceptions).....	266
Kivételek kiterjesztése.....	266
21. Fejezet. Referenciák.....	269
Mik a referenciák.....	269
Mit lehet referenciákkal tenni.....	269
Mit nem lehet referenciákkal tenni.....	271
Referenciáként paraméterátadás.....	271
Refencia visszatérési-érték.....	272
Referenciák megszüntetése.....	273
A PHP által használt referenciák.....	273
global referenciák.....	273
\$this.....	273
IV. Biztonság.....	273
22. Fejezet. Bevezetés.....	274
23. Fejezet. Általános szempontok.....	274
24. Fejezet. CGI futtatható állományként telepített PHP.....	275
Lehetséges támadások.....	275
1. eset : csak publikus fájlok.....	276
2. eset : az --enable-force-cgi-redirect használata.....	276
3. eset : a doc_root vagy user_dir beállítása.....	276
4. eset : PHP feldolgozó a web könyvtárfán kívül.....	277
25. Fejezet. Apache modulként telepített PHP.....	277
26. Fejezet. Fájlrendszer biztonság.....	278
27. Fejezet. Adatbázis biztonság.....	279
Adatbázis-tervezés.....	280
Kapcsolódás az adatbázishoz.....	280
Titkosított tárolás.....	280
SQL "beoltás".....	281
Elhárítási módszerek.....	283

28. Fejezet. Hibakezelés.....	284
29. Fejezet. Globálisan is elérhető változók (Register Globals) használata.....	285
30. Fejezet. Felhasználótól érkező adatok.....	287
31. Fejezet. Magic Quotes.....	288
What are Magic Quotes.....	288
Why use Magic Quotes.....	289
Why not to use Magic Quotes.....	289
Disabling Magic Quotes.....	289
32. Fejezet. A PHP elrejtése.....	290
33. Fejezet. Fontos aktuálisnak maradni.....	291
V. Szolgáltatások.....	291
34. Fejezet. HTTP hitelesítés PHP-vel.....	291
35. Fejezet. Sütik (cookie-k).....	294
36. Fejezet. Sessions.....	294
37. Fejezet. Dealing with XForms.....	294
38. Fejezet. Fájlfeltöltés kezelése.....	295
POST metódusú feltöltések.....	295
Hibaüzenetek magyarázatai.....	298
Tipikus csapdák.....	298
Több állomány egyidejű feltöltése.....	299
PUT metódusú feltöltések.....	300
39. Fejezet. Távoli állományok kezelése.....	301
40. Fejezet. Kapcsolatkezelés.....	302
41. Fejezet. Állandó adatbázis kapcsolatok.....	303
42. Fejezet. Safe Mode.....	304
Security and Safe Mode.....	304
Functions restricted/disabled by safe mode.....	307
43. Fejezet. Parancssori programozás a PHP-ben.....	311

Bevezető

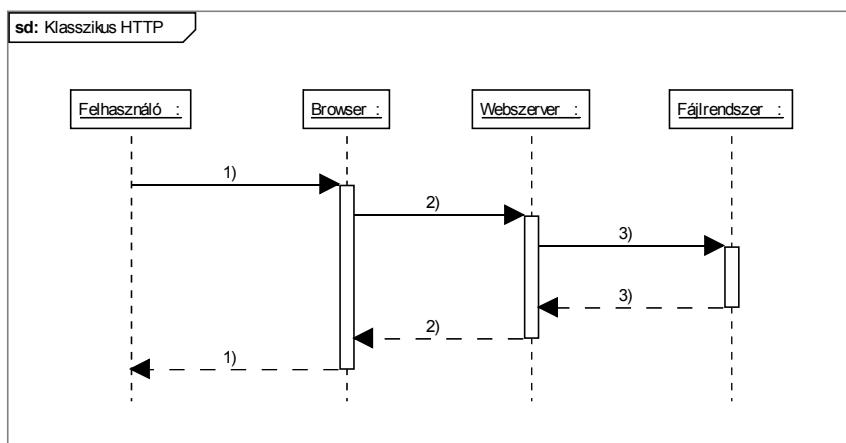
Kedves Hallgató! Jelen jegyzet a „PHP nyelv” című gyakorlati tárgy segédleteként szolgál, célja nem a PHP programozási nyelv részletes ismertetése, hanem a tárggyal kapcsolatos anyagok megfelelő rendszerezése, a tárgy tematikus és a gyakorlati támogatása. Elsődleges célkitűzésünk a vizsgára való önálló felkészülés elősegítése.

A jegyzet alapvető részét, elválaszthatatlan kiegészítőjét képzi a hivatalos és szabadon letölthető PHP kézikönyv, mely a <http://hu.php.net/manual/hu> oldalon olvasható on-line formájában.

A tárgyhoz használandó segédletek és alkalmazások elérhetők a <http://www.tricon.hu/~mcree/php/weblapon>. A jegyzet feltételezi ezek aktív használatát.

Webes architektúrák és a PHP

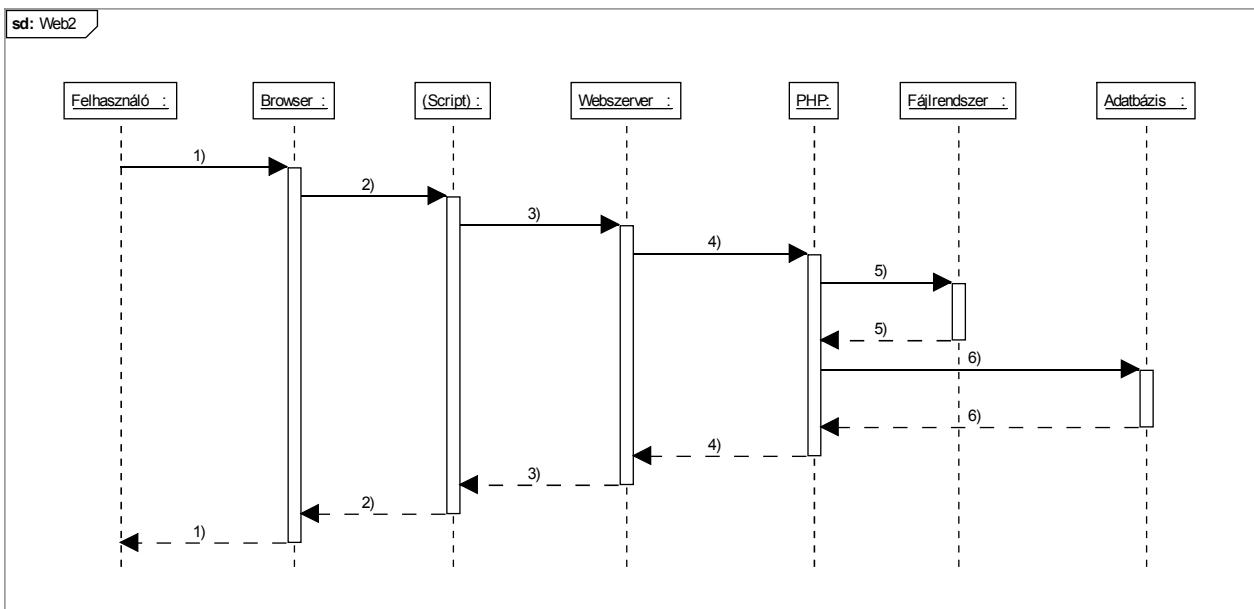
A klasszikus webalkalmazások működése



Ábra 1: klasszikus HTTP szekvenciadiagram

1. A felhasználó kérést küld a webbrowsernek, vagyis például egy linkre (URI) kattint, majd meglátja a link (URI) által azonosított webtartalmat, az URI a fájl szerveren való elhelyezkedését tükrözi
2. A browser az URI-t értelmezi és kiválasztja belőle a webszerver internetes azonosítóját, valamint a szükséges protokolلت, fogadja a fájlt a webszerverről és azt értelmezve grafikusan megjeleníti
3. A webszerver az URI-t értelmezi és kiválasztja az ennek megfelelő fájlt a fájlrendszerrel, azt betölti és visszaküldi a browsernek

A modern webalkalmazások működése



Ábra 2: HTTP szekvenciadiagram Web 2.0 környezetben

1. A felhasználó kérést küld a webbrowsernek, vagyis például egy linkre (URI) kattint, majd meglátja a link (URI) által azonosított webtartalmat, az URI tipikusan könnyen olvasható, logikai tartalmat fed
2. A browser az URI-t értelmezi és kiválasztja belőle a webszerver internetes azonosítóját, valamint a szükséges protokolلت fogadja a fájlt a webszerverről és azt értelmezve grafikusan megjeleníti, opcionálisan a webszerverrel való közvetlen kapcsolatteremtés helyett valamilyen beágyazódó scriptet (Flash, JavaScript) használ a tartalom letöltésére és megjelenítésére
3. Opcionálisan egy, vagy több beágyazódó script a felhasználói felületen végzett akcióira válaszul, vagy önállóan további kéréseket is indíthat a webszerver irányába. (AJAX)
4. A webszerver az URI-t értelmezi, esetleg át is alakítja és kiválasztja az ennek megfelelő erőforrást, például a PHP interpretert (parancsértelmezőt) és továbbküldi neki a kérést. Az erőforrás válaszát továbbítja a böngészőnek.
5. A PHP parancsértelmező betölti az URI által azonosított parancsfájlt, lefordítja és futtatja. A futás eredményét visszaküldi a webszerver számára.
6. A PHP parancsértelmező által futtatott parancsfájl opcionálisan további erőforrásokat, például jellemzően adatbázist ér el, az abból származó adatokat felhasználja kimenetének előállítására.

Munkakörnyezet kialakítása és alapszintű használata

A gyakorlatok során a PHP nyelv megisméréséhez szükségünk lesz egy működőképes futatókörnyezet kialakítására. Számos lehetőség közül választhatunk, azonban az egyszerű telepíthetőséget és a gyors betanulási ciklust figyelembe véve az ingyenesen letölthető WAMP (<http://www.wamp.com>) Apache+PHP+MySQL integrált telepítőcsomagot és a kényelmes szerkesztési lehetőséget biztosító EasyEclipse for PHP (<http://www.easyeclipse.org>) szoftvereket választottuk. Ezek a szoftverek a megadott weblapokról legfrissebb verzióikban letölthetők, azonban a telepítési útmutató és a gyakorlatok során a wamp5 1.7.0 és a easyeclipse-php 1.2.1.1 verziókkal dolgozunk, melyek a <http://www.tricon.hu/~mcree/php> címen is elérhetők. Ajánlatos az egyéni gyakorlás során is ezeket, vagy nem sokkal különböző verziókat használni, mivel az egyes nagyobb verziótételek esetén a programok viselkedése némi képp módosulhat.

Ennek a telepítési útmutatónak célja, hogy egy Windows XP operációs rendszert futtató számítógépen elvezesse az olvasót a fent leírt munkakörnyezetben a klasszikus "Hello World!" PHP alkalmazás futtatásának lehetőségéig, végigjárva az ehhez szükséges szoftverek telepítésének lépései. A következőkben részletes képernyöképeket láthatunk minden egyes lépéshez. Az eligazító információk a képek fölött olvashatók.

A WAMP5 telepítése

A wamp5 telepítőcsomag 1.7.0 verziója a következő szoftvereket telepíti integráltan számítógépünkre:

- Apache 2.2.4
- PHP 5.2.1 + PECL
- SQLiteManager
- MySQL 5.0.27
- PHPMyAdmin

Az alábbiakban megadott lépések a <http://www.tricon.hu/~mcree/php/telepites/> címen képes illusztrációval olvashatók:

1. Első lépésként töltük le és futtassuk a wamp5_1.7.0.exe telepítőszoftvert. A telepítéshez rendszergazdai jogosultságok szükségesek.
2. A wamp5 licensze az általa telepített szoftverekkel megegyezően a General Public License 2-es verziója, mely a megjelenített ablakban elfogadandó. A licensz a <http://www.gnu.org/licenses/gpl.html> címen is olvasható, sajnos hivatalos magyar fordítás még nem készült belőle. A licensz röviden összefoglalva lehetővé teszi más (pl. üzleti) licenszű szoftverek fejlesztését GPL-es termékekkel, de a GPL-es termékek forráskódjára épülő termékek forráskódjának kiadására kötelezi a fejlesztőket. minden GPL licenszű szoftver, vagy annak módosítása felhasználható üzleti célra. A gyakorlatok során a telepített szoftvereket csak felhasználni fogjuk, nem módosítjuk őket.
3. Adjuk meg a telepítés célkönyvtárát. A wamp által telepített összes szoftverkomponenst ebben a könyvtárban találjuk meg a későbbiek során. Példáink során a következő megadott útvonalat feltételezzük: c:\wamp
4. A start menüben regisztrálásra kerülő programcsoport-elnevezést adhatjuk meg a következő ablakban. Ezt az értéket nyugodtan alapértelmezésen hagyhatjuk.
5. A következő dialógusban lehetőségünk van bejelölni, hogy a wamp csomag szolgáltatásai a rendszer indításakor automatikusan elinduljanak. Ellenkező esetben kézzel szükséges a komponenseket elindítani a Windows rendszer szolgáltatáskezelőjében. A gyakorlatok során az automatikus indítást választjuk, vagyis bejelöljük a választónégyzetet, a zökkenőmentes működés érdekében javasoljuk, hogy otthon is ezt a beállítást válassza.
6. Az ellenőrzőképernyő megtekintése után az Install (Telepítés) gombra kattintva megkezdődik a wamp5 alkalmazások másolása a megadott mappába.
7. A telepítés befejezése után ki kell választanunk az Apache webszerver ún. dokumentumgyökerét (documentroot), ahonnan a webes tartalmak kiszolgálását végzi. Ez célszerűen az automatikusan megadott érték, vagyis példánkban: c:\wamp\www. Jegyezzük meg, mert a későbbiekben szükség lesz még rá.
8. A PHP levélküldő funkcióinak kihasználásához meg kell adnunk egy SMTP protokollon emailket fogadó internethoz címet. Ha nem tudunk ilyet megadni, nyugodtan hagyjuk "localhost" értéken, ez csak a PHP levélküldő függvényeinek működését érinti, ezekre pedig egyelőre nem lesz szükség.
9. Ugyanezt a kérdést feszegeti a következő képernyő is, ahol a PHP rendszerből kimenő levelek feladóját lehet megadni. Nyugodtan hagyhatjuk alapértelmezésen.
10. Ha a gépünkön a Firefox webböngésző, a wamp5 felajánlja, hogy alapértelmezett böngészőként ezt használja. Válasszunk ízlésünk szerint.
11. A wamp5 telepítésének végén lehetőségünk van a wamp azonnali elindítására is. Válasszuk

- ki ezt a lehetőséget is, majd fejezzük be a telepítést.
12. Ha minden rendben lezajlott, a tálcán megjelenik a wamp5 ikonja, erre kattintva az alábbi menüt kell látnunk. Itt van lehetőségünk a wamp5 komponensek finomhangolására, valamint elindítására és leállítására.
 13. Végső ellenőrző lépésként látogassuk meg a <http://localhost/> címet. Itt a wamp5 helyi szerverkonfigurációjának kell jelentkeznie. Ezzel a telepítést sikeresen befejeztük.

Az EasyEclipse for PHP telepítése

Az EasyEclipse for PHP az Eclipse (<http://www.eclipse.org>) univerzális fejlesztőkörnyezet PHP szerkesztésre specializált verzióját tartalmazza. Bár lehetőségünk lenne különböző egyéb szövegszerkesztők segítségével is feladataink elvégzésére, a gyakorlatok során az Eclipse fejlesztőkörnyezetet használjuk integráltsága és egyéb szolgáltatásai miatt. A telepítés megkezdése előtt győződjünk meg róla, hogy a wamp5 telepítése sikeresen befejeződött. Útmutatónk során feltételezzük a telepítési sorrend betartását.

Az alábbiakban megadott lépések a <http://www.tricon.hu/~mcree/php/telepites/> címen képes illusztrációval olvashatók:

1. Első lépésként töltük le és futtassuk a `easyeclipse-php-1.2.1.1.exe` telepítőszoftvert. A telepítéshez rendszergazdai jogosultságok szükségesek.
2. Az üdvözlőképernyőt ugorjuk át.
3. A következő képernyőn lehetőségünk van az EasyEclipse for PHP licenszének elfogadására, amit meg is teszünk. A licensz típusa Open Software License (OSL), mely a legtöbb aspektusból a fent leírt GPL-hez hasonlít. Mindkettő megegyezik az üzleti felhasználás területén tett kikötésekben, vagyis üzleti célból történő felhasználást engedélyez. A feltelepített Eclipse licensze az Eclipse Public License (EPL), mely szintén az előbbiekhez hasonló kikötéseket tesz, azonban egy kicsit megengedőbb.
4. A telepítés könyvtárát adhatjuk meg a következő képernyőn. Útmutatónkban ezt az alapértelmezett értéken hagytuk.
5. A telepítés folyamatát várjuk végig.
6. A telepítés végén kattintsunk a Close (Bezár) gombra.
7. A telepítés befejeztével indítsuk el az EasyEclipse for PHP alkalmazást, melyet a Start menüből az alkalmazások között találunk.
8. Az alkalmazás első indításakor megkérdezi, hol legyen a munkaterület könyvtára. Ne fogadjuk el az alapértelmezést, hanem kattintsunk a browse gombra.
9. A felugró ablakban válasszuk ki azt a könyvtárat, ahová a wamp5 documentroot (dokumentum-gyökér) könyvtárat helyeztük. Útmutatónk esetén ez a könyvtár a következő: `c:\wamp\www`
10. A választónégyzet bejölösével lehetőségünk nyílik a kérdés továbbiakban történő elhagyására. Jelöljük be az opciót.
11. Ha a telepítés sikeres volt az következő képernyőn az Eclipse üdvözlőképernyőjét láthatjuk. Kattintsunk a Welcome (Üdvözlet) fülön található X-re.
12. Az Eclipse ekkor az alapértelmezett munkaasztal-nézetbe kerül.
13. Válasszuk ki a File menü New - Project opcióját. Ezzel az opcióval hozhatunk létre a gyakorlatok során további projekteket.
14. A felugró ablakban válasszuk a PHP Project lehetőséget, majd kattintsunk a Next (Következő) gombra.
15. Válasszunk nevet az új projektnek, példánkban ez a név a `helloworld`.
16. Az Eclipse ekkor PHP nézetre vált. A navigációs panelon megjelenik a `helloworld` projekt.
17. Kattintsunk jobb gombbal a `helloworld` projektre és válasszuk a New - PHP File opciót. Ezzel az opcióval hozhatunk létre PHP projektjeinkben új parancsfájlokat.
18. A container értéket változatlanul hagyva az új PHP fájl nevét kell megadnunk a következő ablakban. Útmutatónkban a `helloworld.php` nevet választottuk. Kattintsunk a Finish

(Befejezés) gombra.

19. A munkaterületen megjelenik a helloworld.php és annak forrása. A forrás jelenleg egy előre elkészített minta alapján az aktuális dátumot és még pár Eclipse-specifikus információt jelző megjegyzést tartalmaz.
20. Akár a PHP nyelv minden előzetes ismerete nélkül az alábbi ablakon látható helyre és módon írjuk be az echo "hello world!"; karakterszorozatot.
21. Ezután válasszuk a File menü Save opciónát.
22. Ha minden rendben, a munkaterület alsó részén található panelon a PHP Browser fület választva a hello world! felirat jelenik meg. Ugyanezt az eredményt kell kapnunk akkor is, ha egy tetszőleges webböngészővel gépünkről meglátogatjuk a <http://localhost/helloworld/helloworld.php> címet.

Ezzel sikeresen előállítottuk a gyakorlatok során használt munkakörnyezetet.

Órai jegyzetek

Kedves Hallgató!

Az alábbiakban a gyakorlatokon használt slideok szöveges tartalmát olvashatja. Célunk ezzel az, hogy egyszerűbben nyomtathatóvá tegyük azt a vázlatot, melynek használatával a gyakorlatokon elhangzottak reprodukálhatók a hivatalos PHP kézikönyv szövegének elolvasása mellett. Nem szándékunk a PHP kézikönyv tartalmi reprodukálása, inkább a különböző fejezetek kihangsúlyozása, az önálló felkészülés elősegítése a cél. Ne feledjük, hogy egy programozási nyelvet nem egy szöveges tananyag elsajátításával, hanem önálló próbálkozással lehet leginkább megismerni.

A PHP futtatása

Három fajta futtatási üzemmódot különböztetünk meg:

CLI

- Comamnd Line Interface
- PHP, mint általános scriptnyelv
- tipikusan nem webes célokra

CGI

- Common Gateway Interface
- OS szempontjából azonos a CLI-vel
- de facto szabvány: <http://www.w3.org/CGI>
- 1995-ös a legutóbbi stabil verzió (1.1)

MOD_PHP

- Apache API beépülő modul, bővítmény: <http://httpd.apache.org/docs/1.3/misc/API.html>
- jelenleg a PHP futtatásának legelterjedtebb módja
- az interpreter folyamatosan fut az Apache részeként (az interpretált scriptek nem!)
- Apache direktívákkal konfigurálható
- előnyös és hátrányos is egyben (lásd később...)
- A PHP scriptek sosem folyamatosan futnak! (kivéve CLI)
- A HTTP állapotmentes protokoll, vagyis:
 - kérés érkezik: a script elindul
 - a script nem ismeri a múltbeli kéréseket
 - a script nem ismeri a párhuzamosan futó scripteket

- kérés kiszolgálva: a script kilép
- A PHP válasza: KISS (Keep It Simple, Stupid)
- A Java válasza: application container

A PHP bemenete

- minden esetben lehet:
 - fájlrendszer
 - adatbázis
 - egyszerű kliensként hálózati kapcsolat
- CLI esetben lehet:
 - standard OS parancssor
 - standard futtatási környezeti változók (environment)
- CGI esetben lehet:
 - CGI futtatási környezeti változók (environment)
- MOD_PHP esetén lehet:
 - Apache szerver környezeti változók
- CGI és MOD_PHP esetében a környezet része:
 - a script neve, útvonala
 - a kérést kibocsátó gép címe
 - a kérés által használt csatorna (HTTP/HTTPS)
 - a webszerver által esetleg azonosított felhasználó neve, jelszava
 - a kéréshez tartozó protokoll-fejlécek:
 - POST paraméterek
 - GET paraméterek
 - COOKIE paraméterek

GET

- a GET paramétereket az URI hordozza
- `http://example.com/script.php?par1=ert1&par2=ert2`
 - http protokoll azonosító
 - :// protokoll elválasztó
 - example.com szerver azonosító (host)
 - / erőforrás azonosító elérési útvonal (path)
 - script.php erőforrás név
 - ? elválasztó az erőforrás és paraméterek közt
 - par1 az első paraméter neve, azonosítója
 - = elválasztó a paraméter és értéke közt
 - ert1 az első paraméter értéke
 - & elválasztó a paraméterek között

POST

- a POST adatok
 - a GET adatokhoz hasonlóan a konkrét kéréshez tartoznak
 - ugyanúgy név-érték párokból állnak
 - viszont nem „látszanak” a felhasználó számára, vagyis:
 - oldalújratöltéskor elveszhetnek (a browser hatásköre)
 - nem tárolhatók könyvjelzőként

COOKIE

- a COOKIE adatok

- nem „látszanak” a felhasználó számára
- a böngészőprogramhoz (browser) vannak rendelve
- lejárati idővel rendelkeznek (ez lehet végtelen is)
- nem kötelező őket elfogadni

A PHP kimenete

- minden esetben lehet:
 - fájlrendszer
 - adatbázis
- egyszerű kliensként hálózati kapcsolat(pl: más PHP programok felé: POST/GET/COOKIE)
- CLI esetben lehet:
 - OS standard kimenet (stdout)
- CGI és MOD_PHP esetén lehet:
 - a webszerver bemeneti csatornája
 - vagyis indirekt módon a webböngésző

Előnyök, felhasználási terület

- CLI:
 - általános célú parancsnyelv, batch-feldolgozás, stb...
 - weblapok karbantartási munkáihoz
 - ütemezett futtatáshoz (cron)
- CGI:
 - nem Apache webszerverekhez is használható
 - nem csak a webszerver nevében futhat (suphp, suexec)
 - elkülönülten fagy le, nem rántja a webszert magával
- MOD_PHP:
 - leggyorsabb üzemmód, mert az interpreter mindenkorban ott van
 - Apache virtual hostok, location-ök egyesével hangolhatók

Hátrányok

- CLI:
 - önmagában nehezen alkalmas webkiszolgálásra
 - tipikusan parancssori elérést igényel
 - pár web-specifikus szolgáltatás hiányzik (pl. session)
- CGI:
 - lassabb, mint a MOD_PHP
 - elavult technológia, nem hangolható annyira
- MOD_PHP
 - az egész webszert magával ránthatja
 - a webszerver jogosultságaival fut(nehéz pl. az egy gépen futó virtuális hosztok elválasztása)

A PHP Safe Mode

- Egy hibás/elkeseredett próbálkozás a szerver-oldali biztonság növelésére
- Lekorlátozza:
 - a fájműveleteket (csak saját fájlokra és könyvtárakra)
 - a hálózatelérést
 - az erőforráskorlátokat feloldó parancsokat
- A PHP6-ban már nem lesz benne

WAMP5

- <http://www.wampserver.com>
- Windows Apache MySQL PHP5
 - Apache 2.2.4
 - PHP 5.2.1 + PECL
 - SQLitemanager
 - MySQL 5.0.27
 - PHPMyAdmin
- webszerver PHP bővítménnyel
- adatbázis szerver
- konfigurációs és management WinXP applet
- több előretelepített PHP alkalmazás
- integrált telepítőkörnyezet

EasyEclipse for PHP

- <http://www.easyeclipse.org>
- PHP fejlesztéshez előrecsomagolt Eclipse
 - Eclipse integrált IDE (<http://www.eclipse.org>)
 - PHP Eclipse plugin (<http://phpeclipse.sourceforge.net>)
- szintaxis-kiemelés
- kódkiegészítés
- hibakeresés
- verziókövetés

Szoftver architektúra

- Windows XP (OS), de lehetne Unix, Mac OSX is
 - Apache (OS service, daemon)
 - PHP (Apache modul)
 - MySQL (OS service, daemon)
 - EasyEclipse for PHP (IDE)

Projektkezelés

- Eclipse oldalról:
 - integrált PHP projekt
 - névtér kezelés
 - webszervertől független
- PHP oldalról:
 - független szkriptfájlok halmaza
 - saját strukturálási lehetőségek
 - függhet a documentroot helyétől
- nem ragaszkodunk egyik oldalhoz sem

Hello World!

- lásd: <http://www.tricon.hu/~mcree/php/telepites>
- <? echo 'hello world!' ?>
- amire figyeljünk:
 - a fájlt a PHP értelmező csak az Apache-on keresztül futtatja le, egyébként a forráskódját fogjuk látni
 - az Apache csak a documentroot alatti területeket szolgáltatja, az Eclipse és a (futó) PHP

- kód viszont az egész gépet látja
- a fájlban csak egy nyitó <? és egy záró ?> jel legyen

A PHP nyelv alapjai

Alapvető szintaxis

- A PHP nyelvet a HTML fájlok kibővítésére terveztek
- HTML kompatibilis (kötelező) nyitó és záró tagok
 - <script language="php"> ... </script>
 - <?php ... ?>
 - <? ... ?> (rövidített stílus, opcionális)
 - <% ... %> (ASP stílus, opcionális)
- implicit echo tagok
 - <?= kifejezés ?>
 - <%= kifejezés ?>
- a tagok váltanak a HTML és PHP üzemmód közt
- ezt kihasználhatjuk a tömeges echo (print) utasítások kiváltására
- a végső záró tag opcionális
- utasítások elválasztása (kötelező):
 - ;
 - a záró tag magába foglal egy pontosvesszőt is
 - (mint Java, C és Perl)
- egysoros kommentezés:
 - // ...
 - # ...
 - sor végéig, vagy záró tagig érvényesek
 - (mint Java, C++, Perl, sh)
- többsoros kommentezés
 - /* ... */
 - (mint C)

Típusok

- négy skalár, egyértékű típus
 - boolean (logikai)
 - integer (egész szám)
 - float, double (lebegőpontos szám)
 - string (karaktersorozat)
- két kompozit, összetett típus
 - array (tömb)
 - object (objektum)
- két speciális típus
 - resource (erőforrás, pl. adatbázis kapcsolat)
 - NULL (az üres típus)
- a PHP lazán tipizált (loosely typed) nyelv!
- nincs szükség változók deklarálására
- implicit típuskonverziók a környezet függvényében
- létezik explicit típuskonverzió, de nem szükséges
- láasd a PHP kézikönyv P függelékét!(típuskonverziós táblázat)
- boolean (logikai adattípus)
 - értéke: „true” vagy „false” (kisbetű/nagybetű érzéketlen)
- integer (egész szám)

- megadható
 - oktális alakban (0-val kezdődik)
 - decimális alakban (nem 0-val kezdődik)
 - hexadecimális alakban (0x-el kezdődik)
- minden előjeles
- architektúrafüggő pontosság (pl.: 32bit vagy 64bit)
- a nem ábrázolható egész számok automatikusan lebegőpontossá alakulnak!
- float (lebegőpontos, valós számok)
 - megadható
 - tizedesponnal (12.34)
 - exponenciális alapkban (2E-10)
 - véges (platformfüggő, 64bites IEEE formátum)
 - ne bízz meg a törtszámok értékében az utolsó jegyig!
 - ne vizsgálj pontos egyenlőségre két lebegőpontos számot!
- string (karaktersorozat)
 - aposztrófos formában
 - nincs külön értelmezés
 - idézőjeles és heredoc formában
 - változók értelmezve
 - \ escape szekvenciák
 - összevonása . jelekkel
 - részkarakterek [] jelekkel
 - lásd: PHP kézikönyv és később!
- array (tömb)
 - array() az üres tömb
 - minden tömb asszociatív
 - hivatkozás [] jelekkel
 - kulcs megadása => jel előtt
 - érték akármi lehet
 - kulcs csak int és string
 - lásd: PHP kézikönyv és később!
- object (objektum)
 - OO paradigma
 - egy osztály példánya
 - létrehozása: new utasítás
 - hivatkozás -> jellel
 - lásd később!
- resource (erőforrás)
 - beépített függvényekkel hozható létre és azok igénylik
- NULL (az üres típus)
 - definiálatlan változók értéke
 - unset() függvény vagy NULL értékadás eredménye

Változók

- jelölése: \$ jellel (mint Perl, sh)
- „Egy érvényes változónév betűvel vagy aláhúzással kezdődik, amit tetszőleges számú betű, szám vagy aláhúzás követ” (PHP kézikönyv)
- a név kis/nagybetű érzékeny
- referencia változó: & jellel (mint C)
- változó változók léteznek, pl: \$\$a
- kétérterműségek feloldása: {} jelekkel

- deklarálás nem kötelező, de lehetséges (var kulcsszó)
- a változók hatásköre (scope) nem megszokott!
- három scope létezik:
 - lokális (függvényben)
 - globális (függvényeken kívül)
 - szuperglobális (lokális + globális)
- a globális változók alapértelmezésben nem látszanak a függvényekben!
 - globals kulcsszó
 - \$GLOBALS szuperglobális asszociatív tömb
- static kulcsszó: statikus változó függvényekben
- fontosabb szuperglobális asszociatív tömbök:
 - \$GLOBALS (globális változók)
 - \$_GET és \$_POST (lásd: PHP bemenete rész)
 - \$_COOKIE (lásd: PHP bemenete rész)
 - \$_SESSION (munkamenet, lásd: később)
- konstansok (csak skalár értékkel!)
 - definiálása: define(nev, ertek) függvény
 - elérése: névvel
 - PHP-ben minden nem definiált konstans értéke a neve!

Kifejezések

- kifejezés: „valami, aminek értéke van”
- a PHP számára majdnem minden kifejezés
- fontosabb példák:
 - függvények (function kulcsszó)
 - növelő és csökkentő operátorok (++, --)
 - összehasonlító kifejezések (==, !=, <, >, stb...)
 - értékadás (\$a=\$b=42, \$a+=2, stb...)
 - a háromoperandusú feltételes operátor (bool ? val1 : val2)

Operátorok

- operátor: kifejezésből egy másik kifejezést állít elő
- operandusok száma: unáris, bináris, trenáris
- asszociativitás
 - jobbról balra, pl.: \$a=\$b=42
 - balról jobbra, pl.: \$a=\$b+\$c-42
 - nem köthető, pl. ilyen nincs: 1<\$a<5
- precedencia
 - mint általában más prog. nyelveknél (C, Perl, Java)
 - ha bizonytalan vagy, zárójelezz!
 - lásd: PHP kézikönyv!
- aritmetikai (+, -, *, /, %)
- hozzárendelő (=)
- bitorientált (&, |, ^, ~, <<, >>)
- összehasonlító (==, ===, !=, <>, !==, <, >, <=, >=)
- hibakezelő (@)
- végrehajtó (`)
- növelő, csökkentő (++, --)
- logikai (and, or, xor, !, &&, ||)
 - az and és or szöveges változatának precendenciája kisebb!
- string (., .=)

- tömb (+)
- típus (instanceof)

Vezérlési szerkezetek

- csoportosítás {} jelekkel
- feltételes végrehajtás
 - if, else, elseif, switch
- ciklusok
 - while, do-while, for, foreach, break, continue
- declare (futási paraméterek beállítása)
 - jelenleg csak időmérésre
- return (visszatérés függvényből)
- külső fájlok beillesztése
 - require, include, require_once, include_once

Függvények

- function kulcsszóval (sőt: utasítással!) definiáljuk
- feltételes deklaráció lehetséges (lásd példák!)
- definiálásuktól fogva globálisan elérhetők
- nincs polimorfizmus
- nem lehet függvényt újradefiniálni, megszüntetni
- mély rekurzió (100-200 szint felett) nem ajánlott!
- függvényargumentumok
 - csak névvel kell megadni (nem erősen típusos nyelv)
 - alapértékük megadható (csak konstans lehet)
 - változó számú argumentum is lehetséges

PHP5 OO alapok

- hibrid OO paradigmá (strukturált és OO egyszerre)
- osztály deklaráció: class kulcsszó
- osztály példányosítás (objektum létrehozás): new
- egyszeres öröklés: extends kulcsszó
 - az ősosztály azonosítója: parent
- __construct és __destruct
 - nincs automatikus ősmetódus-hívás
- hívó objektumpéldányra hivatkozás: \$this
- aktív osztályra hivatkozás: self
- dinamikus hivatkozás: -> operátor
- statikus hivatkozás: :: operátor
- mezők definíciója: public/private/protected kulcsszó
- metódusok definíciója: function kulcsszó
- metódusok és mezők láthatósága, attribútumai:
 - public: bárhonnan elérhető
 - private: csak a definiáló osztályból
 - protected: csak öröklő- és szülőosztályokból
 - final: öröklőosztályokban nem felülírható
 - static: csak statikusan érhető el (nincs ->)
 - const: statikus konstans érték
 - abstract: elvont, nem definiált függvénytörzs
 - a tartalmazó osztály nem példányosítható

- ez a kulcsszó osztályra is alkalmazható
- interfések definiálása: interface kulcsszó
 - csak publikus metódusokat tartalmazhat
 - implementálása: implements kulcsszó
- mező és metódus overloading
 - __set(), __get(), __isset(), __unset(), __call() metódusok
- foreach() iteráció lehetséges a publikus mezőkön
 - lásd még: Iterator beépített interfész, SPL bővítmény
- egyéb mágikus metódusok
 - pl.: __toString, __sleep, __wakeup, __clone, __autoload
 - minden __jelekkel kezdődnek, ez fenntartott név
 - bővebben lásd: PHP kézikönyv és később
- objektumok klónozása: clone kulcsszó
 - alapértelmezés: shallow copy (a referenciakat meghagyja)
 - lásd még: __clone mágikus metódus
- objektumok tartalmi összehasonlítása: ===
- objektumpéldányok azonosságának vizsgálata: ===
- kivételkezelés (mint Java):
 - try { ... } catch (Exception \$e) { ... }
 - a beépített függvények egyelőre nem dobnak exception-t
- van reflection API is (mint Java)
- type hinting: metódusok argumentumának típuskényszerítése (csak osztály és tömb lehet)

Referenciák

- a PHP „szeret” mindenöt mindenöt másolni
 - értékadáskor
 - hivatkozáskor
 - paraméterátadáskor
 - stb...
- PHP referencia: „egy tartalom több néven érhető el”
- nem olyan, mint C-ben a pointer!
- inkább a UNIX hardlinkekhez hasonlít
- a referenciákat & jellet adjuk meg
- referenciát használhatunk
 - függvényparaméterek megadásánál, pl.: novel(\$a)
 - referencia visszaadására, pl.: \$modositando=&keres()
 - példányosításnál, pl.: \$obj = &new Osztaly()
 - memória- és sebességoptimalizáláshoz
- ne használunk referenciát
 - összetett tömbökre való hivatkozásra (inkább másolódnak)
 - referenciák másolására (a referencia csak egy név!)
- alapértelmezett referenciák PHP-ban:
 - global kulcsszó: & \$GLOBALS['kulcsszó']
 - \$this mindenöt az adott objektumon dolgozik

Biztonság

- a PHP egy sokszínű, funkciógazdag, elterjedt nyelv
- megfelelő korlátozásokkal kell élnünk, hogy ne használhassák ellenünk
- minimális jogosultságok az éles rendszerben
- ne maradjanak debug információk, error reporting off
- sose adjuk ki a script forrását a webre

- sose bízzunk meg felhasználótól érkező adatokban
 - register globals (már a múlté)
 - sql injection (speciálisan szerkesztett escapeelt bemenetek)
 - magic quotes (egy újabb gyenge próbálkozás)
- lásd: PHP kézikönyv, biztonság fejezete

A PHP API áttekintése

Tömbök

- Teljes referencia: PHP kézikönyv V. fejezet
- ismétlésként, a PHP tömbök minden:
 - asszociatívak (kulcs-érték párokkal dolgoznak)
 - praktikusan „korlátlan”, változó méretűek
 - definiálnak egy alapértelmezett végigjárási sorrendet
- lásd: mellékelt forráskódok!

Dátum és idő

- PHP kézikönyv X. és XXII. fejezet
- A PHP időszámítási alapja:
 - Unix Epoch (1970. január 1. 00:00:00 GMT)
 - A dátumok egy másodperc és egy időzóna információt tartalmaznak
- Lásd: a mellékelt forráskódok!

Fájlok és könyvtárak

- PHP kézikönyv XXIX. és XL. fejezet
- gyakorlatilag felület az O/S funkciókhoz
- sok függvény URL-eken is működik!
- lásd: a mellékelt forráskódok!

SMTP

- PHP kézikönyv LXXV. és LXIII. fejezet
- A PHP külső programmal és saját erőből is képes levelek küldésére
- A levelek MIME törzsét a programozónak kell megadnia
- lásd: a mellékelt forráskódok!

SQL gyorstalpaló

- Structured Query Language programozási nyelv
- adatok központi strukturált tárolására és elérésére
- alapvető strukturális elemek:
 - adatbázis (táblákat tartalmaz)
 - tábla (rekordokat/sorokat tartalmaz)
 - rekord/sor (típusos adatmezőket tartalmaz)
 - adatmező (jellemzően egy skaláris adatot tartalmaz)
- tábla létrehozása:
- CREATE TABLE név (mező1 típus1,)
- a legyakoribb típusok:
 - TEXT, VARCHAR(bájhossz) – szöveg
 - INT(bájhossz) – szám
 - TIMESTAMP – dátum, időbélyeggel

- adatok beszúrása:
- INSERT INTO táblanév (mező1, mező2, ...) VALUES (érték1, érték2 ...)
- az értékekre biztonsági szempontból figyelni kell!
- a típusok közti különbségek néha gondot jelentenek!
- adatok lekérdezése:
- SELECT mező1, mező2, ... FROM tábla1, tábla2,[WHERE feltétel1, feltétel2, ...][ORDER BY mező3, mező4, ... ASC/DESC][GROUP BY mező5, mező6, ...][LIMIT sorok száma][OFFSET sorok száma]
- egy select több táblát kapcsolhat össze

SQL

- PHP kézikönyv CXVIII. és sok más fejezet
- A PHP rengeteg adatbázisrendszeret támogat
- kezdetben: minden adatbázishoz külön API
- PHP 5.1 óta: PDO (PHP Data Objects)
 - uniform objektumorientált API
 - számos adatbázis driver támogatása (ODBC is)
- lásd a mellékelt forráskódokat!

Stringkezelés

- PHP kézikönyv CLIV., XCII., CLX., CLXI. fejezet
- a stringek önálló skalárnak számítanak
- felemás multibyte támogatás!
- lásd a mellékelt forráskódokat!

Reguláris kifejezések

- PHP kézikönyv CXVI. és CXXI. fejezet
- a PHP Perl és Posix stílusú kifejezéseket támogat
- a Perl stílusúak (PCRE) általában „ügyesebbek”
- számtalan felhasználási terület
- lásd a mellékelt forráskódokat!

PCRE gyorstalpaló

- általában minden karakter önmagát jelenti, de vannak speciális karakterek:
 - . bármilyen karaktert helyettesít
 - * a megelőző minta nulla vagy több előfordulása
 - + a megelőző minta egy vagy több előfordulása
 - ? mohóság (greediness) kikapcsolása
 - [] karakterosztályok megadása
 - () részminta megadása
 - / mintahatár
 - \ escape
 - ^ sor eleje
 - \$ sor vége

PHP tervezési minták

HTML FORM

- POST és GET metódus (lásd: bevezető!)

- <FORM> és </FORM> tagek zájják közre
- számos beviteli elem közül választhatunk:
 - text: sima szöveg
 - password: rejttett megjelenítésű szöveg
 - checkbox: jelölőnégyzet, választómező
 - radio: rádiógomb, választócsoporthoz
 - submit: formot beküldő gomb
 - textarea: többsoros szöveg
 - select: legördülő menü és többes opcióválasztás
 - hidden: webes felhasználó elől rejttett adat
- <INPUT> beviteli elemek attribútumai:
 - type – típus
 - text, password, checkbox, radio, submit, hidden
 - name – beviteli mező azonosítására szolgáló név
 - radio esetén a csoportot is azonosítja
 - value – alapértelmezett szöveg
 - submit esetén a gomb címkéje
 - radio es checkbox esetén nincs hatása
 - checked – alapértelmezett állapot
 - nem kell értéket megadni, ha szerepel azt jelenti, be van jelölve
 - csak checkbox és radio
 - disabled – felhasználói bevitel engedélyezett-e
 - ha szerepel, a beviteli mező letiltásra kerül
 - nem bízhatunk abban, hogy a böngésző ezt betartja!
- <TEXTAREA> tag attribútumai:
 - cols, rows – oszlopok és sorok száma
 - name – beviteli mezőt azonosító név
 - disabled – bevitel engedélyezése a felhasználó számára
 - a textarea alapértéke: a bezáró </TEXTAREA> tagig
- <SELECT> tag attribútumai:
 - name – beviteli mezőt azonosító név
 - disabled – bevitel engedélyezése a felhasználó számára
 - multiple – többes választási lehetőség bekapcsolása
 - nem kell értéket megadni, ha szerepel érvénybe lép
 - a select értékei: a bezáró </SELECT> tagig megadott <OPTION tagekben>
- <OPTION> tag attribútumai:
 - value – az opció értéke
 - selected – az opció alapértelmezett állapota
 - nem kell értéket megadni, ha szerepel az opció kijelölésre kerül
 - az opció nevének megadása: </OPTION> tagig
- lásd a mellékelt forráskódokat!

Formkezelés és ellenőrzés

- a beküldött formok adatai: \$_POST és \$_GET szuperglobális asszociatív tömbök
- a tömbök kulcsai: a formelemek name attribútumai
- a name attribútumok speciális használata:
 - ha a név []-re végződik, automatikusan tömb készül belőle
 - ha a név [kulcs]-ra végződik a kulcs lesz a tömbindex
- lásd a mellékelt forráskódokat!

Formok (újra)feltöltése

- mikor van erre szükség?
 - dinamikusan alapértelmezett értékeket akarunk megadni
 - felhasználó által „elrontott” form újrabekérésekor
- minden ismeretett formmezőnél megadható alapérték
- az alapértékek kitöltésekor ügyeljünk:
 - a speciális karakterekre
 - újrakötöltéskor a PHP magic quotes szolgáltatásra
- lásd a mellékelt forráskódokat!

Többnyelvűség

- mikor van erre szükség?
 - ugyanazt a formot akarjuk többször más nyelven megadni
 - ez a minta nem csak formoknál lehet hasznos
- jellemzően GET paraméterrel befolyásoljuk a nyelvet
- lásd a mellékelt forráskódokat!

Fájlfeltöltés kezelése

- csak POST adatként, speciális multipart formban
- saját <INPUT> type attribútum: file
- a feltöltés eredménye a \$_FILES szuperglobálisban
- a korlátokról lásd a PHP kéziköny 38. fejezetét!
- lásd a mellékelt forráskódokat!

Munkamenetek

- lásd: PHP kézikönyv, CXLII. fejezet!
- a HTTP állapotnélküliségek megszüntetése a cél
- középtávú adattárolásra alkalmas
- alapértelmezésben merevlemezen tárolódik
- \$_SESSION szuperglobális asszociatív tömb
- a PHP script indítása után feltöltjük
 - session_start()
- a script futása után elmentjük
 - automatikusan megtörténik, vagy session_write_close()
- munkamenet törlése: session_destroy()
- lásd a mellékelt forráskódokat!

Cookie

- lásd: PHP kézikönyv XCVIII. fejezet!
- a böngésző számára adatokat küldhetünk ki
- a kiküldött adatokat ezután minden kérésnél megadja
- lejárati idővel rendelkező kulcs-érték párok
- a munkamenet-kezelés is ezt használja ha elérhető
- nem kényes adatok hosszú távú tárolására alkalmas
- lásd a mellékelt forráskódokat!

HTTP Hitelesítés

- az azonosítást a webböngésző végzi
- a cookie-hoz hasonlóan minden küldi az adatokat

- a realm-be tartozó felhasználónév és jelszó
- azonosítási információk a \$_SERVER szup.glob.-ban
- azonosítás kezdeményezése: header() függvényvel
 - WWW-Authenticate: Basic realm="Azonosítsd magad!"
 - HTTP/1.0 401 Unauthorized
 - realm: az azonosítási terület
 - Basic: az azonosítás típusa (lehetne még: Digest)
- lásd: PHP kézikönyv 34. fejezet!
- lásd a mellékelt forráskódot!

Cookie alapú hitelesítés

- az azonosítást a PHP program végzi FORM-al
- siker esetén cookie-ba tárolja az információkat
- a „realm” a cookie domainje
- több felelősség a programozón
- lásd: korábbi cookie példák
- lásd a mellékelt forráskódot!

Template rendszerek

- A PHP programok jellemzően vegyesek
 - megjelenítési utasítások, nézet
 - műveletek, „business logic”, logikai modell
 - felhasználói input feldolgozása, felhasználói kontroll
- A felsorolt három terület jobbára elkülöníthető
- A nézet (view)
 - a felhasználó ezt „látja”, tehát gyakran testre kell szabni
 - jellemzően sok HTML kódot tartalmaz
 - a PHP változóból leginkább adatokat nyer ki
 - gyakran hasonló adatokat másként kell megjeleníteni
 - gyakran egy megjelenítéshez más-más adatok tartoznak
 - tehát: az adatok logikai modelljével nem annyira töröklik
- Az adatok és rajtuk végzett műveletek (model)
 - az adatok gyakran adatbázisból származnak
 - a műveleteket gyakran adatbázison kell végrehajtani
 - a műveletek gyakran jellemző mintákat mutatnak
 - célszerű a kód újrafelhasználása (OO paradigma)
 - az adatok és műveletek a megjelenítéstől függetlenek
- Felhasználói input és műveletvégzés (controller)
 - jellemzően az adatokon akar műveleteket indítani
 - PHP esetén itt történik az előfeldolgozás (pl. stripslashes)
 - a HTTP protokoll állapotmentességét itt kell kompenzálni
- a model-view-controller (MVC): ismert paradigma
- minden GUI alkalmazásnál, így a PHP-nál is hasznos
- az MVC vezérlése ciklikus, négy szereplős:
 - a felhasználó kérést bocsát ki
 - a controller feldolgozza a kérést és értesíti a modellt
 - a model műveleteket végez és értesíti a view-t
 - a view a kért formában megjeleníti a modellt
 - a felhasználó megkapja kérésének eredményét
- MVC megvalósítási kérdések PHP esetén
 - a felhasználó egy browserrel dolgozik

- a használt protokoll állapotmentes
- a browser oldalon használt technológiák
 - Cookie, HTTP authentikáció, formok, JavaScript
- controller technológiák
 - session kezelés, \$_POST, \$_GET, stb... feldolgozása
- model technológiák
 - strukturált programozás, OOP, adatbáziskezelés, sessionök
- view technológiák
 - XSLT, skin/template frameworkök (pl.: smarty)
- lásd a mellékelt forráskódot!

Adatbázis alapú funkciók

- A PHP scriptek egymás közti kommunikációja
 - nem triviális, mert a scriptek nem folyamatosan futnak
 - az interakció nehézkes (állapotnélküli környezet)
 - legegyszerűbb megoldás: közös adatbázison keresztül
- Lásd a korábbi adatbázis részt!
- Lásd a mellékelt forráskódot!

Dinamikus képek

- A PHP program kimenete nem csak HTML lehet!
- Lásd: PHP kézikönyv LXII. fejezet!
- A dinamikus képek felhasználási területei
 - on-line grafikonok
 - captcha (robotok kiszűrésére)
 - dinamikus design célokra
- Lásd: a mellékelt forráskódot!

Melléklet: forráskódok

Az alábbiakban a gyakorlatokon példaként ismertetett forráskódokat olvashatjuk. A fontos, kiemelni kívánt aspektusok szabványos PHP komment jelöléssel szerepelnek, ezzel is könnyíteni kívánjuk a kódok közvetlen kipróbálását. Az „offline” felkészülést elősegítendő a példaprogramok futási kimenetét is megadtuk.

_feladatok/alap2 - 010 fibonacci.php

```
<?
// feladat: készítsen egyszerű PHP programot, mely
// a kimeneten tetszőleges lépésekben képes megjeleníteni
// az ún. fibonacci számsorozatot, melyben a soronkövetkező
// elem minden az azt megelőző két elem összegével egyenlő.
//
// egy példa a helyes kimenetre: 0 1 1 2 3 5 8 13 21 ... stb...
//
// pluszfeladat: oldja meg, hogy a program minden
// soronkövetkező elemre jelenítse meg új sorban az adott
// elemig kiszámított teljes sort.
// egy példa a helyes kimenetre:
// 0
// 0 1
// 0 1 1
// 0 1 1 2
// 0 1 1 2 3
// stb...

// egy lehetséges rekurzív megoldás:
function osszead($a,$b,$lepes) {

    echo $a+$b."<BR>";
    $lepes++;

    if($lepes>10) return;
    else osszead($b,$a+$b,$lepes);

}

osszead(0,1,1);

?>
```

A futtatás eredménye:

```
1
2
3
5
8
13
21
```

34
55
89

_feladatok/api1 - 010 oszthatosagi szures.php

```
<PRE>
<?
// feladat: készítsen PHP programot, mely előállít egy tömböt,
// melyben a pozitív egész számok szerepelnek 1-től 100-ig.
// Készítsen függvényt, mely egy megadott tömb egy adott
// egész paraméterrel maradék nélkül osztható elemeit NEM tartalma
// zó
// tömbbel tér vissza. Az eredményeket jelenítse meg!
//
// példa futási eredmény:
// A tömb: 1 2 3 4 5 6 7 8 9 ... stb...
// 2-vel osztható elemek kiszűrése után: 1 3 5 7 9 ... stb...
// 3-al osztható elemek kiszűrése után: 1 5 7 11 13 17 ... stb...
//
// pluszfeladat: használja fel az előállított szűrőfüggvényt az
// 1 és 1000 közti prímszámok megkeresésére és megjelenítésére!
//
// példa futási eredmény:
// 1 3 5 7 11 13 ... stb...

// egy lehetséges megoldás a PHP beépített függvényei segítségével
:
$tomb = range(1,100); // PHP beépített függvény
print_r($tomb);

// segédfüggvény a szűréshez, mindenkor a globális n paraméter alapjá
n
// dönti el egy adott számról, hogy bekerülhet-e a tömbbe
function oszthato($n) {
    return ($n % $GLOBALS["n"] == 0) ? FALSE : TRUE;
}

// a feladat által kért szűrőfüggvény
function szur($tomb, $szam) {
    $GLOBALS["n"]=$szam; // globális szűrési paraméter beállítása a
    // segédfüggvény számára
    return array_filter($tomb,'oszthato'); // PHP beépített szűrőfüg
    gvény
}

$tomb=szur($tomb, 2); // 2-vel osztható elemek kiszűrése a tömbből
print_r($tomb);
$tomb=szur($tomb, 3); // 3-al osztható elemek kiszűrése a tömbből
print_r($tomb);
```

?>

A futtatás eredménye:

Array

```
(  
    [0] => 1  
    [1] => 2  
    [2] => 3  
    [3] => 4  
    [4] => 5  
    [5] => 6  
    [6] => 7  
    [7] => 8  
    [8] => 9  
    [9] => 10  
    [10] => 11  
    [11] => 12  
    [12] => 13  
    [13] => 14  
    [14] => 15  
    [15] => 16  
    [16] => 17  
    [17] => 18  
    [18] => 19  
    [19] => 20  
    [20] => 21  
    [21] => 22  
    [22] => 23  
    [23] => 24  
    [24] => 25  
    [25] => 26  
    [26] => 27  
    [27] => 28  
    [28] => 29  
    [29] => 30  
    [30] => 31  
    [31] => 32  
    [32] => 33  
    [33] => 34  
    [34] => 35  
    [35] => 36  
    [36] => 37  
    [37] => 38  
    [38] => 39  
    [39] => 40  
    [40] => 41  
    [41] => 42  
    [42] => 43  
    [43] => 44  
    [44] => 45  
    [45] => 46  
    [46] => 47  
    [47] => 48  
    [48] => 49  
    [49] => 50  
    [50] => 51  
    [51] => 52  
    [52] => 53  
    [53] => 54
```

```
[54] => 55
[55] => 56
[56] => 57
[57] => 58
[58] => 59
[59] => 60
[60] => 61
[61] => 62
[62] => 63
[63] => 64
[64] => 65
[65] => 66
[66] => 67
[67] => 68
[68] => 69
[69] => 70
[70] => 71
[71] => 72
[72] => 73
[73] => 74
[74] => 75
[75] => 76
[76] => 77
[77] => 78
[78] => 79
[79] => 80
[80] => 81
[81] => 82
[82] => 83
[83] => 84
[84] => 85
[85] => 86
[86] => 87
[87] => 88
[88] => 89
[89] => 90
[90] => 91
[91] => 92
[92] => 93
[93] => 94
[94] => 95
[95] => 96
[96] => 97
[97] => 98
[98] => 99
[99] => 100
)
Array
(
    [0] => 1
    [2] => 3
    [4] => 5
    [6] => 7
    [8] => 9
    [10] => 11
    [12] => 13
    [14] => 15
    [16] => 17
    [18] => 19
    [20] => 21
    [22] => 23
```

```
[24] => 25
[26] => 27
[28] => 29
[30] => 31
[32] => 33
[34] => 35
[36] => 37
[38] => 39
[40] => 41
[42] => 43
[44] => 45
[46] => 47
[48] => 49
[50] => 51
[52] => 53
[54] => 55
[56] => 57
[58] => 59
[60] => 61
[62] => 63
[64] => 65
[66] => 67
[68] => 69
[70] => 71
[72] => 73
[74] => 75
[76] => 77
[78] => 79
[80] => 81
[82] => 83
[84] => 85
[86] => 87
[88] => 89
[90] => 91
[92] => 93
[94] => 95
[96] => 97
[98] => 99
)
Array
(
    [0] => 1
    [4] => 5
    [6] => 7
    [10] => 11
    [12] => 13
    [16] => 17
    [18] => 19
    [22] => 23
    [24] => 25
    [28] => 29
    [30] => 31
    [34] => 35
    [36] => 37
    [40] => 41
    [42] => 43
    [46] => 47
    [48] => 49
    [52] => 53
    [54] => 55
    [58] => 59
```

```
[60] => 61
[64] => 65
[66] => 67
[70] => 71
[72] => 73
[76] => 77
[78] => 79
[82] => 83
[84] => 85
[88] => 89
[90] => 91
[94] => 95
[96] => 97
)

_feladatok/api1 - 020 datumok.php
```

```
<?
// feladat: készítsen programot mely meghatározza, a hét melyik napjára
// esik az Ön születésnapja!
//
// egy lehetséges futási eredmény az 1979-09-10 dátummal:
// hétfő
//
// pluszfeladat: a program azt is adja meg, a megadott évben mely
// napra esett húsvét! Segítségül Meeus Julián algoritmusa:
// Y = évszám
// a = Y mod 4
// b = Y mod 7
// c = Y mod 19
// d = (19 * c + 15) mod 30
// e = (2 * a + 4 * b - d + 34) mod 7
// hónap = (d + e + 114) / 31
// nap = ((d + e + 114) mod 31) + 1
//
// egy lehetséges futási eredmény:
// április 8.
setlocale(LC_TIME,"hu_HU.UTF-8");
$ts = strtotime("1979-09-10");
echo strftime("%A",$ts);
?>
```

A futtatás eredménye:

hétfő

_feladatok/api1 - 030 fajlkezelés.php

```
<?
/*
feladat: készítsen PHP programot, mely egy adott könyvtárban
található összes fájl méretét összeadjá és a felhasználó által
könnyen érhető formában megjeleníti.
egy lehetséges megoldás:
12.34 kilobyte
pluszfeladat: a program ne csak az adott könyvtárban, hanem
az alkönyvtáraiban is számolja össze a fájlok méretét!
pluszfeladat: a porgram számolja össze a könyvtárban található
fájlok sorainak számát is!
egy lehetséges megoldás:
```

```
1.23 megabyte
*/
// egy lehetséges megoldás az aktuális könyvtárra:
// olvasható bájt méret visszaadása egy stringben
function olvashatomeret($Bytes) {
    $Type=array("", "kilo", "mega", "giga", "tera", "peta", "exa", "zetta", "yott
a");
    $Index=0;
    while($Bytes>=1024) { // amíg osztható 1024-el
        $Bytes/=1024; // maradékosan osztjuk
        $Index++; // növeljük a névindexet
    }
    $Bytes = (round($Bytes*100)/100.0); // kettő tizedesjegyre kerekített érték
    return("".$Bytes.".{$Type[$Index]}."byte");
}
$oszz=0;
foreach(scandir(".") as $file) { // végigfut a könyvtár elemein
    $stat = stat($file); // fájlinformációk lekérése
    $oszz+=$stat['size']; // összméret növelése
}
echo olvashatomeret($oszz);
?>
```

A futtatás eredménye:

24.78 kilobyte

feladatok/api2 - 010 adatbaziskezeles.php

```
<PRE>
<?
/*
feladat: készítsen adatbázisalapú programot, mely minden látogató
számára megjeleníti, hogy az általa használt IP címről eddig
hány alkalommal tekintették meg a weblapot!
példa a kimenetre:
az ön IP címe: 127.0.0.1, erről a címről eddig 15 alkalommal néztek meg.
pluszfeladat:
jelenítse meg a 10 leggyakoribb IP címet sorrendben, valamint tárolja
és jelenítse meg a legutóbbi látogatások dátumát is!
példa a kimenetre:
az ön IP címe: 127.0.0.1, erről a címről eddig 15 alkalommal néztek meg,
legutóbb 1956 január 2. 19 óra 33 perckor.
A 10 leggyakoribb látogató:
127.0.0.1 (15 látogatás)
127.0.0.2 (13 látogatás)
...
...
*/
// egy lehetséges megoldás
// opcionális hibakezelő blokk eleje
try {
    // csatlakozás SQLite adatbázishoz
    // példa: csatlakozás MySQL adatbázishoz:
    // $db = new PDO('mysql:host=localhost;dbname=adatbazis', $felhasznalo, $jelsz
o);
    $db = new PDO('sqlite:latogatok.sqlite');
    // adatbázis tábla eldobása
    //$db->exec("drop table latogatok");
    // adatbázis tábla létrehozása
    // a tábla neve: kattintasok
```

```
// két mezőt tartalmaz, egy ip nevű (szöveges) és egy mikor nevűt (időbelyeg)
$db->exec("create table latogatok (ip text)");
// beszúró utasítás előkészítése, az :ipje paraméter használatával
$stat=$db->prepare("insert into latogatok (ip) values (:ipje)");
// az :ipje paraméter feltöltése a PHP scriptet megtekintő felhasználó IP címével
$stat->bindValue(':ipje', $_SERVER['REMOTE_ADDR'], PDO::PARAM_STR);
// beszúró utasítás lefuttatása
$stat->execute();
$stat=$db->prepare("select count(ip) as latogatasok from latogatok where ip = :aktualip");
$stat->bindValue(':aktualip', $_SERVER['REMOTE_ADDR'], PDO::PARAM_STR);
$stat->execute();
$res = $stat->fetch();
echo "az ön IP címe: ".$_SERVER['REMOTE_ADDR'].", erről a címről eddig ".$res[ "latogatasok"]." alkalommal néztek meg.";
} catch (Exception $e) { // opcionális hibakezelő blokk vége
    echo "Hiba: " . $e->getMessage(); // hibaüzenet megjelenítése
}
?>
```

A futtatás eredménye:

az ön IP címe: 195.70.57.4, erről a címről eddig 0 alkalommal néztek meg.

feladatok/api2 - 020 stringek.php

```
<?
/*
feladat: készítsen programot, mely egy hosszabb szövegben megszámlálja a szavak számát, majd a 3 karakternél nagyobb szavakból új szöveget alkot melyben a szavakat szóközök választják el.
Szövegforrásnak használhatja a www.lipsum.org webhelyet.
példa futási eredmény az "ez egy string, lehetne!" bemenetre:
4 szó
string lehetne
pluszfeladat: a program számolja és őrizze meg a punktuációs karaktereket.
példa futási eredmény az "ez egy string, lehetne!" bemenetre:
4 szó, 2 punktuációs karakter
string, lehetne
*/
$szoveg=<<<VEGE
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas at nisi. Cura bitur posuere condimentum neque. Praesent sollicitudin. Aliquam quam est, pulvin ar non, convallis ac, ornare vitae, dolor. Sed metus nisi, laoreet eu, molestie eu, aliquet id, velit. Quisque lorem lorem, varius quis, vehicula at, ultricies quis, massa. Etiam porta nisl.
VEGE;
$nyers=str_replace(array(",",".",",","?",";",":","-"),array(),$szoveg); // punktuációs karakterek törlése
$szavak=explode(" ",$nyers); // robbantás a szóközök alapján
echo count($szavak)." szó<BR>"; // robbantott tömb méretének megjelenítése
$ujszavak=array(); // üres tömb az új szavaknak
foreach($szavak as $szo) { // ciklus a szavakon
    if(strlen($szo)>3) { // ha a szó hossza nagyobb, mint 3
        $ujszavak[]=$szo; // hozzáfűzzük a kimeneti tömbhöz
    }
}
echo implode(" ",$ujszavak); // a kimeneti tömb összefűzése és megjelenítése
?>
```

A futtatás eredménye:

50 szó

Lorem ipsum dolor amet consectetur adipiscing elit Maecenas nisi Curabitur posuere condimentum neque Praesent sollicitudin Aliquam quam pulvinar convallis ornare vitae dolor metus nisi laoreet molestie aliquet velit Quisque lorem lorem varius quis vehicula ultricies quis massa Etiam porta nisl

feladatok/pat1 - 010 formkezeles.php

```
<?
/*
Feladat: készítsen egyszerű adatlekérdező formot, melyen
a felhasználó megadhatja nevét és nemét. A form feldolgozása
során ellenőrizze, hogy a név és nem megfelelően kitöltésre
került-e, ha nem, jelenítse meg a formot a felhasználó által
már kitöltött adatok alapértelmezett értékként való megadásával
és a hiányosságok megjelölésével. Ha minden mező megfelelő,
jelenítsen meg rövid üdvözlő üzenetet, melyben a felhasználót
nevének és nemének megfelelően köszönti! Ha a form még
nem került elküldésre, ne jelenítsen meg hibaüzenetet!
Egy lehetséges futási eredmény helyes formkitöltés esetén:
Tiszttelt Rigó Ernő úr! Köszönjük, hogy kitöltötte a formot!
Egy lehetséges futási eredmény helytelen kitöltés esetén:
Kérém válassza ki a nemét!
Név: Rigó Ernő_____
Nem: ( ) férfi ( ) nő
[küldés]
Pluszfeladat: Hiba esetén emelje ki pirossal azokat a mezőket
amelyek nem megfelelően lettek megadva! Bővítsse a formot, hogy
e-mail címet és születési dátumot is meg lehessen adni,
végezze el az email cím és a dátum formális ellenőrzését is!
A dátum beviteléhez ajánlatos legördülő menüket felvenni külön
az év, hónap és nap értékekre, így szűkíthető a hibalehetőségek
sora. Egészítse ki a formot többnyelvű megjelenítési és
fájlfeltöltési lehetőséggel!
*/
// egy lehetséges megoldás:
?>
<FORM method="POST">
    Név:<INPUT type="text" name="nev" value=<?=$_POST["nev"]?>><BR>
    Nem:férfi <INPUT type="radio" name="nem" value="f" <?=$_POST["nem"]=="f"? "checked": ""?>>
    nő <INPUT type="radio" name="nem" value="n" <?=$_POST["nem"]=="n"? "checked": ""?>><BR>
    <INPUT type="submit" name="kuldt" value="küldés">
</FORM>
?>
if($_POST["kuldt"]) {
    if(strlen($_POST["nev"])==0) {
        echo "Kérém adjon meg a nevét!";
        exit;
    }
    if(strlen($_POST["nem"])==0) {
        echo "Kérém válassza ki a nemét!";
        exit;
    }
    echo "Tiszttelt $_POST[nev] ".($_POST["nem"]=="f"? "úr": "úrhölgy")."! Köszönjük,
    hogy kitöltötte a formot!";
}
?>
```

A futtatás eredménye:

Név:

Nem:férfi
nő

_feladatok/pat2 - 010 sessionkezeles.php

```
<?
/*
Feladat: készítsen négy alapműveletes számológép funkciót
megvalósító PHP programot, mely a műveletet, valamint két
operandusát egy FORMon keresztül kéri be! A program használjon
munkamenetet az összes bevitt számítás és eredményeik megjelenítésére!
Egy lehetséges futási eredmény (az első sor a formot hivatott ábrázolni):
Adja meg a műveletet: _____ [+] _____ [=]
Eddigi műveletek:
23452345 + 23452345 = 46904690
134 - 234235 = -234101
42352345 * 32452345 = 1.3744329115E+15
23452345 - 23452345 = 0
66666 + 66666666 = 66733332
Pluszfeladat: az eddigi műveleteket ne felsorolásként, hanem
egy legördülő menüben jelenítse meg és tegye lehetővé a
felhasználó számára, hogy bármelyik korábbi számításhoz visszatérve
módosítsa azt! A számításhoz való visszatérést úgy valósítsa meg,
hogy a legördülőben kiválasztott számítás adatai az alapműveletes
számológép formjába töltődjenek vissza alapértelmezett értékként!
Tegye lehetővé az eddigi összes művelet törlését!
Egy lehetséges futási eredmény:
Adja meg a műveletet: _____ [+] _____ [=]
Eddigi műveletek:
[23452345 + 23452345 = 46904690] [visszatér] [mindet töröl]
*/
// Egy lehetséges megoldás:
?>
<!-- a műveletet és operandusokat bekérő HTML form --&gt;
&lt;FORM method="POST"&gt;
    &lt;INPUT type="text" name="a"&gt;
    &lt;SELECT name="muvelet"&gt;
        &lt;OPTION value="add"&gt;+&lt;/OPTION&gt;
        &lt;OPTION value="sub"&gt;-&lt;/OPTION&gt;
        &lt;OPTION value="mul"&gt;*&lt;/OPTION&gt;
        &lt;OPTION value="div"&gt;/&lt;/OPTION&gt;
    &lt;/SELECT&gt;
    &lt;INPUT type="text" name="b"&gt;
    &lt;INPUT type="submit" name="eq" value="="&gt;
&lt;/FORM&gt;
&lt;?
// munkamenet indítása
session_start();
// a formot beküldte a felhasználó</pre>
```

```
// az = gomb lenyomásával
if(strlen($_POST["eq"])>0) {
    // a muvelet formparaméter függvényében
    // elvégezzük a műveletet és kiválasztjuk
    // a megfelelő műveleti jelet a megjelenítéshez
    switch($_POST["muvelet"]) {
        case "add":
            $eredmeny=$_POST["a"] + $_POST["b"];
            $jel="+";
            break;
        case "sub":
            $eredmeny=$_POST["a"] - $_POST["b"];
            $jel="-";
            break;
        case "mul":
            $eredmeny=$_POST["a"] * $_POST["b"];
            $jel="*";
            break;
        case "div":
            $eredmeny=$_POST["a"] / $_POST["b"];
            $jel("/");
            break;
    }
    // a számítás megjelenítési formájának összeállítása
    $szamitas=$_POST["a"] . " " . $jel . " " . $_POST["b"] . " = " . $eredmeny;
    // a számítás hozzáfűzése a munkamenethez
    $_SESSION["szamitasok"][]=$szamitas;
}
// az eddig munkamenetbe tárolt számítások megjelenítése
// egyszerű foreach ciklussal
foreach($_SESSION["szamitasok"] as $s) {
    echo "$s<BR>";
}
?>
```

A futtatás eredménye:

_feladatok/pat2 - 020 cookie.php

```
<?
/*
Feladat: készítsen cookie alapú PHP programot, mely
nevén nevezi és köszönti az oldalt meglátogató felhasználót!
Ha a felhasználó még nem ismert, kérdezze meg a nevét,
ha a felhasználó már ismert, tegye lehetővé, hogy az
oldal elfelejtse! Az oldal 30 napig emlékezzen a felhasználóra!
Példa futási eredmény ismeretlen felhasználó esetén:
Szia! Add meg neved: _____ [emlékezz rám!]
Példa futási eredmény ismert felhasználó esetén:
Szia Balamka Sámu... [felejts el!]
```

Pluszfeladat: a felhasználó a neve mellett adhasson meg egy jelszót is, melynek megadása legyen szükséges az elfelejtő funkció használatához!

Példa futási eredmény ismeretlen felhasználó esetén:

Szia! Add meg neved: _____ jelszavad: _____ [emlékezz rám!]

Példa futási eredmény ismert felhasználó esetén:

Szia Balamka Sámu! Jelszó: _____ [felejts el!]

```
/*
// Egy lehetséges megoldás:
// ha a névkérő FORMról adat érkezett
if(strlen($_POST["nev"])>0) {
    // COOKIE beállítása a FORMon megadott névvel, 30 nap lejárattal
    setcookie("felhasznalo",$_POST["nev"],time()+3600*24*30);
    // a változások "megelőlegezéseként" a $_COOKIE szuperglobális
    // már most feltöltjük a megadott névvel. Erré azért van szükség,
    // mert egyébként csak a setcookie függvény hívása utáni futáskor
    // lennének láthatók az értékek. Lásd: PHP kézikönyv XCVIII. fejezet!
    $_COOKIE["felhasznalo"]=$_POST["nev"];
    // felhasználó értesítése
    echo "Üdv!";
}
// ha az elfelejtő formot beküldték
if(strlen($_POST["felejt"])>0) {
    // cookie törlése: false érték megadásával
    setcookie("felhasznalo",false);
    // cookie törlésének megelőlegezése, lásd fent
    $_COOKIE["felhasznalo"]=NULL;
    // felhasználó értesítése
    echo "Viszlát!";
}
// ha a felhasználót már ismerjük, a cookie elérhető
// ha nem elérhető a cookie, a névbekérő formot
// jelenítjük meg, ha elérhető az elfelejtő formot
// mutatjuk
if(strlen($_COOKIE["felhasznalo"])==0): // nincs cookie
?>
<!-- egyszerű névbekérő form -->
<FORM method="POST">
    Szia! Add meg neved: <INPUT type="text" name="nev">
    <INPUT type="submit" value="emlékezz rám!">
</FORM>
<?
else: // egyébként már belépett
    // felhasználó üdvözlése
    echo "Szia ".$_COOKIE["felhasznalo"]."!";
?>
<!-- egyszerű elfelejtő form -->
<FORM method="POST">
    <INPUT type="submit" name="felejt" value="felejts el!">
</FORM>
<?
endif;
?>
```

A futtatás eredménye:

Szia! Add meg neved:

alap1/000 egyszerű hello world.php

```
<!-- a hello world karakterkonstans  
kiirása standard outputra -->  
<?="hello world!"?>
```

A futtatás eredménye:

hello world!

alap1/009 PHP és HTML vegyesen.php

```
<!--  
A HTML és PHP vegyesen is  
használhatók, a HTML alapértelmezésben  
a standard outputra kerül.  
-->  
<B>HTML</B>  
<? echo " és PHP " ?>  
<I>vegyesen</I> is  
<?= " használható" ?>
```

A futtatás eredménye:

HTML

és PHP vegyesen is
használható

alap1/010 PHP és HTML valtakozása - komplex.php

```
<!--  
A PHP kód vezérlési szerkezete  
meghatározza a kódon kívüli  
HTML megjelenítését is...  
-->  
A válasz:  
<?php  
if ($kifejezes) {  
?>  
    igaz.  
<?php  
} else {  
?>  
    hamis.  
<?php  
}  
?>
```

A futtatás eredménye:

A válasz:

hamis.

alap1/020 kommentezés.php

```
<?
# megjegyzés
echo "ezt kiírja"; // ezt nem
/*
    ezt sem
*/
?>
```

A futtatás eredménye:

ezt kiírja

alap1/030 boolean típus.php

```
<?
// minden kettő érték
// kisbetű-nagybetű érzéketlen
echo TRUE; // igaz érték
echo False; // hamis érték
?>
```

A futtatás eredménye:

1

alap1/040 integer típus.php

```
<?
echo 0xdead; // hexadecimális
echo "<BR>";
echo 1000; // decimális
echo "<BR>";
echo 01000; // oktális
echo "<BR>";
// a túl nagy egészek
// lebegőpontossá alakulnak
echo 12345678901234567890;
?>
```

A futtatás eredménye:

57005
1000
512
1.23456789012E+19

alap1/050 float típus.php

```
<?
echo 12.34; // egyszerű decimális alak
echo "<BR>";
echo 1.34E10; // exponenciális alak
echo "<BR>";
// vigyázzunk az ábrázolási pontatlansággal!
echo 0.1+0.7 == 0.8 ? "igaz" : "hamis";
?>
```

A futtatás eredménye:

12.34
13400000000
hamis

alap1/060 string tipus.php

```
<?
// idézőjekek között megadott
// karakterosorzatokban a változók
// értelmezésre kerülnek
$a = "negyvenkettő";
echo "az a erteke: $a<BR>";
// aposztrófoknál nem értelmez
// az interpreter
echo 'az a erteke: $a<BR>';
// többsoros stringek megadásának
// egy lehetősége a heredoc formátum
$s = <<<VALAMI
ez több soros heredoc
formatumu megadas
VALAMI;
// stringek karaktereire hivatkozás
echo $s[10].' és '.$s[11];
?>
```

A futtatás eredménye:

az a erteke: negyvenkettő
az a erteke: \$a
r és o

alap1/070 array tipus.php

```
<PRE>
<?
// üres tömb
$a = array();
print_r($a);
// asszociatív tömb megadása
$b = array("k" => "ert");
print_r($b);
// hivatkozás tömbelemre
$c = $b["k"];
echo $c."<BR>";
// asszociatív tömb bővítésre
$b["k2"] = "ert2";
print_r($b);
?>
```

A futtatás eredménye:

Array
(
)
Array
(
 [k] => ert
)

```
ert
Array
(
    [k] => ert
    [k2] => ert2
)
```

alap1/080 valtozok.php

```
<PRE>
<?
// változó megadása
$a = "az a értéke";
echo $a."<BR>";
// változó változók
$nev = "valtozo";
$valtozo = "ertek";
echo $$nev."<BR>";
// kétértelműségek feloldása
echo "{$valtozo}es<BR>";
?>
```

A futtatás eredménye:

az a értéke
ertek
ertekes

alap1/090 valtozok lathatosaga.php

```
<?
$kulso = "kulso"; // globális változó
$kivul = "kivul"; // másik globális változó
function f() {
    global $kivul; // globális változó importálása
    $belso = "belso"; // lokális változó
    // az importált globális változó elérhető:
    echo "a kivul erteke: $kivul<BR>";
    // a globális változó közvetlenül nem elérhető:
    echo "a kulso erteke: $kulso<BR>";
    // a szuperglobális mindenben láthatók:
    echo "a kulso erteke: ${_GLOBAL}[kulso]<BR>";
}
f(); // az f függvény meghívása
// a lokális változó nem elérhető
echo "a belso erteke: $belso<BR>";
?>
```

A futtatás eredménye:

a kivul erteke: kivul
a kulso erteke:
a kulso erteke: kulso
a belso erteke:

alap1/100 statikus valtozok.php

```
<?
function fv() {
    // statikus lokális változó definiálása és kezdőértéke
```

```
static $statikus = 0;
// lokális váltózó definiálása és kezdőértéke
$lokalis = 0;
// változók értékének növelése egyel
$statikus++;
$lokalis++;
echo "statikus: $statikus, lokális: $lokalis <BR>";
}
fv(); // első függvényhívás
fv(); // második függvényhívás
?>
```

A futtatás eredménye:

```
statikus: 1, lokális: 1
statikus: 2, lokális: 1
```

alap1/110 konstansok.php

```
<?
// konstans definiálása
define("konstans","ertek");
echo konstans."<BR>";
// a konstans kisbetű/nagybetű érzékeny
// definílatlan konstans értéke a neve
echo KonsTans."<BR>";
?>
```

A futtatás eredménye:

```
ertek
KonsTans
```

alap1/120 kifejezések.php

```
<?
// függvény
function fv() { return 42; }
echo fv()."<BR>";
// egyoperandusú operátor
$a=1;
echo ++$a;
echo "<BR>";
// kétoperandusú operátor
echo $a<=2;
echo "<BR>";
// háromoperandusú operátor
echo $a>=2 ? "igaz" : "hamis";
?>
```

A futtatás eredménye:

```
42
2
1
igaz
```

alap1/130 operatorok.php

```
<?
// egyoperandusú operátor
```

```
$a=1;  
echo ++$a;  
echo "<BR>";  
// kétoperandusú operátor  
echo $a<=2;  
echo "<BR>";  
// háromoperandusú operátor  
echo $a>=2 ? "igaz" : "hamis";  
echo "<BR>";  
// asszociatív értékeadás  
echo $a=$b=42;  
?>
```

A futtatás eredménye:

```
2  
1  
igaz  
42
```

alap1/140 tomb operatorok.php

```
<PRE>  
<?  
// tömbök összeadása operátorral  
$a=array("kulcs1"=>"érték1", "kulcs2"=>"érték2");  
$b=array("kulcs3"=>"érték3", "kulcs4"=>"érték4");  
print_r($a+$b);  
print_r($b+$a);  
// tömbök bővítése operátorral  
$a=array();  
$a[]="első";  
$a[]="második";  
print_r($a);  
?>
```

A futtatás eredménye:

```
Array  
(  
    [kulcs1] => érték1  
    [kulcs2] => érték2  
    [kulcs3] => érték3  
    [kulcs4] => érték4  
)  
Array  
(  
    [kulcs3] => érték3  
    [kulcs4] => érték4  
    [kulcs1] => érték1  
    [kulcs2] => érték2  
)  
Array  
(  
    [0] => első  
    [1] => második  
)
```

alap1/150 operator precedencia.php

```
<?
```

```
$a = (false and false || true);
echo $a ? "igaz" : "hamis";
echo "<BR>";
$a = (false and false or true);
echo $a ? "igaz" : "hamis";
echo "<BR>";
$a = ((false and false) or true);
echo $a ? "igaz" : "hamis";
echo "<BR>";
$a = (false and false) or true;
echo $a ? "igaz" : "hamis";
?>
```

A futtatás eredménye:

hamis
igaz
igaz
hamis

alap2/010 if.php

```
<?
// komplex if példa
$a=2;
if ($a==1) {
    echo "az a 1";
} elseif ($a==2) {
    echo "az a 2";
} else {
    echo "az a más";
}
// alternatív szintaxis
if ($a==1):
    echo "a egy";
endif;
?>
```

A futtatás eredménye:

az a 2

alap2/015 switch.php

```
<?
// komplex switch példa
switch ($s) {
case "egy":
case "egyed":
    echo "egyedem";
case "begyed":
    echo "begyedem";
    break;
default:
    echo "tengertánc";
}
?>
```

A futtatás eredménye:
tengertánc

alap2/020 ciklusok.php

```
<?
// while
while($a<2) {
    echo $a++." while<BR>";
}
// do-while
do {
    echo $a++." do<BR>";
} while($a<2);
// for
for ($a=0; $a<2; $a++) {
    echo "$a for<BR>";
}
// foreach
foreach (array(4,2) as $a) {
    echo "$a foreach<BR>";
}
// break, continue
for ($a=0; $a<2; $a++) {
    for ($b=0; $b<2; $b++) {
        if ($a==$b) continue;
        echo "$a:$b<BR>";
        break;
    }
}
?>
```

A futtatás eredménye:

```
while
1 while
2 do
0 for
1 for
4 foreach
2 foreach
0:1
1:0
```

alap2/030 függvények.php

```
<?
// egyszerű függvény
function osszead($a, $b=2) {
    return $a + $b;
    echo "sőse írja ki";
}
echo osszead(40)."<BR>";
// feltételes deklaráció
if(1<2) {
    function foo() {
        function bar() {
            echo "hejj<BR>";
        }
    }
}
foo();
bar();
```

```
// változó argumentumszám
function kiir() {
    echo func_num_args()." arg.<BR>";
    foreach(func_get_args() as $arg) {
        echo $arg."<BR>";
    }
}
kiir(4,2,"barack");
// függvényváltozó
function alma() {
    echo "alma<BR>";
}
$korte="alma";
$korte();
?>
```

A futtatás eredménye:

```
42
hejj
3 arg.
4
2
barack
alma
```

alap2/040 egyszerű osztály.php

```
<?
// osztálydeklaráció
class A {
    public $mezo;
    public function fv() { echo $this->mezo; }
}
// példányosítás
$a = new A();
// mezőelérés
$a->mezo = "mező érték";
// metódushívás
$a->fv();
?>
```

A futtatás eredménye:

```
mező érték
```

alap2/050 oroklodes.php

```
<?
// összetály deklarációja
class A {
    public $mezo;
    public function fv() { echo $this->mezo; }
}
// öröklődés
class B extends A {
    public $mezo="alapértelmezett";
}
// metódushívás
$b = new B();
$b->fv();
```

```
?>
```

A futtatás eredménye:
alapértelmezett

alap2/060 kivetelkezeles.php

```
<?
try {
    echo "utasítás1<BR>";
    // kivétel kiképítése, "dobása"
    throw new Exception("hiba");
    echo "utasítás2<BR>";
} catch (Exception $e) {
    // kivételek kaptunk el:
    echo "hiba: $e";
}
?>
```

A futtatás eredménye:
utasítás1
hiba: Object id #1

alap2/070 type hinting.php

```
<?
// ősosztály deklarációja
class A {
    public $mezo;
    public function fv() { echo $this->mezo; }
}
// öröklődés
class B extends A {
    public $mezo="alapértelmezett";
}
// egy másik osztály
class C {
    public function fvc(A $arg) {
        $arg->fv();
    }
}
// metódushívás
$b = new B();
$c = new C();
$c->fvc($b); // ez OK
$c->fvc(array("semmi")); // ez HIBA
?>
```

A futtatás eredménye:
alapértelmezett

alap2/080 referenciaiak.php

```
<?
// referencia változó
$a = "a értéke";
$b = &$a;
echo $b."<BR>";
```

```
// referencia argumentum
function novel(&$a) { $a++; }
$c=1;
novel($c);
echo "$c<BR>";
// referencia visszaadása
$adatok=array("első adat","második adat");
print_r($adatok);
echo "<BR>";
function &keres() {
    return $GLOBALS["adatok"][1];
}
$mod=&keres();
$mod="módosítva";
print_r($adatok);
?>
```

A futtatás eredménye:

```
a értéke
2
Array
(
    [0] => első adat
    [1] => második adat
)
Array
(
    [0] => első adat
    [1] => módosítva
)
```

api1/010 tombkezeles 1 - tombok kepzese.php

```
<PRE>
<?
// tömb létrehozása (opcionális)
$a = array();
// hozzáfűzés tömbhöz
$a[] = "érték1";
$a[] = "érték2";
$a["kulcs"] = "érték3";
$a[10] = "érték10";
print_r($a);
// tömbök összevonása
$a = array("Peti","Jancsi","Sanyi");
$b = array("Kati","Panni","Viki","Niki");
$c = $a+$b; // az egyszerű összeadásnál az azonos indexek közül a legelső marad meg!
print_r($c);
$d = array_merge($a,$b); // újraszámorra az integer indexeket, ha szükséges
print_r($d);
// a tömbök összevonása másképp működik string indexeknél!
$a = array("key1"=>"Peti","key2"=>"Jancsi");
$b = array("key1"=>"Kati","key3"=>"Viki");
$c = array_merge($a,$b); // az azonos string indexek közül a legutolsó marad meg !
print_r($c);
$d = $a+$b; // az azonos string indexek közül a legelső marad meg!
print_r($d);
?>
```

A futtatás eredménye:

```
Array
(
    [0] => érték1
    [1] => érték2
    [kulcs] => érték3
    [10] => érték10
)
Array
(
    [0] => Peti
    [1] => Jancsi
    [2] => Sanyi
    [3] => Niki
)
Array
(
    [0] => Peti
    [1] => Jancsi
    [2] => Sanyi
    [3] => Kati
    [4] => Panni
    [5] => Viki
    [6] => Niki
)
Array
(
    [key1] => Kati
    [key2] => Jancsi
    [key3] => Viki
)
Array
(
    [key1] => Peti
    [key2] => Jancsi
    [key3] => Viki
)
```

api1/020 tombkezeles 2 - tombelemek elérése.php

```
<PRE>
<?
// hivatkozás tömbelemekre
$tomb = array("hapci", "tudor", "vidor", "kuka");
echo "$tomb[0], $tomb[1]<BR>";
// értékösszerendelés (csak 0-tól folyamatosan, numerikusan indexelt tömbökön!)
$tomb = array("hapci", "tudor", "vidor", "kuka");
list($a,, $b) = $tomb;
echo "$a, $b<BR>";
// értékek változókba fejtése asszociatív tömbből
$tomb = array("izom"=>"trikó", "diesel" => "merdzsó");
extract($tomb);
echo "$izom, $diesel<BR>";
// tömbelemek elérése foreach ciklussal
$tomb = array("répa"=>"sárga", "alma" => "piros");
foreach($tomb as $kulcs => $ertek) {
    echo "$kulcs: $ertek<BR>";
}
?>
```

A futtatás eredménye:

hapci, tudor
hapci, vidor
trikó, merdzsó
répa: sárga
alma: piros

api1/030 tombkezeles 3 - muveletek tombokkel.php

```
<PRE>
<?
// tömb újraindexelése
$tomb = array(0 => "nap", 3 => "csillagok", 2 => "hold");
print_r($tomb);
print_r(array_values($tomb));
// résztömb
$tomb = array("nem", "de", "te", "vagy", "meg");
print_r(array_slice($tomb,3));
print_r(array_slice($tomb,1,3,true));
// tömb rendezése
// string értékekre annyira nem megbízható
// láasd: collation
$tomb = array(4,2,6,8,2,5,1);
sort($tomb);
print_r($tomb);
// tömb egyedi rendezése (hasonlóan működik a tömbök szűrése is!)
function hasonlit($a,$b) { // saját rendezőfüggvény
    setlocale(LC_COLLATE, 'hu_HU'); // magyar sorrendezés
    return strcoll($a,$b); // sorrendező összehasonlítás
}
$tomb = array("árpa", "maláta", "málna", "makk");
sort($tomb); // nem helyes sorrend
print_r($tomb);
usort($tomb,'hasonlit'); // helyes sorrend saját rendezőfüggvénnyel
print_r($tomb);
?>
```

A futtatás eredménye:

```
Array
(
    [0] => nap
    [3] => csillagok
    [2] => hold
)
Array
(
    [0] => nap
    [1] => csillagok
    [2] => hold
)
Array
(
    [0] => vagy
    [1] => meg
)
Array
(
    [1] => de
    [2] => te
```

```
[3] => vagy
)
Array
(
    [0] => 1
    [1] => 2
    [2] => 2
    [3] => 4
    [4] => 5
    [5] => 6
    [6] => 8
)
Array
(
    [0] => makk
    [1] => maláta
    [2] => málna
    [3] => árpa
)
Array
(
    [0] => árpa
    [1] => málna
    [2] => makk
    [3] => maláta
)
```

api1/040 datumok.php

```
<PRE>
<?
// a pillanatnyi dátum ISO 8601 szöveges formátumban
echo date("c");
echo "<BR>";
// a pillanatnyi dátum alapértelmezett helyi formátumban
setlocale(LC_TIME,"hu_HU.UTF-8");
echo strftime("%c");
echo "<BR>";
// pillanatnyi unix idő mikroszekundum pontossággal
// a Unix Epoch (1970. január 1. 00:00:00 GMT) óta eltelt másodpercek száma
echo microtime(true);
echo "<BR>";
// UNIX timestamp előállítása
$ts = mktime(0, 0, 0, 5, 12, 2007);
echo "$ts<BR>";
echo date("c",$ts);
echo "<BR>";
// UNIX timestamp feldolgozása változókká
print_r(getdate($ts));
// angol időmeghatározások UNIX időbelyeggé alakítása
$ts = strtotime("now");
echo strftime("%c<BR>",$ts);
$ts = strtotime("now +7 days");
echo strftime("%c<BR>",$ts);
$ts = strtotime("1979-09-10 10:30 UTC");
echo strftime("%c<BR>",$ts);
?>
```

A futtatás eredménye:
2007-05-24T12:56:53+02:00

```
2007. máj. 24., csütörtök, 12.56.53 CEST
1180004213.08
1178920800
2007-05-12T00:00:00+02:00
Array
(
    [seconds] => 0
    [minutes] => 0
    [hours] => 0
    [mday] => 12
    [wday] => 6
    [mon] => 5
    [year] => 2007
    [yday] => 131
    [weekday] => Saturday
    [month] => May
    [0] => 1178920800
)
2007. máj. 24., csütörtök, 12.56.53 CEST
2007. máj. 31., csütörtök, 12.56.53 CEST
1979. sze. 10., hétfő, 11.30.00 CET
```

api1/050 fajl es konyvtarkezeles.php

```
<PRE>
<?
// adott könyvtár tartalmának tömbbe kérése (a . a helyi könyvtár)
print_r(scandir("."));
// a scandir() nem csak helyi könyvtárrakra működik!
print_r(scandir("ftp://anonymous:@ftp.fsn.hu/pub"));
// a parancssorhoz szintaxisossal szűrést biztosít a glob()
// alapértelmezésben az aktuális könyvtárban dolgozik
print_r(glob("*.*"));
// az aktuális script fájlneve a szerveren
echo __FILE__;
echo "<BR>";
// csak a fájlnév
echo basename(__FILE__);
echo "<BR>";
// csak a könyvtárnév
echo dirname(__FILE__);
echo "<BR>";
// fájlinformációk (méret, dátumok, stb...)
print_r(stat(__FILE__));
// fájl tartalmának olvasása tömbbe
print_r(file(__FILE__));
// a file() nem helyi könyvtárakkal is működik
print_r(file("ftp://anonymous:@ftp.fsn.hu/README"));
?>
```

A futtatás eredménye:

```
Array
(
    [0] => .
    [1] => ..
    [2] => 010 tombkezeles 1 - tombok_kepzese.php
    [3] => 020 tombkezeles 2 - tombelemek_eleresi.php
    [4] => 030 tombkezeles 3 - muveletek_tombokkel.php
    [5] => 040 datumok.php
    [6] => 050 fajl_es_konyvtarkezeles.php
```

```
[7] => 060 SMTP es levelkuldes.php
)
Array
(
    [0] => CDROM-Images
    [1] => CPAN
    [2] => FreeBSD
    [3] => NetBSD
    [4] => OpenBSD
    [5] => OpenDOS
    [6] => beos
    [7] => crashrecoverykit
    [8] => cygwin
    [9] => debian-superseded
    [10] => docs
    [11] => dragonflybsd
    [12] => freebsd
    [13] => isc
    [14] => kde
    [15] => linux
    [16] => mozilla
    [17] => netbsd
    [18] => openbsd
    [19] => openoffice.org
    [20] => proftpd
    [21] => sunfreeware
    [22] => temp
)
Array
(
    [0] => 010 tombkezeles 1 - tombok_kepzese.php
    [1] => 020 tombkezeles 2 - tombelemek_eleres.php
    [2] => 030 tombkezeles 3 - muveletek_tombokkel.php
    [3] => 040 datumok.php
    [4] => 050 fajl es konyvtarkezeles.php
    [5] => 060 SMTP es levelkuldes.php
)
/home/mcree/public_html/php/code/api1/050 fajl es konyvtarkezeles.php
050 fajl es konyvtarkezeles.php
/home/mcree/public_html/php/code/api1
Array
(
    [0] => 2305
    [1] => 58722128
    [2] => 33204
    [3] => 1
    [4] => 1000
    [5] => 1000
    [6] => -1
    [7] => 807
    [8] => 1180004213
    [9] => 1179052446
    [10] => 1179052446
    [11] => -1
    [12] => -1
    [dev] => 2305
    [ino] => 58722128
    [mode] => 33204
    [nlink] => 1
    [uid] => 1000
    [gid] => 1000
```

```
[rdev] => -1
[size] => 807
[atime] => 1180004213
[mtime] => 1179052446
[ctime] => 1179052446
[blksize] => -1
[blocks] => -1
)
Array
(
    [0] =>
        [1] =>
            [3] => // adott könyvtár tartalmának tömbbe kérése (a . a helyi könyvtár)
            [4] => print_r(scandir("."));
            [5] =>
                [6] => // a scandir() nem csak helyi könyvtárrakra működik!
                [7] => print_r(scandir("ftp://anonymous:@ftp.fsn.hu/pub"));
                [8] =>
                    [9] => // a parancssorihoz szintaxisossal szűrést biztosít a glob()
                    [10] => // alapértelmezésben az aktuális könyvtárban dolgozik
                    [11] => print_r(glob("*.php"));
                    [12] =>
                        [13] => // az aktuális script fájlneve a szerveren
                        [14] => echo __FILE__;
                        [15] => echo "
";
                    [16] => // csak a fájlnév
                    [17] => echo basename(__FILE__);
                    [18] => echo "
";
                    [19] => // csak a könyvtárnév
                    [20] => echo dirname(__FILE__);
                    [21] => echo "
";
                    [22] => // fájlinformációk (méret, dátumok, stb...)
                    [23] => print_r(stat(__FILE__));
                    [24] =>
```

```
[25] => // fájl tartalmának olvasása tömbbe
[26] => print_r(file(__FILE__));
[27] =>
[28] => // a file() nem helyi könyvtárakkal is működik
[29] => print_r(file("ftp://anonymous:@ftp.fsn.hu/README"));
[30] =>
[31] =>
[32] => ?>
)
Array
(
    [0] => Welcome to ftp.fsn.hu.

    [1] =>

    [2] => This server was created to collect and distribute FREELY available
    [3] => CD-ROM images, mainly free operating system install CDs (for example
    [4] => FreeBSD, various Linux distributions and so on...).
    [5] => The statements above imply that we are NOT HOSTING ILLEGAL DATA
    [6] => (aka. WAREZ). If you find anything that violates law please drop me a
    [7] => letter and i will delete the illegal software.

    [8] =>

    [9] => Thank you.

    [10] => bra@fsn.hu

    [11] =>

    [12] => ----

    [13] => Udvozollek az ftp.fsn.hu-n!

    [14] =>

        [15] => A szerveren leginkább a különfele ingyenes operációs rendszerek
(Linux,
        [16] => *BSD) telepítő CD-öt találhatod meg. Ezek mindegyike szabadon
letölthető,
        [17] => CD-re irható, arrol telepíthető.

    [18] => Kerlek ha talalsz olyan anyagot, aminek a terjesztése jogellenes ird
meg!

    [19] =>
```

```
[20] => Koszonom.  
  
[21] => bra@fsn.hu  
  
)
```

api1/060 SMTP es leveleküldés.php

```
<?  
// levélküldés a PHP segítségével  
$címzett = 'peelda@peelda.hu';  
$stargy = 'a level targya';  
$szöveg = 'hello';  
$fejezet = 'From: php@gdf.hu' . "\r\n";  
mail($címzett, $stargy, $szöveg, $fejezet);  
?>
```

A futtatás eredménye:

api2/010 adatbazisok.php

```
<PRE>  
<?  
// opcionális hibakezelő blokk eleje  
try {  
    // csatlakozás SQLite adatbázishoz  
    // példa: csatlakozás MySQL adatbázishoz:  
    // $db = new PDO('mysql:host=localhost;dbname=adatbazis', $felhasznalo, $jelszo);  
    $db = new PDO('sqlite:adatbazis.sqlite');  
    // adatbázis tábla eldobása  
    // $db->exec("drop table kattintasok");  
    // adatbázis tábla létrehozása  
    // a tábla neve: kattintasok  
    // két mezőt tartalmaz, egy ip nevűt (szöveges) és egy mikor nevűt (időbélyeg)  
    $db->exec("create table kattintasok (ip text, mikor timestamp)");  
    // beszúró utasítás előkészítése, az :ipje és az :idobelyeg paraméterek használatával  
    $sql=$db->prepare("insert into kattintasok (ip,mikor) values (:ipje,:idobelyeg)");  
    // az :ipje paraméter feltöltése a PHP scriptet megtekintő felhasználó IP címével  
    $sql->bindValue(':ipje',$_SERVER['REMOTE_ADDR'],PDO::PARAM_STR);  
    // az :idobelyeg paraméter feltöltése az aktuális időbélyeggel  
    $sql->bindValue(':idobelyeg',time(),PDO::PARAM_INT);  
    // beszúró utasítás lefuttatása  
    $sql->execute();  
    // a kattintások csökkenő sorrendben történő lekérdezése max. 3 elemig  
    foreach($db->query("select * from kattintasok order by mikor desc limit 3") as $row) {  
        print_r($row); // adatbázis sor adatainak kiíratása  
    }  
} catch (Exception $e) { // opcionális hibakezelő blokk vége  
    echo "Hiba: " . $e->getMessage(); // hibaüzenet megjelenítése  
}  
?>
```

A futtatás eredménye:

```
Array
```

```
(  
    [ip] => 195.111.2.12  
    [0] => 195.111.2.12  
    [mikor] => 1179389478  
    [1] => 1179389478  
)  
Array  
(  
    [ip] => 84.3.72.19  
    [0] => 84.3.72.19  
    [mikor] => 1179341116  
    [1] => 1179341116  
)  
Array  
(  
    [ip] => 193.225.1.1  
    [0] => 193.225.1.1  
    [mikor] => 1179222761  
    [1] => 1179222761  
)
```

api2/020 stringek.php

```
<PRE>  
<?  
// string hossza  
echo strlen("bekebelezhetetlen-elmeketymereg");  
echo "<BR>";  
// speciális karakterek HTML formátumúvá konvertálása  
// a $szo es a $kodolt változó értéke között csak a  
// generált oldal forrását megtekintve látunk különbséget,  
// mert a legtöbb böngésző a kódolatlan és egyszeresen  
// kódolt karaktereket ugyanúgy jeleníti meg!  
$szo="A négy az < mint az öt!";  
$kodolt=htmlentities($szo,ENT_COMPAT,"UTF-8");  
$duplan_kodolt=htmlentities($kodolt);  
echo "$szo<BR>$kodolt<BR>$duplan_kodolt<BR>";  
// string tömbre robbantása megadott elválasztóknál  
$tomb = explode(" ", "Már volt vagy ősz szinte");  
print_r($tomb);  
// tömb stringgé olvasztása  
echo implode(", ", $tomb);  
echo "<BR>";  
// nem szóalkotó karakterek eltávolítása a string két végéről  
echo trim("      mikor egy őszinte Ősz inte,      ");  
// sorvége jelek HTML formátumúvá konvertálása  
echo nl2br(" legyek őszinte\nmert Ő szinte őszinte!\n");  
// URL változók tömbbe fejtése  
$tomb=array();  
parse_str("asztal=papir&mellette=pap_ir", &$tomb);  
print_r($tomb);  
// formázott megjelenítés (mint C-ben)  
echo sprintf("%x %4s' és a %s %d?!<BR>", 7500000, "év", "válasz", 42);  
// stringek részeinek lecserélése  
echo str_replace(array("e", "á"), array("eve", "ává"), "hát hebegsz rendesen?<BR>");  
// substringek kiválasztása  
echo substr("egy medve nem lehet", 4, 3);  
?>
```

A futtatás eredménye:

```
31
A négy az < mint az öt!
A négy az < mint az öt!
A n&acute;gy az &lt; mint az &ouml;t!
Array
(
    [0] => Már
    [1] => volt
    [2] => vagy
    [3] => ōsz
    [4] => szinte
)
Már,volt,vagy,ōsz,szinte
mikor egy ōszinte œsz inte, legyek ōszinte

mert Ő szinte ōszinte!

Array
(
    [asztal] => papir
    [mellette] => pap_ir
)
7270e0 ' év' és a válasz 42?!
hávát hevebevegsz revendeveseven?
med
```

api2/030 regularis kifejezések.php

```
<PRE>
<?
// emailcím ellenőrzése
// a minta felépítése:
// ^      sor eleje
// \w+    legalább egy szóalkotó karakter
// @      @
// \w+    legalább egy szóalkotó karakter
// \.      .
// \w+    legalább egy szóalkotó karakter
// $      sor vége
echo preg_match("/^\\w+@\\w+\\.\\w+$/","emailcim@domain.hu") ? "helyes" : "helytelen"
;
echo "<BR>";
// szöveg hevegővé alakítása
// a minta felépítése:
// [aeiouáéíóúöőű] egy magánhangzó a felsoroltak közül
// i - kisbetű/nagybetű érzéketlenség
// u - UTF-8 támogatás bekapcsolása
echo preg_replace("/[aeiouáéíóúöőű]/iu","$0v$0","tudsz így beszélni?<BR>");
// szöveg részmintázatok kifejtése egy weblapról
// a minta felépítése:
// <a          <a
// .*?        minél kevesebb bármilyen karakter
// class=l>   class=l>
// (          részminta eleje
// .*?        minél kevesebb bármilyen karakter
// )          részminta vége
// </a          </a
// i - kisbetű/nagybetű érzéketlenség
preg_match_all("<a.*?class=l>(.*)</a/i",file_get_contents("http://www.google.com/search?q=php"),$m);
```

```
print_r($m[1]); // az első részmintára történt illeszkedések megjelenítése  
?>
```

A futtatás eredménye:

```
helyes  
tuvudsz ívígy beveszévelnivi?  
Array  
(  
    [0] => PHP: Hypertext Preprocessor  
    [1] => PHP: Downloads  
    [2] => PHP - Wikipedia, the free encyclopedia  
    [3] => PHP Tutorial  
    [4] => PHP : Home  
    [5] => Error - Not Found  
    [6] => PHPBuilder.com, the best resource for PHP tutorials, templates ...  
    [7] => Hotscripts.com :: PHP  
    [8] => PHP: Hypertext Preprocessor  
    [9] => The PHP Resource Index  
)
```

pat1/010 - osszetett form - sima html.php

```
<!--  
Az alábbi tiszta HTML kóddal szemléltetni kívánjuk a legfontosabb  
formelemeket és azok használatát. Vegyük észre, hogy ez a fájl  
nem tartalmaz PHP kódokat!  
-->  
<!-- a HTML formok elemeit FORM tagok közé kell zárni -->  
<FORM method="POST">  
    <!-- egysoros szövegdoboz alapértelmezett értékkel -->  
    Név:  
    <INPUT type="text" name="nev" value="Balamka Sámuel"><BR>  
    <!-- az alábbi hárrom választómező közül a középső  
        alapértelmezetten kijelölésre került -->  
    Szereti az almát:  
    <INPUT type="checkbox" name="alma" value="alma"><BR>  
    Szereti az epret:  
    <INPUT type="checkbox" name="eper" value="eper" checked><BR>  
    Szereti az barackot:  
    <INPUT type="checkbox" name="barack" value="barack"><BR>  
    <!-- a "rádiógombok" a választómezőkhöz hasonlóan működnek,  
        egy csoportba tartoznak az azonos name értékkel megadott gombok -->  
    Fiú:  
    <INPUT type="radio" name="nem" value="fiú"><BR>  
    Lány:  
    <INPUT type="radio" name="nem" value="lány" checked><BR>  
    <!-- a legördülő menü megadása bonyolultabb szintaxist követ -->  
    Hol lakik:  
    <SELECT name="lakik">  
        <!-- az alábbi három választási lehetőségből alapértelmezésben  
            a második kerül kiválasztásra -->  
        <OPTION value="haz">ház</OPTION>  
        <OPTION value="lakas" selected>lakás</OPTION>  
        <OPTION value="mas">máshol</OPTION>  
    </SELECT><BR>  
    <!-- a HTML lehetőséget ad többsoros szövegbeviteli mező  
        megadására és az alapértelmezett érték megadására is -->  
    Egyéb megjegyzések:  
    <TEXTAREA cols=20 rows=3 name="egyeb">Ide több soros megjegyzést is írhatsz!</  
    TEXTAREA><BR>
```

```
<!-- a submit, vagy beküldőgomb típusú formelemből  
több is lehet egy formon. Lenyomására a form tartalmát  
a böngésző a szerverre továbbítja POST vagy GET metódussal -->  
<INPUT type="submit" name="kuld" value="küld">  
<INPUT type="submit" name="kuld2" value="küld más képp">  
</FORM><!-- form vége -->
```

A futtatás eredménye:

Név:

Szereti az almát:

Szereti az epret:

Szereti az barackot:

Fiú:

Lány:

Hol lakik:

Egyéb megjegyzések:

pat1/020 - osszetett form feldolgozása.php

```
<!-- a fájlból példát mutatunk a böngésző által beküldött  
form PHP oldali feldolgozására. A formmezők alapértelmezésben
```

a name HTML paraméter által megadott néven kerülnek elküldésre.

```
-->
<FORM method="POST">
    Név:<INPUT type="text" name="nev"><BR>
    <!-- ha a name paraméterben tömb szintaxiszt használunk, a PHP
        összefogja a hasonló nevű mezőket -->
    Szereti az almát:<INPUT type="checkbox" name="szereti[]" value="alma"><BR>
    Szereti az epret:<INPUT type="checkbox" name="szereti[]" value="eper"><BR>
    Szereti az barackot:<INPUT type="checkbox" name="szereti[]" value="barack"><BR>
>
    Fiú:<INPUT type="radio" name="nem" value="fiú"><BR>
    Lány:<INPUT type="radio" name="nem" value="lánya"><BR>
    Hol lakik:<SELECT name="lakik">
        <OPTION value="ház">ház</OPTION>
        <OPTION value="lakás">lakás</OPTION>
        <OPTION value="mas">máshol</OPTION>
    </SELECT><BR>
    Egyéb megjegyzések:<TEXTAREA cols=20 rows=3 name="egyeb"></TEXTAREA><BR>
    <INPUT type="submit" name="kuld" value="küld">
    <INPUT type="submit" name="kuld2" value="küld más képp">
</FORM>
<PRE>
<?
// a form adatait a $_POST szuperglobális tartalmazza
// 
// figyeljük meg az alapértelmezésben bekapcsolt
// állapotban levő magic quotes szolgáltatás
// hatását, a szövegmezőkben adjunk meg \, " és ' karaktereket,
// majd hasonlítsuk össze a bevitt értékeket a $_POST tömb
// tartalmával!
print_r($_POST);
?>
</PRE>
```

A futtatás eredménye:

Név:

Szereti az almát:

Szereti az epret:

Szereti az barackot:

Fiú:

Lány:

Hol lakik:

Egyéb megjegyzések:

```
Array
(
)
```

pat1/030 - egyszerű form ellenorzése.php

```
<!-- HTML FORM MEGJELENÍTÉSE -->
<FORM method="GET" action=""><!--GET metódus, akció: önmagunknak-->
    <INPUT type="text" name="nev"><!--szövegmező-->
    <INPUT type="submit" name="kuld" value="küldés"><!--gomb-->
</FORM><!--form vége-->

<!-- FORM FELDOLGOZÓ ÉS ELLENŐRZŐ PHP KÓD -->
<?
    // GET metódus esetén a $_GET szuperglobális listát használjuk
    if($_GET["kuld"]) { // ha a gombot megnyomták, a "kuld" kulcsnak érték
        $nev = $_GET["nev"]; // a "nev" kulcs érteke a szövegmező tartalma
        if(strlen($nev)==0) { // formmező ellenőrzése
            echo "nem adott meg nevet!"; // a név hossza 0 karakter volt
        } else {
            echo "a megadott név: $nev"; // a név megadásra került
        };
    } else {
        echo "a form még nem lett elküldve!";
    }
?>
```

A futtatás eredménye:

a form még nem lett elküldve!

pat1/040 - összetett form újramegjelenítése.php

```
<?
/*
Az egyszer már elküldött form paraméterek újramegjelenítésénél
gondot okozhat ha a magic quotes szolgáltatás be van kapcsolva,
illetve ha a html vezérlőkarakterek egy az egyben visszakerülnek
a form forráskódjába, ezért a magic quotes beállítástól függően
definiáljuk az sz() függvényt, mely elvégzi a szükséges
átalakításokat.
*/
if(get_magic_quotes_gpc()) {
```

```
// az sz() függvény egyik lehetséges definíciója
function sz($s) {
    // bekapcsolt magic quotes esetén a stripslashes() eltávolítja
    // a felesleges escape szekvenciákat
    return htmlentities(stripslashes($s));
}
} else {
    // az sz() függvény másik lehetséges definíciója
    function sz($s) {
        // kikapcsolt magic quotes esetén is szükség van a html
        // speciális karakterek (pl. < és >) entitásokra való
        // cseréjére, erre szolgál a már bemutatott htmlentities()
        return htmlentities($s);
    }
};

/*
A c() függvényt azért hoztuk létre, hogy a form HTML forráskódjába
ne kelljen ezt a hosszú részt annyiszor beírnunk, ahány tömbb változóba
tárolt checkbox értékünk van.

A függvény a megadott tömbben keresi a megadott értéket,
a PHP array_search() függvényének felhasználásával;
ha megtalálja, a checked stringgel tér vissza, ha nem találja
akkor üres stringet ad vissza.
*/
function c($tomb,$ertek) {
    if(array_search($ertek,$tomb)!==false) {
        return "checked";
    } else {
        return "";
    }
}
?>

<!--
A már korábban ismertetett összetett formot
kiegészítettük az esetlegesen a $_POST tömbben
kapott korábbi értékek alapértelmezett értékként
való megjelenítésére. Erre a célra a különböző
html form elemek számára különböző vizsgálatokat
és átalakításokat kell végeznünk.

Figyeljük meg a ?: háromoperandusú operátor
felhasználásának szintaktikai egyszerűsítő szerepét,
valamint a PHP < ? = implicit echo tag használatát!
-->
<FORM method="POST">
    Név:<INPUT type="text" name="nev" value="<?=sz($_POST["nev"])?>"><BR>
    Szereti az almát:<INPUT type="checkbox" name="szereti[]" value="alma" <?=c($_P
    OST["szereti"],"alma")?>><BR>
    Szereti az epret:<INPUT type="checkbox" name="szereti[]" value="eper" <?=c($_P
    OST["szereti"],"eper")?>><BR>
    Szereti az barackot:<INPUT type="checkbox" name="szereti[]" value="barack" <?
    c($_POST["szereti"],"barack")?>><BR>
    Fiú:<INPUT type="radio" name="nem" value="fiú" <?=$_POST["nem"]=="fiú"??"che
    cked":?"?>><BR>
    Lány:<INPUT type="radio" name="nem" value="lány" <?=$_POST["nem"]=="lány"??"che
    cked":?"?>><BR>
    Hol lakik:<SELECT name="lakik">
        <OPTION value="haz" <?=$_POST["lakik"]=="haz"??"selected":?"?>>ház</OPTION>
```

```
<OPTION value="lakas" <?=$_POST["lakik"]=="lakas"? "selected": ""?>>lakás</OPTION>
    <OPTION value="mas" <?=$_POST["lakik"]=="mas"? "selected": ""?>>máshol</OPTION>
>
</SELECT><BR>
Egyéb megjegyzések:<TEXTAREA cols=20 rows=3 name="egyeb"><?=sz($_POST["egyeb"])?></TEXTAREA><BR>
<INPUT type="submit" name="kuld" value="küld">
<INPUT type="submit" name="kuld2" value="küld más képp">
</FORM>

<PRE>
<?

// a form adattartalmát a szokásos módon jelenítjük meg
print_r($_POST);

?>
</PRE>
```

A futtatás eredménye:

Név:

Szereti az almát:

Szereti az epret:

Szereti az barackot:

Fiú:

Lány:

Hol lakik:

Egyéb megjegyzések:

```
Array
(
)
```

pat1/050 - többnyelvű formok.php

```
<!--
Többnyelvű formok megjelenítésére több lehetőségünk
is van, a gyakorlatokon a GET paraméterrel meghatározott
nyelvbeállítás megvalósításával foglalkozunk.
```

Gyakori példát láthatunk a GET és POST paraméterátadás kombinációjára is.

```
-->
<?
// az alábbi asszociatív tömbszerkezetet használjuk
// a megfelelő nyelvi kifejezések kiválasztására,
```

```
// így megspóroljuk az if vagy a ?: operátorok
// ismételt használatát.
//
// Természetesen számos más módon is megoldható a feladat,
// például bonyolultabb esetekben ajánlott a gettext
// használata, lásd: a PHP kézikönyv XLIX. fejezetét!
$po=array(
    "nev" => array("hu" => "név", "en" => "name"),
    "nem" => array("hu" => "nem", "en" => "gender"),
    "ffi" => array("hu" => "férfi", "en" => "male"),
    "no" => array("hu" => "nő", "en" => "female"),
    "kulds" => array("hu" => "beküldés", "en" => "send form")
);
// a nyelvválasztó GET paramétert egy rövidebb változóba tesszük
// itt végezhetünk el a nyelvparaméter tartalmi ellenőrzését is
$l=$_GET["lang"];
if(strlen($l)==0) $l="hu"; // alapértelmezett nyelv
?>
<!-- GET paraméterek beállítása közvetlen HTML linkkel -->
<A href="?lang=en">in English</A><BR>
<A href="?lang=hu">Magyar nyelven</A><BR>

<!-- többnyelvű form megjelenítése a $po és $l változók segítségével -->
<FORM method="POST">
    <?=$po["nev"][$l]?>: <INPUT type="text" name="neve"><BR>
    <?=$po["nem"][$l]?>:
        <?=$po["ffi"][$l]?><INPUT type="radio" name="neme" value="f">
        <?=$po["no"][$l]?><INPUT type="radio" name="neme" value="n"><BR>
    <INPUT type="submit" name="kuldes" value="<?=$po["kulds"][$l]?>">
</FORM>

<PRE>
<?
// kapott értékek megjelenítése
print_r($_POST);
?>
</PRE>
```

A futtatás eredménye:

in English
Magyar nyelven

név:
nem: férfi nő

Array
(
)

pat1/060 - fajlfeltoltes.php

```
<!--
Fájlfeltöltésre csak multipart/form-data típusú,
POST metódsú formokat használhatunk!
```

A fájlfeltöltés buktatóiról lásd a PHP kéziköny 38. fejezetét!

```
-->
<FORM enctype="multipart/form-data" method="POST">
  Fájl: <INPUT NAME="file" TYPE="file">
  <INPUT TYPE="submit" VALUE="Fájl feltöltése">
</FORM>
<PRE>
<?

// fájlfeltöltés valódiságának ellenőrzése
echo is_uploaded_file($_FILES['file']['tmp_name'])."<BR>";

// fájlfeltöltési adatok megjelenítése
print_r($_FILES);

// feltöltött fájl mozgatása egy célhelyre
// az el nem mozgatott feltöltött fájlok a PHP script
// futásának befejeztével automatikusan törlődnek!
//move_uploaded_file($_FILES['file']['tmp_name'],$celfajl);

?>
```

A futtatás eredménye:

Fájl:

```
Array
(
)
```

pat2/010 - sessionkezeles.php

```
<?
// munkamenet-kezelés bekapcsolása
// láasd: PHP kézikönyv, CXLII. fejezet!
session_start();
// a $_SESSION szuperglobális értékei mostantól
// megőrződnek futtatások között
$_SESSION["szamlalo"]++;
echo $_SESSION["szamlalo"]."<BR>";
// munkamenet élettartamának lekérdezése (másodperc)
// általában nem szükséges
echo "session élettartam: ".ini_get("session.gc_maxlifetime");
// session megszüntetése, például kilépés után:
//session_destroy();
//$_SESSION=array();
// munkamenet élettartamának beállítása (másodperc)
// általában nem szükséges
ini_set("session.gc_maxlifetime", "18000");
// bongésző cache kikapcsolása
// fontosabb értékek:
// nocache - munkamenetet használó oldalak cachelésének tiltása
// public - munkamenetet használó oldalal cachelésének engedélyezése
// általában nem szükséges
session_cache_limiter("nocache");
// munkamenet-kezelés befejezése (nem kötelező!)
// erőforrásokat spórolhatunk vele, tipikusan frame-es
// oldalaknál praktikus!
```

```
// általában nem szükséges
session_write_close();
?>
```

A futtatás eredménye:

```
1
session élettartam: 1440
```

pat2/020 - bevasarlokosar.php

```
<?
/* form- és sessionkezelés kombinálva */
?>
<!-- egyszerű szövegbeviteli FORM -->
<FORM method="POST">
    Áru:<INPUT type="text" name="aru">
    <INPUT type="submit">
</FORM>
<?
// munkamenet-kezelés megkezdése
session_start();
// ha kaptunk új árut a formról
if(strlen($_POST["aru"])>0) {
    // hozzáfűzzük a session kosár tömbjéhez
    $_SESSION["kosar"] []=$_POST["aru"];
}
// kosár tartalmának megjelenítése
// egyszerű foreach ciklussal
echo "a kosár tartalma:<BR>";
if(is_array($_SESSION["kosar"])) {
    foreach($_SESSION["kosar"] as $cikk) {
        echo $cikk."<BR>";
    }
}
?>
```

A futtatás eredménye:

Áru:

a kosár tartalma:

pat2/030 - cookie kezeles.php

```
<?
// cookie beállítási példa
// csak ha kaptunk nevet a formon keresztül
if(strlen($_POST["nev"])>0) {
    // a setcookie hívást nem előzheti meg kimenet,
    // vagyis pl. echo vagy HTML tartalom!
    setcookie("tesztcookie", $_POST["nev"], time() + 60 * 60 * 24 * 30);
    // visszajelzés a felhasználónak
    echo "új név: ".$_POST["nev"]."<BR>";
}
```

```
?>
<!-- egyszerű névbekérő form -->
<FORM method="POST">
    Mi neved? <INPUT type="text" name="nev">
    <INPUT type=submit>
<PRE>
<?
// A cookie beállításának eredménye csak a következő
// oldaltöltéskor lesz látható!
print_r($_COOKIE);
?>
```

A futtatás eredménye:

Mi neved?

```
Array
(
)
```

pat3/010 HTTP hitelesites.php

```
<?
/* Példa egyszerű HTTP Basic felhasználóazonosításra.
   Lásd: PHP kézikönyv 34. fejezet!
*/
if( !isset($_SERVER['PHP_AUTH_USER']) ) {
    header('WWW-Authenticate: Basic realm="Azonositsd magad!"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Ez jelenik meg, ha a Cancel/Mégsem gombot nyomja a felhasználó';
    exit;
} else {
    echo "Helló {$_SERVER['PHP_AUTH_USER']}!<BR>";
    echo "A megadott jelszavad: {$_SERVER['PHP_AUTH_PW']}";
}
?>
```

A futtatás eredménye:**pat3/020 - cookie alapú hitelesites.php**

```
<?
/*
Cookie alapú hitelesítést szemléltető PHP program.
*/
// felhasználóneveket és jelszavakat tároló tömb
// akár érkezhetne SQL adatbázisból is...
$felhasznalok=array(
    "felhasznalo"=>"php",
    "erno"=>"alma"
);
if($_POST["belep"]) { // ha kaptunk beléptető formot
    if($felhasznalok[$_POST["nev"]]==$_POST["jelszo"] and strlen($_POST["jelszo"])
>0) { // ha a név és jelszó is érvényes
```

```
    setcookie("belepve", $_POST["nev"], time() + 3600); // egy órányi érvényesség
    $_COOKIE["belepve"] = $_POST["nev"]; // cookie megelőlegezése (lásd korábban!)
} else {
    echo "Hibás név vagy jelszó";
}
if($_POST["kilep"]) { // ha kaptunk kiléptető formot
    setcookie("belepve", false); // cookie törlése
    unset($_COOKIE["belepve"]); // cookie törlés megelőgezése (lásd korábban!)
}
if($_COOKIE["belepve"]) { // kaptunk cookie-t
    echo "Belpéve: ".$_COOKIE["belepve"];
// egyszerű kiléptető form
?>
<FORM method="POST">
    <INPUT type="submit" name="kilep" value="kilépés">
</FORM>
<?
} else { // nem kaptunk cookie-t
// egyszerű beléptető form
?>
<FORM method="POST">
    Név:<INPUT type="text" name="nev">
    Jelszó:<INPUT type="password" name="jelszo">
    <INPUT type="submit" name="belep" value="belépés">
</FORM>
<?
}
?>
```

A futtatás eredménye:

Név:

Jelszó:

pat3/030 - template rendszer.php

```
<?
// minimalista template rendszer és MVC példa
//
// a program vezérlése a következő feladatláncon folyik
// felhasználó -> controller -> model -> view -> felhasználó
// a model megjelenítése template-ek használatával
function view($adat, $lista, $üzemmod) {
    // a különböző template-ek származhatnának SQL adatbázisból
    // vagy XSLT transzformációs leírásokból is
    $template["bevezeto_üzemmod"] = "<H1>|CIM|</H1>Bevezető: |BEVEZETO|<P>Írta: |SZERZO|";
    $template["egysoros_üzemmod"] = "|CIM| (írta: |SZERZO|)";
    // template feldolgozása és megjelenítése
    // (adattartalom behelyettesítése)
    echo str_replace(
        array("|CIM|", "|BEVEZETO|", "|SZERZO|"),
        array($adat, $üzemmod, $szemantika),
        $template[$üzemmod]
    );
}
```

```
array($adat["cim"], $adat["bevezeto"], $adat["szerzo"]),
$template[$uzemmod]
);
// a szükséges vezérlőelemek megjelenítése
// (ez is template használatot igényelne!)
// üzemmód váltó form
echo "<FORM method=GET>";
foreach(array_keys($template) as $umod) {
    echo "<INPUT type='submit' name='uzemmod' value='$umod'>";
}
echo "</FORM>";
// adatmező választó form
echo "<FORM method=POST>";
foreach($lista as $elem) {
    echo "<INPUT type='submit' name='azonosito' value='$elem'>";
}
echo "</FORM>";
}
// adattartalom és működési logika megvalósítása
function model($azonosito, $uzemmod) {
    // a $db változó egy SQL adatbázist is jelképezhet
    // feladata az adattartalom tárolása
    $db["első könyv"]=array("cím"=>"Halak Jelleme", "bevezeto"=>"Humoros könyv sokok állattal", "szerzo"=>"Gerald Durrell");
    $db["masodik könyv"]=array("cím"=>"Falak Szelleme", "bevezeto"=>"Kalandos krimi az elején gyilkossággal", "szerzo"=>"Leslie Durrell");
    // a működési logika szerint kiválasztjuk az adatbázis megfelelő sorát
    if(strlen($azonosito)!=0) {
        $adat=$db[$azonosito];
    } else {
        $adat=$db["első könyv"];
    }
    // a rendelkezésre álló adatok azonosítói
    $lista=array_keys($db);
    // a megjelenítési alrendszerét értesítjük a változásokról
    view($adat, $lista, $uzemmod);
}
// bejövő adatok feldolgozása, a model állapotának megváltoztatása
function controller() {
    // itt történhetne a bejövő adatok
    // ártalmatlanítása, előfeldolgozása
    // lásd: htmlentities() és stripslashes()
    model($_POST["azonosito"], $_GET["üzemmód"]);
}
// program belépési pont: bejövő adatok feldolgozása
controller();
?>
```

A futtatás eredménye:

pat4/010 chat.php

```
<?
/* Egyszerű chat program adatbázis felhasználásával */
```

```
// opcionális hibakezelő blokk eleje
try {
    // csatlakozás SQLite adatbázishoz
    $db = new PDO('sqlite:chat.sqlite');
    // adatbázis tábla eldobása
    //$db->exec("drop table chat");
    // adatbázis tábla létrehozása
    $db->exec("create table chat (ip text, szoveg text, mikor timestamp)");
    // ha kaptunk POST adatot
    if(strlen($_POST["szoveg"])>0) {
        // beszúró utasítás előkészítése, az :ipje és az :idobelyeg paraméterek használatával
        $stat=$db->prepare("insert into chat (ip,szoveg,mikor) values (:ipje,:szoveg,:idobelyeg)");
        // az :ipje paraméter feltöltése a PHP scriptet megtekintő felhasználó IP címével
        $stat->bindValue(':ipje',$_SERVER['REMOTE_ADDR'],PDO::PARAM_STR);
        // az :szovege paraméter feltöltése a POST adattal
        $stat->bindValue(':szovege',$_POST["szoveg"],PDO::PARAM_STR);
        // az :idobelyeg paraméter feltöltése az aktuális időbélyeggel
        $stat->bindValue(':idobelyeg',time(),PDO::PARAM_INT);
        // beszúró utasítás lefuttatása
        $stat->execute();
    }
/*
A megjelenítést két részre kellett bontanunk, mert ha azt szeretnénk, hogy hozzászólás beküldése nélkül is folyamatosan frissüljön a chat tartalma, akkor az oldalt újra és újra meg kell jeleníttetni. Ebben az esetben viszont problémát jelentene az, hogy ha a felhasználó éppen gépelne a hozzászólást bekérő formba akkor az oldal kifrissülne alóla.
*/
    if($_GET["mod"]!="chat") { // ha nem kapunk GET paramétert, a főoldalt kell megjeleníteni
?>
<!-- HTML oldal beágyazása, GET paraméter megadásával saját magunkra hivatkozunk -->
<IFRAME src="?mod=chat" width="100%" height="500px"></IFRAME>
<!-- egyszerű hozzászólás-bekérő form -->
<FORM method="POST">
<INPUT type="text" name="szoveg">
<INPUT type="submit" value="hozzászól">
</FORM>
<?
    } else { // az IFRAME tartalmát kell megjeleníteni
        // az IFRAME-en belüli oldal tartalmának másodpercenkénti frissítésére
        // a browserrel (a felhasználó megszakíthatja, de ha nem nyúl hozzá, megfelelően fog működni)
        echo "<meta http-equiv='refresh' content='1'>";
        // a hozzászólások növekvő sorrendben történő lekérdezése max. 20 elemig
        foreach($db->query("select * from chat order by mikor asc limit 20 ") as $row) {
            // adatbázis sor adatainak megjelenítése
            echo strftime("%c",$row["mikor"])." (".$row["ip"].") : ".$row["szoveg"]."<BR>";
        };
    }
} catch (Exception $e) { // opcionális hibakezelő blokk vége
    echo "Hiba: " . $e->getMessage(); // hibaüzenet megjelenítése
}
?>
```

A futtatás eredménye:

pat4/020 dinamikus kepek.php

```
<?
/*
Beágyazott dinamikus kép megjelenítő PHP program.
A chat alkalmazáshoz hasonlóan itt is egy programmal
állítunk elő egyszerre két fajta tartalmat egy GET
paramétertől függően.
*/
// ha kaptunk GET színparamétereket, a képet kell megjelenítenünk
if(strlen($_GET["piros"])!=0) {
    // lásd: PHP kézikönyv LXII. fejezet!

    // kép erőforrás létrehozása
    $im = @imagecreatetruecolor(200, 100)
        or die("Nem tudok képet létrehozni!");
    // a szöveg színének allokálása
    $text_color = imagecolorallocate($im, 233, 14, 91);
    // szöveg megjelenítése a képen (vigyázat, nem UTF-8!)
    imagestring($im, 1, 5, 5, "Udv a kepbol!", $text_color);
    // az ellipszis színének allokálása
    $col_ellipse = imagecolorallocate($im, $_GET["piros"], $_GET["zold"], $_GET["kek"]);
}
// ellipiszis megjelenítése a képen
imagefilledellipse($im, 200, 150, 300, 200, $col_ellipse);
// PNG típusú tartalom jelzése a böngésző számára
header ("Content-type: image/png");
// PNG tartalmának küldése a böngésző felé
imagepng($im);
// kép adatok megsemmisítése
imagedestroy($im);
} else { // nem kaptunk GET paramétert, a HTML oldalt kell megjelenítenünk
    // GET paraméterekkel megadott kép hivatkozás küldése a böngészőnek
    echo "<IMG src='?piros=".(int)$_POST["red"]."&zold=".(int)$_POST["green"]."&kek=".(int)$_POST["blue"]."'>";
?
<!-- Egyszerű színparaméter-bekérő form -->
<FORM method="POST">
Piros: <INPUT type="text" name="red">
Zöld: <INPUT type="text" name="green">
Kék: <INPUT type="text" name="blue">
<INPUT type="submit">
</FORM>
<?
}
?>
```

A futtatás eredménye:



Piros:

Zöld:

Kék:

Melléklet: a hivatalos PHP kézikönyv

Mint ahogy említettük, a gyakorlati anyag szerves részét képzi a hivatalos PHP kézikönyv, ezért ennek első fejezeteit módosítás nélkül mellékeltük. A teljes kézikönyv több, mint 4000 oldal lenne, ezért ennek tanulmányozását inkább on-line formában ajánljuk. A kézikönyv elérhető a tantárgy segédletei közt, vagy a php.net weblapon keresztül is.

PHP kézikönyv

Mehdi Achour

Friedhelm Betz

Antony Dovgal

Nuno Lopes

Philip Olson

Georg Richter

Damien Seguy

Jakub Vrana

[És még sokan mások](#)

Fordítók:

Bagi Levente László

Csontos András

Heilig Szabolcs

Hojtsy Gábor

Kontra Gergely

Papp Győző

Tóth Attila

Varanka Zoltán

Másképpen segített:

Jouni Ahto

2006-10-30

Copyright © 1997-2006 the PHP Documentation Group

Copyright © 2000-2006 A PHP dokumentáció magyar fordítói

Copyright

Copyright © 1997 - 2006 by the PHP Documentation Group. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later. A copy of the [Open Publication License](#) is distributed with this manual, the latest version is presently available at <http://www.opencontent.org/openpub/>.

Distribution of substantively modified versions of this document is prohibited without the explicit permission of the copyright holder.

Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.

In case you are interested in redistribution or republishing of this document in whole or in part, either modified or unmodified, and you have questions, please contact the copyright holders at doc-license@lists.php.net. Note that this address is mapped to a publicly archived mailing list.

The [46. fejezet](#) section of the documentation is based on an initial contribution by Zend Technologies.

A magyar copyright megegyezik az angol feltételekkel. Az [Open Publication License](#) egy másolata ezen dokumentációhoz csatolva is olvasható. A PHP kézikönyv magyar fordításának oldala a

http://wfsz.njszt.hu/projektek_phpdoc.php címen található. A fordítók nevei szintén a fedlapon olvashatóak.

Tartalom

Előszó

[Szerzők és más segítők](#)

I. Első lépések

1. [Bevezetés a PHP-be](#)
2. [A simple tutorial](#)

II. Telepítés és beállítás

3. [Általános telepítési szempontok](#)
4. [Telepítés Unix rendszerre](#)
5. [Telepítés Mac OS X rendszerre](#)
6. [Telepítés Windows rendszerekre](#)
7. [PECL kiterjesztések telepítése](#)
8. [Problémák?](#)
9. [Futásidőjű beállítások](#)

III. A nyelv alapjai

10. [Alapvető szintaxis](#)
11. [Típusok](#)
12. [Változók](#)
13. [Állandók](#)
14. [Kifejezések](#)
15. [Operátorok](#)
16. [Vezérlési szerkezetek](#)
17. [Függvények](#)
18. [Osztályok, objektumok \(PHP 4-ben\)](#)
19. [Osztályok és objektumok \(PHP 5\)](#)
20. [Kivételek \(Exceptions\)](#)
21. [Referenciák](#)

IV. Biztonság

22. [Bevezetés](#)
23. [Általános szempontok](#)
24. [CGI futtatható állományként telepített PHP](#)
25. [Apache modulként telepített PHP](#)
26. [Fájlrendszer biztonság](#)
27. [Adatbázis biztonság](#)
28. [Hibakezelés](#)
29. [Globálisan is elérhető változók \(Register Globals\) használata](#)
30. [Felhasználótól érkező adatok](#)
31. [Magic Quotes](#)
32. [A PHP elrejtése](#)
33. [Fontos aktuálisnak maradni](#)

V. Szolgáltatások

34. [HTTP hitelesítés PHP-vel](#)
35. [Sütik \(cookie-k\)](#)
36. [Sessions](#)
37. [Dealing with XForms](#)
38. [Fájlfeltöltés kezelése](#)
39. [Távoli állományok kezelése](#)
40. [Kapcsolatkezelés](#)

41. [Állandó adatbázis kapcsolatok](#)
42. [Safe Mode](#)
43. [Parancssori programozás a PHP-ben](#)

Előszó

A PHP, bővebben "PHP: Hypertext Preprocessor" egy széles körben használt, nyílt forráskódú, általános célú programozási nyelv, különösen jó web-fejlesztés támogatással, és HTML-be ágyazási képességekkel. A szintaksza a C, Java és Perl nyelvekre épül, könnyen megtanulható. A nyelv fő célja lehetőséget teremteni dinamikusan generált weboldalak gyors készítésére, de a PHP ennél sokkal többre is képes.

Ez a kézikönyv legfőképpen [függvény referenciaként](#) használható, azonban a nyelvi elemek ismertetése is megtalálható benne, a főbb [PHP szolgáltatások](#) leírása, és más [kapcsolódó információk](#) mellett.

Ez a kézikönyv számos nyelven és formátumban letölthető a <http://www.php.net/download-docs.php> címen. Ha érdekel, hogyan készül ez a kézikönyv, olvasd el az '[About the manual](#)' című függeléket. Ha a [PHP történetére](#) vagy kíváncsi, olvasd el az ide vonatkozó függeléket.

A magyar kézikönyvet téritésmentesen készítjük, szabadidőnkben. A le nem fordított fejezetek mindenkorban a legfrissebb angol szöveggel jelennek meg, ezért vegyes nyelvű oldalakkal igen ritkán fogsz találkozni. Időnként az azonban előfordulhat, hogy bizonyos már lefordított fejezetek nem teljesen aktuálisak. Emiatt ha minden aktuális és pontos információra van szükséged, és értesz angolul, érdemes ezt a kézikönyvet az angol változattal párhuzamosan használni. Az esetleges kellemetlenségekért elnázést kérünk, bár felelősséget természetesen nem tudunk vállalni.

A PHP kézikönyv magyar fordítása a [Neumann János Számítógép-tudományi Társaság Webalkalmazások Fejlesztése Szakosztályának](#) egyik projektje. A fordítás oldala a http://wfsz.njszt.hu/projektek_phpdoc.php címen található. Ha szeretnél bekapcsolódni a fordításba, látogasd meg ezt a címet.

Szerzők és más segítők

A utóbbi idők leginkább aktív szerzőit soroljuk fel a kézikönyv fedőlapján, de fontos megjegyeznünk, hogy sokkal több hozzájáruló segítő dolgozik jelenleg is a PHP dokumentációján, és még többen segítettek a múltban. Szintén sok névtelen személy van, akik beküldötték megjegyzésekkel, kommentárjaikkal egészítették ki a kézikönyvet, hiszen ezeket időről időre beépítjük az oldalakra. minden alább látható lista ábécésorrendben sorolja fel a segítőket.

Szerzők és szerkesztők

A következő hozzájáruló segítők a kézikönyv írásával, kibővítésével jelentős mértékben hatottak a jelenlegi tartalomra: Jouni Ahto, Alexander Aulbach, Daniel Beckham, Stig Bakken, Jesus M. Castagnetto, Ron Chmara, Sean Coates, John Coggeshall, Simone Cortesi, Markus Fischer, Wez Furlong, Sara Golemon, Rui Hirokawa, Brad House, Moriyoshi Koizumi, Rasmus Lerdorf, Andrew Lindeman, Stanislav Malyshev, Rafael Martinez, Yasuo Ohgaki, Derick Rethans, Sander Roobol, Egon Schmid, Thomas Schoefbeck, Sascha Schumann, Dan Scott, Lars Torben Wilson, Jim Winstead, Jeroen van Wolffelaar és Andrei Zmievski.

A következő hozzájáruló segítők jelentős munkát fektettek a kézikönyv szerkesztésébe: Stig

Bakken, Gabor Hojtsy, Hartmut Holzgraefe és Egon Schmid.

Felhasználói megjegyzések kezelői

A jelenleg legaktívabb karbantartók: Mehdi Achour, Friedhelm Betz, Vincent Gevers, Aidan Lister, Nuno Lopes és Tom Sommer.

A következő személyeket szintén köszönet illeti jelentős részvételükért: Daniel Beckham, Victor Boivie, Jesus M. Castagnetto, Nicolas Chaillan, Ron Chmara, James Cox, Sara Golemon, Zak Greant, Szabolcs Heilig, Oliver Hinckel, Hartmut Holzgraefe, Rasmus Lerdorf, Andrew Lindeman, Maxim Maletsky, James Moore, Sebastian Picklum, Derick Rethans, Sander Roobol, Damien Seguy, Jason Sheets, Jani Taskinen, Yasuo Ohgaki, Philip Olson, Lars Torben Wilson, Jim Winstead, Jared Wyles és Jeroen van Wolfelaar.

I. Első lépések

Tartalom

1. [Bevezetés a PHP-be](#)
 2. [A simple tutorial](#)
-

1. Fejezet. Bevezetés a PHP-be

Mi az a PHP?

A PHP (rekurzív rövidítéssel "PHP: Hypertext Preprocessor") széles körben használt általános célú szkriptnyelv, amely kifejezetten alkalmas - akár HTML-be ágyazott - webalkalmazások fejlesztésére.

Egyszerű meghatározás, de mit is jelent ez valójában? Egy példán bemutatva:

Példa 1-1. Egy bevezető példa

```
<html>
  <head>
    <title>Példa</title>
  </head>
  <body>

    <?php
      echo "Helló, Én egy PHP szkript vagyok!";
    ?>

  </body>
</html>
```

Vedd észre, hogy ez mennyire különbözik más nyelveken (például Perl vagy C) írt hagyományos szkripttől. Sok parancsból álló programok helyett csak egy HTML fájlba kell egy kevés programkódot beépíteni, hogy a megfelelő HTML kimenetet produkálja. A PHP kódblokkokat speciális kezdő és befejező jelölések közé kell elhelyezni, és ezek biztosítják, hogy a feldolgozás során a váltogathasd a "PHP módot".

Az különbözteti meg a PHP-t a kliens oldali nyelvectől - pl. JavaScript -, hogy a kód a kiszolgálón fut. Az első példában látható oldal eredményét böngészőben megnézve, nem lehet megállapítani, hogy milyen kód állíthatta azt elő. Ezen felül a webszervert be lehet állítani úgy, hogy a PHP feldolgozzon minden HTML fájlt PHP blokkokat keresve, ezek után már tényleg nincs rá mód, hogy

kitalálják, mit is rejt egy-egy programod.

A legjobb dolog a PHP használatában, hogy különösen egyszerű egy kezdő számára, de számos, fejlett szolgáltatást nyújt professzionális programozó számára is. Ne ijeszen el a PHP hosszú szolgáltatás listája. Gyors léptekkel lehet haladni a PHP használatában, és pár órán belül már egyszerű szkriptek írására is képes lehetsz.

Habár a PHP fejlesztésében a szerver-oldali programozás kapja a legnagyobb hangsúlyt, annál sokkal többet tud. Olvasd tovább ezt a fejezetet a [Mit tud a PHP?](#) című résszel folytatva, vagy ugorj el a [bevezető oktatóanyagunkra](#), ha csupán úgy általában vagy kíváncsi a web programozására.

Mit tud a PHP?

Bármit. A PHP főleg szerver-oldali szkriptek írására készült, azaz bármire képes, amit más CGI programok el tudnak végezni, ilyen funkciók az ürlap adatok feldolgozása, dinamikus tartalom generálása, vagy sütik küldése és fogadása. De a PHP ennél sokkal többet tud.

Három fő területen használnak PHP programokat.

- Szerver oldali programozás. Ez a hagyományos, és fő használati formája a PHP-nek. Három komponens szükséges ahhoz, hogy ezt a formát használhasd. Az első a PHP értelmező (CGI vagy szerver modul formájában), egy webszerver és egy webböngésző. Egy webszerverrel mindenkorban rendelkezni kell, megfelelően telepített és beállított PHP-vel. A PHP program kimenetét a webböngészővel lehet megtekinteni, a szerveren keresztül elérve a szkriptet. Mindezek képesek elutni a te otthoni gépeden is, ha csupán csak ismerkedni kívánsz a nyelvvel. Lásd a [telepítési utasításokat](#)c. részt további információkért!
- Parancssori programozás. PHP programok szerver és böngésző nélkül is futtathatóak. Ha ilyen környezetben szeretnéd használni a PHP-t, csak a PHP értelmezőre van szükséged. Ebben a formában gyakran valamilyen ütemező program segítségével (cron *nix és Linux alatt, Task Scheduler Windows alatt) futtatott programokat írnak, vagy egyszerű szövegfeldolgozó szkripteket készítenek. Lásd a [Parancssori használat](#) című függeléket további információért!
- Ablakozós alkalmazások írása. A PHP valószínűleg nem a legjobb nyelv grafikus felületű asztali alkalmazások írásához, de ha nagyon jól ismered a PHP-t, és szeretnél néhány fejlett PHP szolgáltatást használni a kliens-oldali programaidban, a PHP-GTK-t is használhatod ilyen programok írásához. Ezt használva lehetőséged van operációs rendszerfüggetlen programok írására is. A PHP-GTK a PHP egy kiterjesztése, nem érhető el a hivatalos PHP csomagban. Ha további információkra van szükséged látogasd meg a [PHP-GTK webhelyet](#)!

A PHP használható a legfontosabb operációs rendszereken, beleértve a Linuxot, sok Unix változatot (beleértve a HP-UX, Solaris és OpenBSD rendszereket), a Microsoft Windows-t, a Mac OS X rendszert, a RISC OS-t, és másokat. A PHP a legtöbb webszervert is támogatja, beleértve az Apache, Microsoft Internet Information Server, Personal Web Server, Netscape és iPlanet szervereket, az O'Reilly Website Pro, Caudium, Xitami, OmniHTTPD, és más szervereket. A legtöbb szerverhez a PHP modul szintű támogatást nyújt, de más a CGI szabványt támogató szerverekkel is együtt tud működni CGI feldolgozóként.

Összességében a PHP használatakor szabadon választhatasz operációs rendszert és webszervert. Ráadásul a függvény-alapú és objektum orientált programozás, vagy ezek keveréke közötti választás is rajtad áll. Bár nem minden szokásos OOP szolgáltatás került megvalósításra a PHP 4-es változatában, sok eljáráskönyvtár és nagyobb alkalmazás is az OOP-t használja, például a PEAR könyvtár. A PHP 5-ös változata helyetteszi a PHP 4 OOP terén mutatott gyengeségeit, teljes objektum modell áll rendelkezésünkre.

A PHP képességei nem csak HTML kimenet előállítására korlátozódnak. Képeket, PDF állományokat vagy akár Flash mozikat (libswf vagy Ming kiterjesztéssel) is létrehozhatsz futásidőben. Természetesen egyszerűen generálhatsz bármilyen szöveges kimenetet, mint az XHTML vagy bármilyen más XML. A PHP elő tudja állítani ezeket az állományokat, és el tudja menteni a szerven a közvetlen kiküldésük helyett, valamelyen szerver-oldali gyorsítótárat valósítva meg ezzel.

Az egyik legjobb és legfontosabb tulajdonsága a nyelvnek az adatbázisok széles körű támogatása. Adatbázisokat kezelő weblap készítése PHP segítségével hihetetlenül egyszerű. A következő adatbázisok támogatja jelenleg:

Adabas D	InterBase	PostgreSQL
dBase	FrontBase	SQLite
Empress	mSQL	Solid
FilePro (csak olvasásra)	Direct MS-SQL	Sybase
Hyperwave	MySQL	Velocis
IBM DB2	ODBC	Unix dbm
Informix	Oracle (OCI7 és OCI8)	
Ingres	Ovrimos	

A PHP rendelkezik egy DBX adatbázis absztrakciós kiterjesztéssel is, amellyel egyötöntően és áttetsző módon lehet kezelní bármilyen adatbázist, amit ez a kiterjesztés támogat. Ezen kívül a PHP támogatja az ODBC-t, ezért bármilyen más, ezt a szabványt támogató adatbázishoz is lehet kapcsolódni.

A PHP támogatja a kommunikációt más szolgáltatásokkal is különböző protokollok segítségével, úgy mint LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (Windows rendszeren), és számos más. Sőt, nyithatsz hálózati foglalatokat is (socket) és kommunikálhatsz más protokollokkal is. A PHP támogatja a WDDX komplex adatcseréket, ami bármely más web programozási nyelvvel való kommunikációt megkönnyítheti. A PHP szintén rendelkezik a Java objektumok létrehozásának, és átlátszóan PHP objektummokként való kezelésének képességével. A CORBA kiterjesztés távoli objektumok elérésére használható.

A PHP rendkívül jó szövegfeldolgozó képességekkel rendelkezik, a POSIX és Perl reguláris kifejezésektől az XML állományok kezeléséig. Az XML dokumentumok feldolgozásához és eléréséhez PHP4-ben a SAX és DOM szabványok is használhatók, Az XSLT kiterjesztés XML dokumentumok általakítására használható. A PHP 5 az XML kapcsolatos feladatokat egységesen, a libxml2 függvénykönyvtárra támaszkodva látja el. Ezen biztos alapokon a PHP 5 bevezeti a SimpleXML és az XMLReader támogatást is.

Ha elektornikus üzleti környezetben használod a PHP-t, hasznosnak fogod találni a Cybercash, CyberMUT, VerySign Payflow Pro and MCVE függvényeket az internetes fizetést megvalósító programokban.

Végül, de nem utolsósorban a PHP számos más érdekes kiterjesztéssel szolgálhat, mint például az mnoGoSearch kereső függvények, az IRC átvájtó függvények, tömörítő eszközök (gzip, bz2), naptár átalakítás, fordítás...

Ahogy látható, ez az oldal nem elegendő a PHP minden szolgáltatásának és előnyének felsorolásához. Lásd a [PHP telepítése](#) és a [függvény referencia](#) részeket további információkért!

2. Fejezet. A simple tutorial

Here we would like to show the very basics of PHP in a short, simple tutorial. This text only deals

with dynamic webpage creation with PHP, though PHP is not only capable of creating webpages. See the section titled [What can PHP do](#) for more information.

PHP-enabled web pages are treated just like regular HTML pages and you can create and edit them the same way you normally create regular HTML pages.

What do I need?

In this tutorial we assume that your server has activated support for PHP and that all files ending in .php are handled by PHP. On most servers, this is the default extension for PHP files, but ask your server administrator to be sure. If your server supports PHP, then you do not need to do anything. Just create your .php files, put them in your web directory and the server will automatically parse them for you. There is no need to compile anything nor do you need to install any extra tools. Think of these PHP-enabled files as simple HTML files with a whole new family of magical tags that let you do all sorts of things. Most web hosts offer PHP support, but if your host does not, consider reading the [PHP Links](#) section for resources on finding PHP enabled web hosts.

Let us say you want to save precious bandwidth and develop locally. In this case, you will want to install a web server, such as [Apache](#), and of course [PHP](#). You will most likely want to install a database as well, such as [MySQL](#).

You can either install these individually or choose a simpler way. Our manual has [installation instructions for PHP](#) (assuming you already have some webserver set up). In case you have problems with installing PHP yourself, we would suggest you ask your questions on our [installation mailing list](#). If you choose to go on the simpler route, then [locate a pre-configured package](#) for your operating system, which automatically installs all of these with just a few mouse clicks. It is easy to setup a web server with PHP support on any operating system, including MacOSX, Linux and Windows. On Linux, you may find [rpmfind](#) and [PBone](#) helpful for locating RPMs. You may also want to visit [apt-get](#) to find packages for Debian.

Your first PHP-enabled page

Create a file named `hello.php` and put it in your web server's root directory (*DOCUMENT_ROOT*) with the following content:

Példa 2-1. Our first PHP script: `hello.php`

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
  <body>
    <?php echo '<p>Hello World</p>'; ?>
  </body>
</html>
```

Use your browser to access the file with your web server's URL, ending with the "/hello.php" file reference. When developing locally this URL will be something like `http://localhost/hello.php` or `http://127.0.0.1/hello.php` but this depends on the web server's configuration. If everything is configured correctly, this file will be parsed by PHP and the following output will be sent to your browser:

```
<html>
  <head>
    <title>PHP Test</title>
  </head>
```

```
<body>
<p>Hello World</p>
</body>
</html>
```

This program is extremely simple and you really did not need to use PHP to create a page like this. All it does is display: *Hello World* using the PHP [echo\(\)](#) statement. Note that the file *does not need to be executable* or special in any way. The server finds out that this file needs to be interpreted by PHP because you used the ".php" extension, which the server is configured to pass on to PHP. Think of this as a normal HTML file which happens to have a set of special tags available to you that do a lot of interesting things.

If you tried this example and it did not output anything, it prompted for download, or you see the whole file as text, chances are that the server you are on does not have PHP enabled, or is not configured properly. Ask your administrator to enable it for you using the [Installation](#) chapter of the manual. If you are developing locally, also read the installation chapter to make sure everything is configured properly. Make sure that you access the file via http with the server providing you the output. If you just call up the file from your file system, then it will not be parsed by PHP. If the problems persist anyway, do not hesitate to use one of the many [PHP support](#) options.

The point of the example is to show the special PHP tag format. In this example we used `<?php` to indicate the start of a PHP tag. Then we put the PHP statement and left PHP mode by adding the closing tag, `?>`. You may jump in and out of PHP mode in an HTML file like this anywhere you want. For more details, read the manual section on the [basic PHP syntax](#).

A Note on Line Feeds: Line feeds have little meaning in HTML, however it is still a good idea to make your HTML look nice and clean by putting line feeds in. A linefeed that follows immediately after a closing `?>` will be removed by PHP. This can be extremely useful when you are putting in many blocks of PHP or include files containing PHP that aren't supposed to output anything. At the same time it can be a bit confusing. You can put a space after the closing `?>` to force a space and a line feed to be output, or you can put an explicit line feed in the last echo/print from within your PHP block.

A Note on Text Editors: There are many text editors and Integrated Development Environments (IDEs) that you can use to create, edit and manage PHP files. A partial list of these tools is maintained at [PHP Editors List](#). If you wish to recommend an editor, please visit the above page and ask the page maintainer to add the editor to the list. Having an editor with syntax highlighting can be helpful.

A Note on Word Processors: Word processors such as StarOffice Writer, Microsoft Word and Abiword are not optimal for editing PHP files. If you wish to use one for this test script, you must ensure that you save the file as *plain text* or PHP will not be able to read and execute the script.

A Note on Windows Notepad: If you are writing your PHP scripts using Windows Notepad, you will need to ensure that your files are saved with the .php extension. (Notepad adds a .txt extension to files automatically unless you take one of the following steps to prevent it.) When you save the file and are prompted to provide a name for the file, place the filename in quotes (i.e. "hello.php"). Alternatively, you can click on the 'Text Documents' drop-down menu in the 'Save' dialog box and change the setting to "All Files". You can then enter your filename without quotes.

Now that you have successfully created a working PHP script, it is time to create the most famous

PHP script! Make a call to the [phpinfo\(\)](#) function and you will see a lot of useful information about your system and setup such as available [predefined variables](#), loaded PHP modules, and [configuration](#) settings. Take some time and review this important information.

Példa 2-2. Get system information from PHP

```
<?php phpinfo(); ?>
```

Something Useful

Let us do something more useful now. We are going to check what sort of browser the visitor is using. For that, we check the user agent string the browser sends as part of the HTTP request. This information is stored in a [variable](#). Variables always start with a dollar-sign in PHP. The variable we are interested in right now is `$_SERVER['HTTP_USER_AGENT']`.

Megjegyzés: `$_SERVER` is a special reserved PHP variable that contains all web server information. It is known as an autoglobal (or superglobal). See the related manual page on [superglobals](#) for more information. These special variables were introduced in PHP [4.1.0](#). Before this time, we used the older `$HTTP_*_VARS` arrays instead, such as `$HTTP_SERVER_VARS`. Although deprecated, these older variables still exist. (See also the note on [old code](#).)

To display this variable, you can simply do:

Példa 2-3. Printing a variable (Array element)

```
<?php echo $_SERVER['HTTP_USER_AGENT']; ?>
```

A sample output of this script may be:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

There are many [types](#) of variables available in PHP. In the above example we printed an [Array](#) element. Arrays can be very useful.

`$_SERVER` is just one variable that PHP automatically makes available to you. A list can be seen in the [Reserved Variables](#) section of the manual or you can get a complete list of them by looking at the output of the [phpinfo\(\)](#) function used in the example in the previous section.

You can put multiple PHP statements inside a PHP tag and create little blocks of code that do more than just a single echo. For example, if you want to check for Internet Explorer you can do this:

Példa 2-4. Example using [control structures](#) and [functions](#)

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
    echo 'You are using Internet Explorer.<br />';
}
?>
```

A sample output of this script may be:

```
You are using Internet Explorer.<br />
```

Here we introduce a couple of new concepts. We have an [if](#) statement. If you are familiar with the basic syntax used by the C language, this should look logical to you. Otherwise, you should probably pick up an introductory PHP book and read the first couple of chapters, or read the [Language Reference](#) part of the manual. You can find a list of PHP books at <http://www.php.net/books.php>.

The second concept we introduced was the [strpos\(\)](#) function call. [strpos\(\)](#) is a function built into PHP which searches a string for another string. In this case we are looking for 'MSIE' (so-called needle) inside `$_SERVER['HTTP_USER_AGENT']` (so-called haystack). If the needle is found

inside the haystack, the function returns the position of the needle relative to the start of the haystack. Otherwise, it returns **FALSE**. If it does not return **FALSE**, the **if** expression evaluates to **TRUE** and the code within its {braces} is executed. Otherwise, the code is not run. Feel free to create similar examples, with **if**, **else**, and other functions such as **strtoupper()** and **strlen()**. Each related manual page contains examples too. If you are unsure how to use functions, you will want to read both the manual page on [how to read a function definition](#) and the section about [PHP functions](#).

We can take this a step further and show how you can jump in and out of PHP mode even in the middle of a PHP block:

Példa 2-5. Mixing both HTML and PHP modes

```
<?php  
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {  
?>  
<h3>strpos() must have returned non-false</h3>  
<p>You are using Internet Explorer</p>  
<?php  
} else {  
?>  
<h3>strpos() must have returned false</h3>  
<p>You are not using Internet Explorer</p>  
<?php  
}  
?>
```

A sample output of this script may be:

```
<h3>strpos() must have returned non-false</h3>  
<p>You are using Internet Explorer</p>
```

Instead of using a PHP echo statement to output something, we jumped out of PHP mode and just sent straight HTML. The important and powerful point to note here is that the logical flow of the script remains intact. Only one of the HTML blocks will end up getting sent to the viewer depending on the result of [strpos\(\)](#). In other words, it depends on whether the string *MSIE* was found or not.

Dealing with Forms

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts. Please read the manual section on [Variables from outside of PHP](#) for more information and examples on using forms with PHP. Here is an example HTML form:

Példa 2-6. A simple HTML form

```
<form action="action.php" method="post">  
  <p>Your name: <input type="text" name="name" /></p>  
  <p>Your age: <input type="text" name="age" /></p>  
  <p><input type="submit" /></p>  
</form>
```

There is nothing special about this form. It is a straight HTML form with no special tags of any kind. When the user fills in this form and hits the submit button, the *action.php* page is called. In this file you would write something like this:

Példa 2-7. Printing data from our form

```
Hi <?php echo htmlspecialchars($_POST['name']); ?>.  
You are <?php echo (int)$_POST['age']; ?> years old.
```

A sample output of this script may be:

```
Hi Joe. You are 22 years old.
```

Apart from the [htmlspecialchars\(\)](#) and *(int)* parts, it should be obvious what this does. [htmlspecialchars\(\)](#) makes sure any characters that are special in html are properly encoded so people can't inject HTML tags or Javascript into your page. For the age field, since we know it is a number, we can just [convert](#) it to an [integer](#) which will automatically get rid of any stray characters. You can also have PHP do this for you automatically by using the [filter](#) extension. The `$_POST['name']` and `$_POST['age']` variables are automatically set for you by PHP. Earlier we used the `$_SERVER` autoglobal; above we just introduced the `$_POST` autoglobal which contains all POST data. Notice how the *method* of our form is POST. If we used the method *GET* then our form information would live in the `$_GET` autoglobal instead. You may also use the `$_REQUEST` autoglobal, if you do not care about the source of your request data. It contains the merged information of GET, POST and COOKIE data. Also see the [import_request_variables\(\)](#) function.

You can also deal with XForms input in PHP, although you will find yourself comfortable with the well supported HTML forms for quite some time. While working with XForms is not for beginners, you might be interested in them. We also have a [short introduction to handling data received from XForms](#) in our features section.

Using old code with new versions of PHP

Now that PHP has grown to be a popular scripting language, there are a lot of public repositories and libraries containing code you can reuse. The PHP developers have largely tried to preserve backwards compatibility, so a script written for an older version will run (ideally) without changes in a newer version of PHP. In practice, some changes will usually be needed.

Two of the most important recent changes that affect old code are:

- The deprecation of the old `$HTTP_*_VARS` arrays (which need to be indicated as global when used inside a function or method). The following [autoglobal arrays](#) were introduced in PHP [4.1.0](#). They are: `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`, `$_FILES`, `$_ENV`, `$_REQUEST`, and `$_SESSION`. The older `$HTTP_*_VARS` arrays, such as `$_HTTP_POST_VARS`, still exist as they have since PHP 3. Az 5.0.0 változattól kezdődően az [előre definiált hosszú PHP változók](#) létrehozása kikapcsolható a [register_long_arrays](#) direktíva segítségével.
- External variables are no longer registered in the global scope by default. In other words, as of PHP [4.2.0](#) the PHP directive [register_globals](#) is *off* by default in `php.ini`. The preferred method of accessing these values is via the autoglobal arrays mentioned above. Older scripts, books, and tutorials may rely on this directive being on. If it were on, for example, one could use `$id` from the URL `http://www.example.com/foo.php?id=42`. Whether on or off, `$_GET['id']` is available.

For more details on these changes, see the section on [predefined variables](#) and links therein.

What's next?

With your new knowledge you should be able to understand most of the manual and also the various example scripts available in the example archives. You can also find other examples on the `php.net` websites in the links section: <http://www.php.net/links.php>.

To view various slide presentations that show more of what PHP can do, see the PHP Conference Material Sites: <http://conf.php.net/> and <http://talks.php.net/>

II. Telepítés és beállítás

Tartalom

3. [Általános telepítési szempontok](#)
 4. [Telepítés Unix rendszerre](#)
 5. [Telepítés Mac OS X rendszerre](#)
 6. [Telepítés Windows rendszerekre](#)
 7. [PECL kiterjesztések telepítése](#)
 8. [Problémák?](#)
 9. [Futásidőjű beállítások](#)
-

3. Fejezet. Általános telepítési szempontok

Mielőtt hozzálatnál a telepítéshez, meg kell tudnod határozni, hogy mire akarod használni a PHP-t. A [Mire jó a PHP?](#) című fejezetben leírt következő három fő területen használhatod:

- Szerveroldali programozásra
- Parancssori programok írására
- Kliens oldali GUI alkalmazások kifejlesztéséhez.

Az első és legsokványosabb használatához három dologra lesz szükséged: magára a PHP-re, egy webszerverre és egy webböngészőre. Valószínűleg már rendelkezel böngészővel, és az operációs rendszer telepítésétől függően webszervered is lehet (Linux és MacOS X alatt Apache, Windows alatt IIS). Bérelhetsz tárterületet egy cégtől is, ebben az esetben neked nem kell semmit telepítened, csak a PHP szkripteket megírni és feltölteni őket a bérlet webtárhelyedre és nézni a végeredményt a böngészőben.

Ha saját magad telepíted a PHP-t, akkor kétféleképpen csatlakoztathatod a kiszolgálóhoz. A PHP-nek rengeteg kiszolgálóhoz van közvetlen modulinterfésze (SAPI). Ezek a szerverek az Apache, Microsoft Internet Information Server, Netscape és az iPlanet. Ezenkívül sok szerver támogatja az ISAPI-t, a Microsoft modulinterfészét mint pl. az OmniHTTPD. Ha PHP-nek nem létezika a webszerveredhez modul, minden lehetőséged van CGI vagy FastCGI feldolgozóként futtatni. Ez azt jelenti, hogy úgy állítod be a webszerveredet, hogy a PHP CGI programját használja a PHP fájlokra érkezett kérések kiszolgálására.

Ha a PHP-t parancssori programozásra is szeretnéd használni (pl. offline képgenerálást végző szkript, vagy szöveges fájlok feldolgozása az átadott argumentumoktól függően), mindenképpen a futtatható állományra lesz szükséged. További információért lásd a [Parancssori PHP alkalmazások írása](#) című részt. Ebben az esetben nem lesz szükséged szerverre és böngészőre.

A PHP és a PHP-GTK kiterjesztés segítségével kliens oldali GUI alkalmazásokat is fejleszthetsz. Ez teljesen eltérő megközelítés, mint a weboldalak programozása, mivel itt nem HTML kimenetet kell generálni, hanem ablakokat és a bennük levő objektumokat kell kezelni. A PHP-GTK-ról többet is olvashatsz, ha ellátogatsz a [honlapjára](#). A PHP-GTK nem része a hivatalos PHP disztribúciójának.

Ettől a ponttól ez a fejezet a PHP webszerverhez történő telepítésével foglalkozik, Unix és Windows alatt, modulként és CGI feldolgozóként. A futtatható állományról is találsz információt a következő részekben.

A forráskód és a futtatható állományok Windows-ra megtalálhatóak <http://www.php.net/downloads.php> címen. Ha lehet, akkor valamelyik hozzá legközelebb eső tükörszervet vedd igénybe a letöltésekhez.

4. Fejezet. Telepítés Unix rendszerre

Ez a fejezet bemutatja, hogy miként kell feltelepíteni és beállítani a PHP-t Unix rendszeren. Mielőtt elkezded a telepítést, nézd meg a rendszerednek ás szerverednek megfelelő fejezetet is!

Mint ahogy azt kézikönyvünk az [Általános telepítési szempontok](#) című részben felvázolta, ebben a részben legfőképpen a web-centrikus telepítésekkel foglalkozunk, bár a PHP parancssori használatának telepítését is tárgyaljuk.

Számos módja van a PHP telepítésének a Unix rendszereken, vagy egy fordítási és konfigurálási eljárással, vagy különböző előre-csomagolt megoldások használatával. Ez a dokumentáció legfőképpen a fordítási és konfigurálási folyamatokra fekteti a hangsúlyt. Sok Unix-féle rendszernek van valamilyen telepítőrendszer. Ez segít feltelepíteni egy általános konfigurációt, de ha különböző szolgáltatáskészleteket szeretnél (mint például egy biztonságos szerver, vagy különböző adatbázis-meghajtók), magad kell fordítanod a PHP és/vagy a webszervert. Ha nem vagy jártas saját szoftver fordításában, jobb, ha utánanezel, hogy valaki már lefordított-e egy olyan PHP csomagot, amilyet szeretnél.

A fordításhoz nélkülözhetetlen előismeretek és szoftverek:

- Alapvető Unix ismeretek ("make" és a C fordító használata)
- Egy ANSI C fordító
- flex: 2.5.4-es verzió
- bison: 1.28-as (ajánlott), 1.35-ös, vagy 1.75-ös verzió
- A web server
- Bármilyen modul által igényelt elem (mint például a gd vagy pdf könyvtárak, stb.)

Az alapvető konfigurálási folyamatot a **configure** számára átadott parancssori paraméterek szabályozzák. A **./configure --help** parancssal kérhetsz egy listát a rendelkezésre álló opciókról, rövid magyarázatokkal. Ez a kézikönyv külön tárgyalja a különböző opciókat. Az [alapvető opciókat a függelékben](#) találod, a különböző kiterjesztés-specifikus opciók pedig a referencia oldalakon vannak leírva.

Amikor a PHP konfigurálása megtörtént, elkezdheted létrehozni a modulokat és/vagy a végrehajtható állományokat. A **make** parancs ezt el kell tudja intézni. Ha hiba lép fel, és nem tudod elképzelni, hogy miért, olvasd el a [problémákról szóló részt](#).

Apache 1.3.x Unix rendszereken

Ez a rész a PHP Apache-hoz történő telepítéséhez tartalmaz megjegyzéseket és javaslatokat Unix rendszerek esetén. Az [Apache 2-höz tartozó útbaigazítások és javaslatok](#) külön oldalon találhatóak.

A 10. sorban a **configure**-nak átadandó argumentumokat a [az alapvető konfigurációs beállítások listájából](#) valamint a kiterjesztésekhez tartozó beállításokat a kézikönyv megfelelő részeiből válogathatod ki. A verziószámokat itt elhanyagoltuk, hogy bizonyosak legyünk abban, hogy az utasítások nem helytelenek. Az 'xxx' helyébe a fájlok neveiből származó megfelelő értékeket kell majd behelyettesítened.

Példa 4-1. Telepítési parancsok (Apache osztott modul verzió) PHP-hez

1. gunzip apache_xxx.tar.gz
2. tar -xvf apache_xxx.tar
3. gunzip php-xxx.tar.gz
4. tar -xvf php-xxx.tar
5. cd apache_xxx

```
6. ./configure --prefix=/www --enable-module=so
7. make
8. make install
9. cd ../php-xxx

10. Itt konfigurálnod kell a PHP-t. Itt kell testreszabnod a PHP-t
    különféle opciókkal, mint pl. milyen kiterjesztések legyenek
    bekapcsolva. Hajts végre egy ./configure --help parancsot, hogy
    megkapd a használható opciók listáját. Példánkban egy egyszerű
    konfigurációt készítünk Apache 1 és MySQL támogatással.
    Az apxs útvonala különbözhet a példában szereplőtől.
```

```
./configure --with-mysql --with-apxs=/www/bin/apxs
```

```
11. make
12. make install
```

Ha a configure opciókat telepítés után szeretnéd megváltoztatni, csak az utolsó három parancsot kell újra kiadnod. Az új modul betöltéséhez, csak újra kell indítanod az Apache-t. Az Apache újrafordítása nem szükséges.

Figyelj arra, hogy alapesetben a 'make install' a PEAR-t, különböző PHP eszközöket, mint pl. phpize, a PHP CLI-t és egyebeket is telepít.

13. A php.ini állomány beállítása:

```
cp php.ini-dist /usr/local/lib/php.ini
```

PHP opciók beállításához szerkeszd az .ini fájlt. Ha szeretnéd a php.ini-t másolva tenni, a 10. lépésben használd a --with-config-file-path=/ut/vonal

Ha a php.ini-recommended fájlt választod, ne felejtsd el elolvasni a változtatások listáját, mivel az befolyásolhatja a PHP működését.

14. Módosítsd a httpd.conf állományt úgy, hogy a PHP modult töltse be. A LoadModule jobb oldalán szereplő útvonal a rendszereden lévő PHP modul útvonala kell legyen. A fenti make install parancs már el kellett végezze ezt helyetten, de jobb ha megbizonyosodsz róla.

PHP 4 esetén:

```
LoadModule php4_module libexec/libphp4.so
```

PHP 5 esetén:

```
LoadModule php5_module libexec/libphp5.so
```

15. A httpd.conf AddModule részében, valahol a ClearModuleList alá tudd be ezt:

PHP 4 esetén:

```
AddModule mod_php4.c
```

PHP 5 esetén:

```
AddModule mod_php5.c
```

16. Meg kell mondani az Apache-nak, hogy bizonyos kiterjesztéssel rendelkező fájlokat a PHP-vel szerint értelmezzen. Például mondjuk meg az Apache-nak, hogy a .php fájlokat PHP-vel interpretálja. Több kiterjesztést úgy adhatsz meg, hogy egyszerűen szóközzel választod el őket. Példaképpen itt még a .phtml-t adjuk meg.

```
AddType application/x-httdp-php .php .phtml
```

Elég népszerű, hogy a .phps kiterjesztést úgy állítják be, hogy a színezett PHP-t mutassa, ez így érhető el:

```
AddType application/x-httdp-php-source .phps
```

17. Használ a normális eljárást az Apache szerver elindítására. (Le kell állítanod, majd elindítanod, nem elegendő a szerver újratöltése HUP vagy USR1 szignállal.)

A PHP-t statikus objektumként is telepítheted:

Példa 4-2. Telepítési parancsok (Apache statikus modul verzió) PHP-hez

1. gunzip -c apache_1.3.x.tar.gz | tar xf -
2. cd apache_1.3.x
3. ./configure
4. cd ..

5. gunzip -c php-4.x.y.tar.gz | tar xf -
6. cd php-4.x.y
7. ./configure --with-mysql --with-apache=../apache_1.3.x
8. make
9. make install

10. cd ../apache_1.3.x

11. ./configure --prefix=/www --activate-module=src/modules/php4/libphp4.a
(Mindamellett, hogy ezen a ponton a libphp4.a nem létezik, a fenti sor helyes. Nem is kell léteznie, létre lesz hozva.)

12. make
(most kell rendelkezzél egy httpd bináris-sal, amelyet az Apache bin könyvtárába másolhatsz, ha ez az első telepítésed, majd a "make install" parancsot is ki kell adnod)

13. cd ../php-4.x.y
14. cp php.ini-dist /usr/local/lib/php.ini

15. Szerkeszd a /usr/local/lib/php.ini állományt a PHP opciók beállításához.
Szerkeszd a httpd.conf vagy a srm.conf állományt és add hozzá ezt a sort:
AddType application/x-httdp-php .php

Az Apache telepítéstől és a Unix változattól függően, több módon lehet a szervert leállítani vagy újraindítani. Az alábbiakban felsorolunk néhány a tipikus sorokat amelyeket különböző apache/unix konfigurációkon szerver-újraindításra használnak. Az /útvonal/ helyébe helyettesítsd a megfelelő alkalmazás útvonalát.

Példa 4-3. Példa-utasítások az Apache újraindításához

1. Számos Linux és SysV variáns:
/etc/rc.d/init.d/httpd restart

2. Az apachectl szkriptek használata:
/útvonal/apachectl stop

```
/útvonal/apachectl start  
  
3. A httpdctl és a httpsdctl (OpenSSL használata  
esetén), hasonlóan az apachectl-hez:  
/útvonal/httpsdctl stop  
/útvonal/httpsdctl start  
  
4. mod_ssl vagy más SSL szerver használata esetén,  
kézzel kell leállítani és elindítani:  
/útvonal/apachectl stop  
/útvonal/apachectl startssl
```

Az apachectl és a http(s)dctl helyen gyakran változik. Ha van a rendszerben *locate*, *whereis* vagy *which* parancs, ezek segíthetnek megtalálni a szervervezérlő programokat.

Itt egy másik példa a PHP Apache-hoz való fordítására:

```
./configure --with-apxs --with-pgsql
```

Ez létrehozza a libphp4.so osztott könyvtárat ami az Apache-ba töltödik be a httpd.conf LoadModule sora által. A PostgreSQL támogatás bele van építve ebbe a libphp4.so fájlba. library.

```
./configure --with-apxs --with-pgsql=shared
```

Ez létrehoz egy libphp4.so osztott könyvtárat az Apache számára, de egy pgsql.so osztott könyvtérat is készít, amely a PHP-be a php.ini extension direktívája által vagy a szkriptben szereplő [dl\(\)](#) függvénytel kerül betöltésre.

```
./configure --with-apache=/útvonal/apache_source --with-pgsql
```

Ez létrehoz egy libmodphp4.a könyvtárat, egy mod_php4.c-t és néhány velejáró fájlt és bemásolja ezeket az Apache forrásfájának src/modules/php4 könyvtárába. Ezután lefordítod az Apache-t --activate-module=src/modules/php4/libphp4.a opcióval, majd az Apache fordító rendszere létrehozza a libphp4.a-t, majd statikusan belinkeli a httpd binárisba. A PostgreSQL támogatás közvetlenül ebben a httpd binárisban van benne, így a végeredmény egy egyedüli httpd bináris, ami tartalmazza az egész Apache-t és az egész PHP-t.

```
./configure --with-apache=/útvonal/apache_source --with-pgsql=shared
```

Ez hasonló az előzővel, kivéve, hogy a PostgreSQL-t nem építjük közvetlenül bele a végső httpd-be, hanem készítünk egy pgsql.so osztott könyvtárat, amit betölthetsz a PHP-be php.ini-ből, vagy a [dl\(\)](#) függvénytel.

Amikor a PHP különféle fordítási módjai közül választasz, figyelembe kell venned a különböző módszerek előnyeit és hátrányait. Ha osztott fájlt hozol létre, az Apache-t külön fordíthatod, és nem kell minden újrafordítanod, ha valamit hozzá akarsz adni, vagy módosítani a PHP-ben. Ha a PHP az Apache-ba épít被打 be, azt jelenti, hogy a PHP gyorsan töltödik be és gyorsan fut. További információkért lásd [az Apache DSO támogatásról szóló weboldalát](#).

Megjegyzés: Az Apache alapértelmezett httpd.conf fájlja jelenleg egy hasonló részt tartalmaz:

```
User nobody  
Group #-1
```

Amíg nem változtatod meg "Group nogroup"-ra vagy valami hasonlóra (a "Group daemon" elég általános) a PHP nem fog tudni fájlokat megnyitni.

Megjegyzés: Győződj meg róla, hogy az Apache-nak a telepített verzióját adod meg a --

with-apxs=/útvonal/apxs opcióban. NE használd a apache forráskódban szereplő apxs verziót, hanem azt, amelyik ténylegesen telepítve van a rendszeredre.

Apache 2.0 Unix rendszerekben

Ez a rész a PHP Apache 2.0-hoz történő telepítéséhez tartalmaz megjegyzéseket és javaslatokat Unix rendszerek esetén.

Figyelem

Nem ajánljuk az Apache2 threaded MPM-jének alkalmazását éles környezetben. Ehelyett a prefork MPM használata, vagy az Apache 1.3-as változatának használata javallott. Ha kíváncsi vagy a miértekre, olvasd el a kapcsolódó FAQ bejegyzést: [Apache2 with a threaded MPM!](#)

Erősen ajánljuk, hogy tanulmányozd az [Apache dokumentációt](#), hogy alapvetően megértsd az Apache 2.0 Server működését.

PHP és Apache 2.0.x kompatibilitása: A PHP alábbi verziói biztosan működnek az Apache 2.0.x legújabb verziójával:

- PHP 4.3.0 vagy későbbi, elérhető itt: <http://www.php.net/downloads.php>.
- a legfrissebb stabil fejlesztői verzió. A forráskód letöltése: <http://snaps.php.net/php4-latest.tar.gz>, Windows binárisok letöltése: <http://snaps.php.net/win32/php4-win32-latest.zip>.
- prerelease verzió, letölthető itt: <http://qa.php.net/>.
- bármikor letöltheted a PHP-t [anonymous CVS](#)-el.

A PHP ezen verziói kompatibilisek az Apache 2.0.40-ás és későbbi verzióival.

Az Apache 2.0 SAPI-támogatása a PHP 4.2.0-ás verziójával kezdődött. A PHP 4.2.3 az Apache 2.0.39-el működik, ne használj más Apache verziót vele. Az ajánlott konfiguráció: PHP 4.3.0 vagy frissebb, és az Apache2 legfrissebb verziója.

A PHP minden említett verziója továbbra is működik az Apache 1.3.x verziójával.

Töltsd le az [Apache 2.0](#) legújabb verzióját és a PHP-nek egy megfelelő verzióját a fent említett helyekről. Ez a gyors útmutató csak az alapokat tartalmazza az Apache 2.0 és a PHP használatával kapcsolatban. További információért olvasd el az [Apache dokumentációt](#). A verziószámokat itt elhanyagoltuk, hogy bizonyosak legyünk abban, hogy az utasítások nem helytelenek. Az 'NN' helyébe a fájlok neveiből származó megfelelő értékeket kell majd behelyettesítened.

Példa 4-4. Telepítési parancsok (Apache osztott modul verzió) PHP-hez

```
1. gzip -d httpd-2_0_NN.tar.gz
2. tar xvf httpd-2_0_NN.tar
3. gunzip php-NN.tar.gz
4. tar -xvf php-NN.tar
5. cd httpd-2_0_NN
6. ./configure --enable-so
7. make
8. make install
```

Most az Apache 2.0.NN rendelkezésedre áll a /usr/local/apache2 könyvtárban, modul betöltés támogatással, és a standard MPM prefork-al. A telepítés teszteléséhez használd a normális

```
eljárást az Apache szerver elindítására, azaz:  
/usr/local/apache2/bin/apachectl start  
majd állítsd le, hogy folytathasd a PHP konfigurálásval:  
/usr/local/apache2/bin/apachectl stop.
```

9. cd .../php-NN

10. Itt konfigurálnod kell a PHP-t. Itt kell testreszabnod a PHP-t különféle opciókkal, mint pl. milyen kiterjesztések legyenek bekapcsolva. Hajts végre egy ./configure --help parancsot, hogy megkapd a használható opciók listáját. Példánkban egy egyszerű konfigurációt készítünk Apache 2 és MySQL támogatással. Az apxs útvonala különbözet, valójában előfordulhat, hogy a bináris neve apxs2.

```
./configure --with-apxs2=/usr/local/apache2/bin/apxs --with-mysql
```

11. make

12. make install

Ha a configure opciókat telepítés után szeretnéd megváltoztatni, csak az utolsó három parancsot kell újra kiadnod. Az új modul betöltéséhez, csak újra kell indítanod az Apache-t. Az Apache újrafordítása nem szükséges.

Figyelj arra, hogy alapesetben a 'make install' a PEAR-t, különböző PHP eszközöket, mint pl. phpine, a PHP CLI-t és egyebeket is telepít.

13. A php.ini állomány beállítása:

```
cp php.ini-dist /usr/local/lib/php.ini
```

PHP opciók beállításához szerkeszd az .ini fájlt. Ha szeretnéd a php.ini-t másolva tenni, a 10. lépésben használd a --with-config-file-path=/ut/vonal

Ha a php.ini-recommended fájlt választod, ne felejtsd el elolvasni a változtatások listáját, mivel az befolyásolhatja a PHP működését.

14. Módosítsd a httpd.conf állományt úgy, hogy a PHP modult töltse be. A LoadModule jobb oldalán szereplő útvonal a rendszerden lévő PHP modul útvonala kell legyen. A fenti make install parancs már el kellett végezze ezt helyetten, de jobb ha megbizonyosodsz róla.

PHP 4 esetén:

```
LoadModule php4_module libexec/libphp4.so
```

PHP 5 esetén:

```
LoadModule php5_module libexec/libphp5.so
```

15. Meg kell mondani az Apache-nak, hogy bizonyos kiterjesztéssel rendelkező fájlokat a PHP-vel szerint értelmezzen. Például mondjuk meg az Apache-nak, hogy a .php fájlokat PHP-vel interpretálja. Több kiterjesztést úgy adhatsz meg, hogy egyszerűen szóközzel választod el őket. Példaképpen itt még a .phtml-t adjuk meg.

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

Elég népszerű, hogy a .phps kiterjesztést úgy állítják be, hogy a színezett PHP-t mutassa, ez így érhető el:

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

16. Használd a normális eljárást az Apache szerver elindítására, azaz:

```
/usr/local/apache2/bin/apachectl start
```

A fenti lépések végrehajtása után, rendelkezel egy futó Apache 2.0-al, amely támogatja a PHP-t *SAPI* modulként. Természetesen sokkal több konfigurációs opció létezik mind az Apache-hoz, mind a PHP-hez. További információért használd a **./configure --help** parancsot a megfelelő forrásfában. Ha szeretnél egy többszálú Apache 2.0 verziót fordítani, felül kell írnod a standard MPM-Module *prefork*-ot *worker*-el vagy *perchild*-al. Ehhez, a 6. sorban szereplő *configure*-hoz add hozzá a *--with-mpm=worker* vagy a *--with-mpm=perchild* opciót. Bizonyosodj meg a következményekről és arról, hogy tudod mit csinálsz. További információért olvasd el az Apache dokumentáció [MPM-Modules](#) című részét.

Megjegyzés: Ha szeretnél tartalom-egyeztetést (content negotiation) alkalmazni, olvasd el a [kapcsolódó FAQ-t](#).

Megjegyzés: Ahhoz, hogy többszálú Apache verziót tudj készíteni, az operációs rendszered kell támogassa a szálakat. Ez a PHP-nak a kísérleti jellegű Zend Thread Safety (ZTS)-vel való fordítására is vonatkozik. Emiatt nem lesz használható minden kiterjesztés. Ajánlott az Apache-nak a standard *prefork* MPM-modullal való fordítása.

Caudium

PHP 4-et Pike modulként építheted be a [Caudium webszerveréhez](#), a PHP 3 nem támogatja ezt. A PHP 4-et az alábbi egyszerű utasításokat követve lehet telepíteni Caudium alá.

Példa 4-5. Caudium telepítési utasítások

1. Győződj meg arról, hogy a PHP 4 telepítése előtt már telepítve lett a Caudium. A PHP 4 helyes működéséhez a Pike 7.0.268 vagy ennél újabb verziójára lesz szükséged. A példa kedvéért feltételezzük, hogy a Caudium a /opt/caudium/server/könyvtárban van.
2. Lépj át php-x.y.z könyvtárba (x.y.z a PHP verziószáma).
3. `./configure --with-caudium=/opt/caudium/server`
4. `make`
5. `make install`
6. Indítsd újra a Caudium szervert, ha jelenleg fut.
7. Lépj be a grafikus konfigurációs felületre és ott válaszd ki azt a "virtual server" pontot, amelyhez a PHP 4 támogatást akarsz adni.
8. Kattints az "Add Module"-ra és keresd meg majd add hozzá a "PHP 4 Script Support" modult.
9. Ha a dokumentáció azt mondja, hogy 'PHP 4 interpreter isn't available', akkor nézd meg, biztosan újraindítottad a szervert. Ha már ellenőrizted az /opt/caudium/logs/debug/default.1 fájlt mindenféle PHP4.so-val összefüggő hiba kapcsán, akkor nézd meg azt is, hogy a caudium/server/lib/[pike-verzio]/PHP4.so létezik-e.
10. Állítsd be a PHP Script Support modult, ha szükséges.

Természetesen számos kiterjesztéssel együtt fordíthatod le a Caudium szerverhez is a PHP 4-t. A kiterjesztés-specifikus opciókért nézd meg a referenciaoldalakat.

Megjegyzés: Ha MySQL témogatást is szeretnél, akkor figyelj arra, hogy a normál MySQL kliens kódot használd, különben ütközések léphetnek fel, ha a Pike is rendelkezik MySQL támogatással. Ezt a MySQL telepítési könyvtárának (install directory) megadásával teheted meg: `--with-mysql`.

fhttpd megjegyzések

Ahhoz, hogy fhttpd modulként állítsd be a PHP-t, válaszolj "yes"-el a "Build as an fhttpd module?" kérdésre (ez a `--with-fhttpd=DIR` opció a configere-ban) és add meg az fhttpd könyvtárát. Ez alapbeállításban /usr/local/src/fhttpd. Ha fhttpd-t használsz, a PHP felépítése modulként jobb teljesítményt fog nyújtani, több beavatkozási lehetőséget és távoli futtatást biztosít.

Megjegyzés: Az fhttpd támogatása a PHP 4.3.0-tól megszűnt.

Sun, iPlanet and Netscape servers on Sun Solaris

This section contains notes and hints specific to Sun Java System Web Server, Sun ONE Web Server, iPlanet and Netscape server installs of PHP on Sun Solaris.

From PHP 4.3.3 on you can use PHP scripts with the [NSAPI module](#) to [generate custom directory listings and error pages](#). Additional functions for Apache compatibility are also available. For support in current webservers read the [note about subrequests](#).

You can find more information about setting up PHP for the Netscape Enterprise Server (NES) here: <http://benoit.noss.free.fr/php/install-php4.html>

To build PHP with Sun JSWS/Sun ONE WS/iPlanet/Netscape webservers, enter the proper install directory for the `--with-nsapi=[DIR]` option. The default directory is usually /opt/netscape/suitespot/. Please also read /php-xxx-version/sapi/nsapi/nsapi-readme.txt.

1. Install the following packages from <http://www.sunfreeware.com/> or another download site:

```
autoconf-2.13
automake-1.4
bison-1_25-sol26-sparc-local
flex-2_5_4a-sol26-sparc-local
gcc-2_95_2-sol26-sparc-local
gzip-1.2.4-sol26-sparc-local
m4-1_4-sol26-sparc-local
make-3_76_1-sol26-sparc-local
mysql-3.23.24-beta (if you want mysql support)
perl-5_005_03-sol26-sparc-local
tar-1.13 (GNU tar)
```
2. Make sure your path includes the proper directories `PATH=.: /usr/local/bin:/usr/sbin:/usr/bin:/usr/ccs/bin` and make it available to your system `export PATH`.
3. `gunzip php-x.x.x.tar.gz` (if you have a .gz dist, otherwise go to 4).
4. `tar xvf php-x.x.x.tar`

5. Change to your extracted PHP directory: `cd ../php-x.x.x`
6. For the following step, make sure `/opt/netscape/suitespot/` is where your netscape server is installed. Otherwise, change to the correct path and run:

```
./configure --with-mysql=/usr/local/mysql \
--with-nsapi=/opt/netscape/suitespot/ \
--enable-libgcc
```

7. Run **make** followed by **make install**.

After performing the base install and reading the appropriate readme file, you may need to perform some additional configuration steps.

Configuration Instructions for Sun/iPlanet/Netscape. Firstly you may need to add some paths to the `LD_LIBRARY_PATH` environment for the server to find all the shared libs. This can best done in the start script for your webserver. The start script is often located in: `/path/to/server/https-servername/start`. You may also need to edit the configuration files that are located in: `/path/to/server/https-servername/config/`.

1. Add the following line to `mime.types` (you can do that by the administration server):

```
type=magnus-internal/x-httdp-php exts=php
```

2. Edit `magnus.conf` (for servers ≥ 6) or `obj.conf` (for servers < 6) and add the following, shlib will vary depending on your system, it will be something like `/opt/netscape/suitespot/bin/libphp4.so`. You should place the following lines after `mime types init`.

```
Init fn="load-modules" funcs="php4_init,php4_execute,php4_auth_trans" shlib="/opt/n...
```

```
Init fn="php4_init" LateInit="yes" errorString="Failed to initialize PHP!" [php_ini...]
```

(PHP $\geq 4.3.3$) The `php_ini` parameter is optional but with it you can place your `php.ini` in your webserver config directory.

3. Configure the default object in `obj.conf` (for virtual server classes [version 6.0+] in their `vserver.obj.conf`):

```
<Object name="default">
.
.
.
#NOTE this next line should happen after all 'ObjectType' and before all 'AddLog'
Service fn="php4_execute" type="magnus-internal/x-httdp-php" [inikey=value inikey=va...
```

(PHP $\geq 4.3.3$) As additional parameters you can add some special `php.ini`-values, for example you can set a `docroot="/path/to/docroot"` specific to the context `php4_execute` is called. For boolean ini-keys please use 0/1 as value, not "On", "Off", ... (this will not work correctly), e.g. `zlib.output_compression=1` instead of `zlib.output_compression="On"`

4. This is only needed if you want to configure a directory that only consists of PHP scripts (same like a `cgi-bin` directory):

```
<Object name="x-httdp-php">
ObjectType fn="force-type" type="magnus-internal/x-httdp-php"
Service fn=php4_execute [inikey=value inikey=value ...]
</Object>
```

After that you can configure a directory in the Administration server and assign it the style `x-httdp-php`. All files in it will get executed as PHP. This is nice to hide PHP usage by renaming files to `.html`.

5. Setup of authentication: PHP authentication cannot be used with any other authentication. ALL AUTHENTICATION IS PASSED TO YOUR PHP SCRIPT. To configure PHP Authentication for the entire server, add the following line to your default object:

```
<Object name="default">
AuthTrans fn=php4_auth_trans
.
.
.
</Object>
```

6. To use PHP Authentication on a single directory, add the following:

```
<Object ppath="d:\path\to\authenticated\dir\*"
AuthTrans fn=php4_auth_trans
</Object>
```

Megjegyzés: The stacksize that PHP uses depends on the configuration of the webserver. If you get crashes with very large PHP scripts, it is recommended to raise it with the Admin Server (in the section "MAGNUS EDITOR").

CGI environment and recommended modifications in php.ini

Important when writing PHP scripts is the fact that Sun JSWS/Sun ONE WS/iPlanet/Netscape is a multithreaded web server. Because of that all requests are running in the same process space (the space of the webserver itself) and this space has only one environment. If you want to get CGI variables like *PATH_INFO*, *HTTP_HOST* etc. it is not the correct way to try this in the old PHP 3.x way with [getenv\(\)](#) or a similar way (register globals to environment, *\$_ENV*). You would only get the environment of the running webserver without any valid CGI variables!

Megjegyzés: Why are there (invalid) CGI variables in the environment?

Answer: This is because you started the webserver process from the admin server which runs the startup script of the webserver, you wanted to start, as a CGI script (a CGI script inside of the admin server!). This is why the environment of the started webserver has some CGI environment variables in it. You can test this by starting the webserver not from the administration server. Use the command line as root user and start it manually - you will see there are no CGI-like environment variables.

Simply change your scripts to get CGI variables in the correct way for PHP 4.x by using the superglobal *\$_SERVER*. If you have older scripts which use *\$HTTP_HOST*, etc., you should turn on *register_globals* in *php.ini* and change the variable order too (important: remove "E" from it, because you do not need the environment here):

```
variables_order = "GPCS"
register_globals = On
```

Special use for error pages or self-made directory listings (PHP >= 4.3.3)

You can use PHP to generate the error pages for "404 Not Found" or similar. Add the following line to the object in *obj.conf* for every error page you want to overwrite:

```
Error fn="php4_execute" code=XXX script="/path/to/script.php" [inikey=value inikey=value...]
```

where *XXX* is the HTTP error code. Please delete any other *Error* directives which could interfere with yours. If you want to place a page for all errors that could exist, leave the *code* parameter out.

Your script can get the HTTP status code with `$_SERVER['ERROR_TYPE']`.

Another possibility is to generate self-made directory listings. Just create a PHP script which displays a directory listing and replace the corresponding default Service line for `type="magnus-internal/directory"` in `obj.conf` with the following:

```
Service fn="php4_execute" type="magnus-internal/directory" script="/path/to/script.php" [
```

For both error and directory listing pages the original URI and translated URI are in the variables `$_SERVER['PATH_INFO']` and `$_SERVER['PATH_TRANSLATED']`.

Note about [nsapi_virtual\(\)](#) and subrequests (PHP >= 4.3.3)

The NSAPI module now supports the [nsapi_virtual\(\)](#) function (alias: [virtual\(\)](#)) to make subrequests on the webserver and insert the result in the webpage. This function uses some undocumented features from the NSAPI library. On Unix the module automatically looks for the needed functions and uses them if available. If not, [nsapi_virtual\(\)](#) is disabled.

Megjegyzés: But be warned: Support for [nsapi_virtual\(\)](#) is EXPERIMENTAL!!!

CGI és parancssori verzió

Alapbeállításban a PHP CGI programként fordul le. Ez létrehoz egy parancssorból használható értelmezőt, ami CGI feldolgozásra vagy nem webbel kapcsolatos PHP programozásra is használható. Ha egy olyan webszervert futtatsz, amelyhez a PHP modul szintű támogatással rendelkezik, akkor jobb teljesítmény eléréséhez használ inkább azt a módszert. A CGI verzió azonban lehetővé teszi az Apache-ot használóknak, hogy más-más PHP oldalakat más-más user-id-kkel futtassanak.

Figyelem

Ha a PHP-t CGI felületen dolgoztatod, ez a szervereden bizonyos támadási felületeket nyit. Kérlek, olvasd el [CGI biztonság](#) fejezetünket, hogy megtudd, hogy tudod megvédeni magad ezen támadásokkal szemben.

As of PHP 4.3.0, some important additions have happened to PHP. A new SAPI named CLI also exists and it has the same name as the CGI binary. What is installed at `{PREFIX}/bin/php` depends on your configure line and this is described in detail in the manual section named [Using PHP from the command line](#). For further details please read that section of the manual.

Tesztelés

Ha CGI programként fordítottad le a PHP-t, tesztelheted az eredményt azzal, hogy beírod **make test**. Mindig jól jön, ha leteszteled, mert így rögtön észlelhetsz olyan problémákat, amik esetleg csak később bukkantak volna fel.

Szintmérés (benchmarking)

Ha CGI programként fordítottad le a PHP 3-at, tesztelheted a sebességét azzal, hogy beírod a **make bench** parancsot. Ha a be van kapcsolva, talán nem fog jól lefutni a benchmark, a megengedett 30 másodperc alatt. Ez azért van, mert a [set_time_limit\(\)](#) nem használható [safe mode](#)-ban. Használ a [max_execution_time](#) konfigurációs beállítást, hogy megadhasd ezt az időt a szkriptjeidnek. A **make bench** nem veszi figyelembe a [konfigurációs fájlt](#).

Megjegyzés: A **make bench** csak a PHP 3-ban érhető el.

Változók használata

Egyes [szerver által biztosított környezeti változók](#) nincsenek a jelenlegi [CGI/1.1 specifikációban](#) definiálva. Csak a következő változók vannak itt definiálva: *AUTH_TYPE*, *CONTENT_LENGTH*, *CONTENT_TYPE*, *GATEWAY_INTERFACE*, *PATH_INFO*, *PATH_TRANSLATED*, *QUERY_STRING*, *REMOTE_ADDR*, *REMOTE_HOST*, *REMOTE_IDENT*, *REMOTE_USER*, *REQUEST_METHOD*, *SCRIPT_NAME*, *SERVER_NAME*, *SERVER_PORT*, *SERVER_PROTOCOL*, és *SERVER_SOFTWARE*. minden más 'gyártó kiterjesztésnek' minősül.

Telepítés HP-UX rendszerre

Ez a rész a PHP HP-UX rendszerre történő telepítési útmutatóját tartalmazza. (paul_mckay kukac clearwater-it pont co pont uk hozzájárulásával)

Megjegyzés: Ezek a tippek a PHP v4.0.4 és Apache v1.3.9 verziókra vonatkoznak.

1. Szükséged lesz a gzip-re. Töltsd le a <http://hpxx.connect.org.uk/ftp/hpxx/Gnu/gzip-1.2.4a/gzip-1.2.4a-sd-10.20.depot.Z> címről a bináris disztribúciót, tömörítsd ki, és telepítsd swinstall-al.
2. Szükséged lesz a GNU binutils-ra. Töltsd le a <http://gatekeep.cs.utah.edu/ftp/hpxx/Gnu/gcc-2.95.2/gcc-2.95.2-sd-10.20.depot.gz> címről a bináris disztribúciót, tömörítsd ki, és telepítsd swinstall-al.
3. Szükséged lesz a GNU binutils-ra. Töltsd le a <http://hpxx.connect.org.uk/ftp/hpxx/Gnu/binutils-2.9.1/binutils-2.9.1-sd-10.20.depot.gz> címről a bináris disztribúciót, tömörítsd ki, és telepítsd swinstall-al.
4. Szükséged lesz a bison-ra. Töltsd le a <http://hpxx.connect.org.uk/ftp/hpxx/Gnu/bison-1.28/bison-1.28-sd-10.20.depot.gz>, címről a bináris disztribúciót és telepítsd a fentiek szerint.
5. Szükséged lesz a flex-re. Ennek a forráskódját kell letöltened tudod letölteni az egyik <http://www.gnu.org> ükörkiszolgálóról. A non-gnu könyvtárban találod az ftp helyen. Töltsd le az állományt, tömörítsd ki **gunzip**-pel, majd a **tar -xvf** parancssal megkapod a szükséges fájlokat. Lépj be a létrejött flex könyvtárba, és hajsd végre **./configure**, **make**, **make install** parancsokat.

Ha hibaüzeneteket kapsz, feltehetően az a probléma, hogy gcc vagy egyéb eszköz nincs a PATH-ban, tehát add hozzá a PATH-hoz.

6. Töltsd le a PHP és Apache forráskódokat.
7. Alkalmazd rájuk a **gunzip** és a **tar -xvf** parancsokat. Módosítanod kell pár állományt, hogy helyesen leforduljanak.
8. Először a **configure** nevű fájl szorul némi szerkesztésre, mivel úgy tűnik nem tudja követni, hogy HP-UX gépen vagyunk. Lesz még egy jobb módszer ennek a megoldására, de addig is egy olcsó és jól működő javítás a *lt_target=hpxx10.20* beillesztése a **configure** szkript

47286-adik soránál.

9. A következő lépésben az Apache GuessOS fájl szorul javításra. Az apache_1.3.9/src/helpers álból írd át a 89. sort. Ennek jelenlegi tartalma: *echo "hp\${HPUXNAME}-hpxx\${HPUXVER}"; exit 0*, erre kell átirni: *echo "hp\${HPUXNAME}-hp-hpxx\${HPUXVER}"; exit 0*
10. Megosztott modulként nem telepíthető a PHP HP-UX rendszeren, ezért statikusan bele kell fordítanod a szerverbe. Ehhez kövesd az Apache oldalon található utasításokat.
11. A PHP és Apache most már sikeresen lefordult, de az Apache nem fog elindulni. Létre kell hoznod egy új felhasználót az Apache számára, például www vagy apache néven. Utána a 252-253-ik sort kell módosítanod az Apache conf/httpd.conf állományban, tehát ehelyett a két sor helyett:

```
User nobody  
Group nogroup
```

valami hasonló lesz:

```
User www  
Group sys
```

Ez azért szükséges, mivel az Apache nem futtatható a nobody nevű felhasználóval HP-UX alatt. Most már az Apache és a PHP is sikeresen kell működjön.

Telepítés OpenBSD rendszerre

Ez a fejezet a PHP [OpenBSD 3.6](#) rendszerre történő telepítéséhez tartalmaz megjegyzéseket és javaslatokat.

Bináris csomagok használata

A PHP OpenBSD-re való telepítéséhez az ajánlott és legegyszerűbb módszer a bináris csomagok (package) használata. Az alap csomag el van választva a különböző moduloktól, és mindegyik a többiből függetlenül telepíthető és távolítható el. A szükséges fájlokat az OpenBSD CD-den vagy az FTP site-on találod.

A fő csomag, amit telepítened kell az a php4-core-4.3.8.tgz, amely az alapvető motort (és még a gettext-et és az iconv-ot) tartalmazza. Ezután következnek a modul-csomagok, mint pl. a php4-mysql-4.3.8.tgz vagy a php4-imap-4.3.8.tgz. Ezen modulok aktiválásához vagy deaktiválásához a php.ini-ben szereplő **phpxs** parancsot kell használnod.

Példa 4-6. Példa OpenBSD csomagtelepítésre

```
# pkg_add php4-core-4.3.8.tgz  
# /usr/local/sbin/phpxs -s  
# cp /usr/local/share/doc/php4/php.ini-recommended /var/www/conf/php.ini  
    (mysql hozzáadása)  
# pkg_add php4-mysql-4.3.8.tgz  
# /usr/local/sbin/phpxs -a mysql  
    (imap hozzáadása)  
# pkg_add php4-imap-4.3.8.tgz  
# /usr/local/sbin/phpxs -a imap  
    (a mysql és a test eltávolítása)  
# pkg_delete php4-mysql-4.3.8  
# /usr/local/sbin/phpxs -r mysql  
    (a PEAR könyvtárak telepítése)
```

```
# pkg_add php4-pear-4.3.8.tgz
```

Az OpenBSD bináris csomagjairól további információt a [packages\(7\)](#) man oldalon találsz.

Port-ok használata

A PHP-t forráskódiból is fordíthatod, a [ports tree](#) használatával, bár ez csak az OpenBSD-ben járatos felhasználóknak ajánlott. A PHP 4 port két alkönyvtárra van osztva: core és extensions. Az extensions könyvtár minden támogatott PHP modul számára alcsomagokat generál. Ha ezek közül nem szeretnél minden csomagot elkészíteni, használ a **no_*** FLAVOR-t. Ha például az imap modult szeretnéd kihagyni, a FLAVOR-t állítsd be **no_imap**-re.

Általános problémák

- Az Apache alapértelmezett telepítése egy [chroot\(2\) jail](#)-ben fut, amely megtiltja a PHP szkripteknek, hiógy elérjék a /var/www alatti fájlokat. Emiatt létre kell hoznod egy /var/www/tmp könyvtárat, ahol a PHP munkamenet fájlok létrejöhétek, vagy használj egy másik munkamenet kezelőt. Ráadásul az adatbázis socket-eket a jail-en belülre kell helyezni vagy a localhost interfészen lehet hallgatózni. Ha hálózatkezelő függvényeket használsz, egyes fájlokat az /etc könyvtárból, mint pl. az /etc/resolv.conf és az /etc/services a /var/www/etc könyvtárba kell helyezni. Az OpenBSD PEAR csomag automatikusan a helyes chroot könyvtárakba települ, tehát itt nem szükséges semmi különleges módosítás elvégzése. Az OpenBSD Apache-ról további információt az [OpenBSD FAQ](#)-ban találasz.
- A [gd](#) kiterjesztéshez tartozó OpenBSD 3.6 csomag számára telepítve kell legyen az XFree86. Ha nem akarsz néhány font szolgáltatást használni, amit az X11 megkövetel, használ inkább a `php4-gd-4.3.8-no_x11.tgz` csomagot.

Régebbi kiadások

Az OpenBSD régebbi kiadásai a FLAVORS rendszert statikusan beillesztett PHP fordítására használta. Mivel így nehezen lehet bináris csomagokat generálni, ez a módszer nem javasolt. Továbbra is használhatod a régi stabil port tree-ket, de ezt az OpenBSD csapata nem támogatja. Ha bármilyen hozzáfűznivalód van ehhez, a port jelenlegi karbantartója Anil Madhavapeddy (avsm kukac openbsd pont org).

Telepítés Solaris rendszerre

Ez a fejezet a PHP Solaris rendszerre történő telepítéséhez tartalmaz megjegyzéseket és javaslatokat.

Szükséges programok

A Solaris telepítések gyakran nem tartalmaznak C fordítót, és a kapcsolódó eszközöket. Olvasd el [ezt a FAQ-t](#), hogy megtudd miért szükséges ezen szoftverek GNU verzióját használni. A szükséges programok:

- gcc (javasolt, más C fordító is jó lehet)

- make
- flex
- bison
- m4
- autoconf
- automake
- perl
- gzip
- tar
- GNU sed

Természetesen szükséged lehet további szoftverek telepítésére (és esetleg fordítására), ha plusz funkciókat szeretnél elérni (például Oracle vagy MySQL).

Csomagok használata

Egyszerűsítheted a Solaris telepítési eljárást, ha a pkgadd-et használd a szükséges komponensek nagyrészének telepítésénél.

Telepítés Gentoo rendszerre

Ez a rész a PHP [Gentoo Linux](#) rendszerre történő telepítéséhez tartalmaz megjegyzéseket és javaslatokat.

A Portage (emerge) használata

Amellett, hogy a PHP forráskódot letöltheted és saját magad fordíthatod, a Gentoo csomagrendszerének használata a legegyszerűbb és legtisztább módja a PHP telepítésének. Ha nem vagy jártas a Linux alatti szoftver-fordításban, ez lesz számodra a megfelelő módszer.

Ha már felépítetted a Gentoo rendszeredet, valószínűleg már használtad a Portage-t. Az Apache és PHP telepítése nem különbözik az egyéb rendszeres közök telepítésétől.

Az első döntés, amit meg kell hoznod, hogy Apache 1.3.x-t vagy Apache 2.x-t akarsz telepíteni. Mindamellett, hogy mindenekelőtt használható PHP-vel, az alábbi lépések Apache 1.3.x-t fognak használni. Egy másik dolog, amit figyelembe kell venni az, hogy a Portage fád naprakész-e. Ha már rég frissíteted, mindenekelőtt érdemes futtatnod az **emerge sync** parancsot. Így az Apache és a PHP legfrissebb verziót fogod használni.

Ha most minden a helyén van, használd a következő példát az Apache és a PHP telepítéséhez:

Példa 4-7. Példa PHP telepítésére Gentoo rendszeren Apache 1.3-hoz

```
# emerge \<apache-2
# USE="-*" emerge php mod_php
# ebuild /var/db/pkg/dev-php/mod_php-<your PHP version>/mod_php-<your PHP version>.ebuild
# nano /etc/conf.d/apache
    Add hozzá a "-D PHP4"-et az APACHE_OPTS-hoz

# rc-update add apache default
```

```
# /etc/init.d/apache start
```

Az emerge-ről a kiváló [Portage kézikönyv](#)ben olvashatsz, amit a Gentoo weboldalán találsz.

Ha Apache 2-t szeretnél használni, használd egyszerűen az **emerge apache** utasítást a példában.

A konfiguráció beállítása

Az utolsó részben a PHP aktívált modulok nélkül volt emerge-elve. Amikor ez a leírás készült, az egyetlen alapértelmezésben aktivált modul az XML, amely szüksége a [PEAR](#)-hez. Ez valószínű nem az amit szeretnél, és hamarosan rájössz, hogy több aktivált modulra van szükséged, mint pl. MySQL, gettext, GD, stb.

Amikor a PHP-t saját magad fordítod, a **configure** parancs során kell aktiválnod modulokat. A Gentoo esetén egyszerűen megadhatsz USE jelzőket, amelyek automatikusan át lesznek adva a configure szkriptnek. Próbáld ki az alábbi parancsot, hogy lásd milyen USE jelzők adhatók meg az emerge-nek:

Példa 4-8. Az érvényes USE jelzők listájának lekérése

```
# USE="-*" emerge -pv php

[ebuild N      ] dev-php/php-4.3.6-r1  -X -berkdb -crypt -curl -debug -doc
-fdftk -firebird -flash -freetds -gd -gd-external -gdbm -gmp -hardenedphp
-imap -informix -ipv6 -java -jpeg -kerberos -ldap -mcal -memlimit -mssql
-mysql -ncurses -nl -oci8 -odbc -pam -pdfplib -png -postgres -qt -readline
-snmp -spell -ssl -tiff -truetype -xml2 -yaz 3,876 kB
```

Amint a fenti kimenetből észreveheted, PHP sok USE jelzöt figyelembe vesz. Nézd meg őket közelebbről és válaszd ki amire szükséged van. Ha kiválasztasz egy jelzőt, de nem rendelkezel alkalmas könyvtárakkal, a Portage lefordítja azokat. Jó ötlet az **emerge -pv** újból használata, hogy lásd, mit fog a Portage fordítani azért, hogy használhasd a USE jelzőidet. Ha például nincs az X telepítve, és használod USE jelzőként, a Portage a PHP előtt lefordítja az X-et, ami néhány órába telhet.

Ha a PHP-t szeretnéd MySQL, cURL és GD támogatással fordítani, a parancs így fog kinézni:

Példa 4-9. PHP telepítése USE jelzőkkel

```
# USE="-* curl mysql gd" emerge php mod_php
```

Mint ahogy a fenti példa is mutatja, ne felejtsd el a php-t és a mod_php-t emerge-dni. A php a parancssori változatért felelős, a mod_php pedig az Apache modul verziójáért.

Általános problémák

- Ha PHP szkript kimenete helyett a forráskódot kapod meg, valószínűleg megfeledkeztél az `/etc/conf.d/apache` fájl szerkesztéséről. Az Apache-t a `-D PHP4` jelzővel kell elindítani. Ha a jelző jelen van, látnod kell a `ps ax | grep apache` parancs kimenetében, miközben az Apache fut.
- Beillesztési problémák miatt, előfordulhat, hogy nem telepíthetsz több PHP verziót. Ebben az esetben a unmerge-elned kell a régi verziót a `emerge unmerge mod_php-<régi verzió>` parancccsal.
- Ha nem tudod emerge-dni a PHP-t a Java miatt, próbáld meg úgy, hogy a `-*` karaktereket beilleszed a USE jelzők elő, mint ahogyan a fenti példában látható.
- Ha problémád akad az Apache és a PHP konfigurálásával, bármikor kereshetsz a [Gentoo](#)

[fórumokban](#). Próbálj meg az "Apache PHP" kulcsszavakkal keresni.

5. Fejezet. Telepítés Mac OS X rendszerre

Ez a fejezet a PHP Mac OS X rendszerre történő telepítésének dokumentációját tartalmazza. A Mac OS X rendszernek két némileg eltérő változata van, a kliens és szerver. Kézikönyvünk minden rendszerhez tartalmaz telepítési útmutatót. A PHP nem futtatható a MacOS 9 vagy korábbi verziókon.

Csomagok használata

Rendelkezésre áll néhány előre-csomagolt és előre-fordított PHP változat a Mac OS X rendszerekre. Ez sokat tud segíteni egy általános telepítés elvégzésében, de ha más funkciókra is vágysz (például biztonságos szerver funkciókra, vagy más adatbázis támogatására), előfordulhat, hogy még magadnak kell fordítanod a PHP-t és/vagy a szervert. Ha nem vagy tapasztalt az önálló fordításban, megéri utánanézni, hogy elkészítette-e már valaki más azt a csomagot, ami a kívánt funkciókkal rendelkezik.

Fordítás OS X szerveren

Mac OS X szerver telepítés.

1. Töltsd le az Apache és PHP legfrissebb változatait
2. Tömörítsd ki ezeket, és futtasd a **configure** programot az Apache-ra:

```
./configure --exec-prefix=/usr \
--localstatedir=/var \
--mandir=/usr/share/man \
--libexecdir=/System/Library/Apache/Modules \
--iconsdir=/System/Library/Apache/Icons \
--includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
--enable-shared=max \
--enable-module=most \
--target=apache
```

3. Ha szeretnéd, hogy a fordító végezzen optimizációt, add hozzá a következő sort is:

```
setenv OPTIM=-O2
```

4. Lépj be a PHP 4 forrás könyvtárába, és futtasd a configure programot:

```
./configure --prefix=/usr \
--sysconfdir=/etc \
--localstatedir=/var \
--mandir=/usr/share/man \
--with-xml \
--with-apache=/src/apache_1.3.12
```

Ha bármilyen más kiterjesztést szeretnél (MySQL, GD, stb.), szerepeltesd a megfelelő paramétereket itt. A **--with-apache** paraméternek az Apache forráskönyvtárát add meg, például `/src/apache_1.3.12`.

5. Add ki a **make**, majd a **make install** parancsokat. Ez létre fog hozni egy könyvtárat az Apache forráskönyvtárában az `src/modules/php4` alatt.

6. Most újra futtatnod kell a **configure**-t az Apache-ra, hogy beépítsd a PHP 4-ét:

```
./configure --exec-prefix=/usr \
```

```
--localstatedir=/var \
--mandir=/usr/share/man \
--libexecdir=/System/Library/Apache/Modules \
--iconsdir=/System/Library/Apache/Icons \
--includedir=/System/Library/Frameworks/Apache.framework/Versions/1.3/Headers \
--enable-shared=max \
--enable-module=most \
--target=apache \
--activate-module=src/modules/php4/libphp4.a
```

Itt kaphatsz egy üzenetet, ami azt állítja, hogy a `libmodphp4.a` elvadult. Ha ez történik, menj a `src/modules/php4` könyvtárba, az Apache forráskönyvtár alatt, és hajtsd végre ezt: **ranlib libmodphp4.a**. Aztán lépj vissza az Apache forráskönyvtár gyökerébe, és ismét futtasd az előző `configure` parancssort. Ez aktualizálja a link táblát. Add ki újra a `make` és a `make install` parancsokat.

7. Másold, és nevezd át a `php.ini-dist` állományt a `bin` könyvtárba a PHP 4 forráskönyvtárból: `cp php.ini-dist /usr/local/bin/php.ini` vagy (ha nincs local könyvtárad) `cp php.ini-dist /usr/bin/php.ini`.

Fordítás MacOS X kliensre

A következő útmutató segítségedre lesz abban, hogy a PHP-t modulként telepítsd a MacOS X-ben jelenlévő Apache webszerverhez. Ez a verzió támogatja a MySQL és a PostgreSQL adatbáziskezelő rendszereket. Ezeket a tippeket [Marc Liyanage](#) adta szívesen közre..

Figyelem

Légy körültekintő az alábbiakban, mert tönkreteheted az Apache szerveredet!

Telepítéshez az alábbiakat kell tenni:

1. Nyiss egy terminál ablakot.
2. Írd be: `wget http://www.diax.ch/users/liyanage/software/macosx/libphp4.so.gz`, és várd meg, amíg letöltődik.
3. Írd be: `gunzip libphp4.so.gz`.
4. Írd be: `sudo apxs -i -a -n php4 libphp4.so`
5. Most írd be, hogy: `sudo open -aTextEdit /etc/httpd/httpd.conf`.
TextEdit megnyitja a webszerver konfigurációs fájlját. Keresd meg a következő két sort a fájl vége felé (használd a Find parancsot):

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

Töröld ki a két hashmark-ot (#), és ezután mentsd el a fájlt, lépj ki a TextEdit-ből.

6. Végül írd be: `sudo apachectl graceful` hogy újrainduljon a webszerver.

Mostantól a PHP-nak futni kell. Ezt ellenőrizheted, ha a `Sites` könyvtáradba bemásolod a `test.php` fájlt, amiben nincs más csak ez a sor: `<?php phpinfo() ?>`.

Most nyisd meg a `127.0.0.1/~a_sajat_userneved/test.php` címen levő oldalt a webböngésződben. A PHP modul információs táblázatait kell látnod ezen az oldalon.

6. Fejezet. Telepítés Windows rendszerekre

Ez a fejezet a Windows 98/Me és a Windows NT/2000/XP/2003 rendszerekre történő telepítésekre vonatkozik. A PHP nem működik 16 bites környezetben mint például a Windows 3.1. Bizonyos esetekben a támogatott Windows környezetekre Win32 néven hivatkozunk. A PHP a 4.3.0 verziótól kezdődően nem támogatja a Windows 95-öt.

A PHP Windows-ra telepítésének két fő módja van: [kézi telepítővarázslóval](#) történő telepítés.

Ha van Microsoft Visual Studio telepítve a rendszereden, akkor a PHP-t az eredeti forrásból is [lefordíthatod](#).

Amint sikerült a PHP-t telepítened a Windows-odra, [különböző kiterjesztéket](#) is betölthetsz további funkcionálitások eléréséhez.

Figyelem

Számos telepítő-csomag található az Interneten, de egyikük sincs jóváhagyva a PHP.net által, mivel úgy gondoljuk, hogy a kézi telepítés a legjobb választás, hogy a rendszer biztonságos és optimális legyen.

A Windows telepítő

A Windows PHP telepítőprogram letölthető a <http://www.php.net/downloads.php> címről, ez a PHP CGI változatát telepíti is beállítja az IIS, PWS és Xitami szervereket is. A telepítő nem tartalmaz semmilyen külső PHP kiterjesztést (php_*.dll), ezért csak a Windows zip csomagban és a PECL-ben találod meg.

Megjegyzés: Bár a Windows telepítő egyszerű módszer a PHP működésre bírásának, sok oldalról korlátozva van, mert például a kiterjesztések automatikus telepítése nem történik meg. A telepítő használata nem a megefelelő módszer a PHP telepítésére.

Először telepítsd a választott HTTP (web-) szervert a gépedre, és ellenőrizd, hogy jól működik-e.

Futtassd a telepítő exe fájlt, és kövesd a varázsló által adott utasításokat. Kétféle telepítés közül választhatsz - a standard telepítés ésszerű alapbeállításokat ad, az advanced kérdéseket tesz fel (amelyekre tudni kell válaszolni :).

A telepítés varázslója elég információt gyűjt ahhoz, hogy elvégezhesse a php.ini fájl beállítását és konfiguráljon számos webszervert a PHP számára. Az egyik webszerver, amit a PHP telepítő nem konfigurál, az Apache, ezért kézzel kell konfigurálnod.

Mikor a telepítés befejeződött, a varázsló informál arról, hogy szükséges-e a rendszer, ill. a szerver újraindítása, vagy rögtön elkezdheted a munkát a PHP-vel.

Figyelem

Légy tekintettel arra, hogy a PHP ezen telepítési módja nem biztonságos. Ha biztonságos PHP-ra vágysz, akkor jobban teszed, ha a kézikönyv további fejezeteit is elolvasod, és minden beállítást körültekintően elvégzel. Ez az automatikus telepítő egy azonnal használható PHP-t varázsol a gépedre, de nem online szerverekre szánták.

A kézi telepítés lépései

Ez a fejezet segít a PHP kézi telepítésében és konfigurálásában Microsoft Windows-os webszerverek számára. Ehhez a zippelt bináris disztribuciót kell letölteni a

<http://www.php.net/downloads.php> címről.

A Microsoft Windows-hoz számos telepítőkészlet létezik, és mi is terjesztünk egy PHP telepítőt, mi mégis azt javasoljuk, hogy szándd rá az időt, hogy magad telepítsd a PHP-t, mivel így jobban megérted a rendszert és lehetővé teszi számodra, hogy szükség esetén a PHP kiterjesztések telepítését könnyedén evégezd.

Egy előző PHP verzió frissítése: A kéziköny korábbi kiadásai azt javasolták, hogy a különböző ini és DLL fájlokat a rendszerkönyvtárba (pl. C:\WINDOWS) tudd, és bár ez egyszerűsíti telepítést, a frissítést megnehezíti. Azt tanácsoljuk, távolítsd el ezeket a fájlokat, (mint pl. a Windows rendszermappában található php.ini és a PHP-vel kapcsolatoss DLL-ek) mielőtt elkezdenéd az új PHP telepítését. Ne felejtsd el biztonsági másolatot készíteni ezekről arra az esetre, ha törekennéd a rendszert. A régi php.ini is hasznos lehet az új PHP beállításánál. Hamarosan megtanulod, hogy előnyösebb a PHP-vel kapcsolatos fájlokat egy könyvtárban tartani, és ezt a könyvtárat a PATH-ba beírni.

MDAC követelmények: Ha *Windows 98/NT4*-et használsz, töltsd le a Microsoft Data Access Components (MDAC) rendszerhez tartozó legfrissebb verzióját. Az MDAC letölthető a <http://msdn.microsoft.com/data/> címen. Ez azért követelmény, mert a terjesztett Windows binárisokba bele van építve az ODBC támogatás.

Az alábbi lépéseket minden rendszeren végre kell hajtani, a szerver specifikus lépések elvégzése előtt:

Tömörítsd ki a disztribúciós állományt egy általad választott könyvtárba. Ha PHP 4-et telepítesz, a C:\ könyvtárba tömörítsd ki. A zip fájl egy php-4.3.7-Win32-höz hasonló könyvtárba tömörítődik ki. Ha PHP 5-el van dolgod, akkor a C:\php-be tömörítsd ki, mivel ebben az esetben más a zip fájl felépítése. Más útvonalat is választhatasz, de ne tartalmazzon szóközt. (mint pl. a C:\Program Files\PHP) mivel egyes webszerverek összeomlanak, ha így teszel.

A PHP 4-es és 5-ös verziójának kizippelt könyvtárstruktúrája különbözik:

Példa 6-1. A PHP 4 csomag felépítés

```
c:\php
|
+--cli
|   |
|   |-php.exe           -- CLI program - KIZÁRÓLAG parancssori használatra
|
+--dlls                -- segéd DLL-ek néhány kiterjesztés számára
|   |
|   |-expat.dll
|
|   |-fdftk.dll
|
|   |...
|
+--extensions          -- PHP kiterjesztések DLL-jei
|   |
|   |-php_bz2.dll
|
|   |-php_cpdf.dll
|
|   |...
|
+--mibs               -- SNMP segédfájlok
```

```
|  
+--openssl           -- OpenSSL segédfájlok  
|  
+--pdf-related      -- PDF segédfájlok  
|  
+--sapi              -- SAPI (szerver modul támogatás) DLL-ek  
| |  
| |-php4apache.dll  
| |  
| |-php4apache2.dll  
| |  
| |-.  
|  
+--PEAR              -- a PEAR kezdeti másolata  
|  
|  
|-go-pear.bat        -- PEAR telepítő szkript  
|  
|-.  
|  
|-php.exe            -- CGI program  
|  
|-.  
|  
|-php.ini-dist       -- alapértelmezett php.ini beállítások  
|  
|-php.ini-recommended -- ajánlott php.ini beállítások  
|  
|-php4ts.dll         -- alap PHP DLL  
|  
|-.  
|
```

Or:

Példa 6-2. A PHP 5 csomag felépítése

```
c:\php  
|  
+--dev  
| |  
| |-php5ts.lib  
|  
+--ext               -- PHP kiterjesztés DLL-ek  
| |  
| |-php_bz2.dll  
| |  
| |-php_cpdf.dll  
| |  
| |-.  
|  
+--extras  
| |  
| +-mibs             -- SNMP segédfájlok  
| |  
| +-openssl          -- OpenSSL segédfájlok  
| |  
| +-pdf-related      -- PDF segédfájlok  
| |  
| |-mime.magic  
|  
+--pear              -- a PEAR kezdeti másolata  
|
```

```
|  
|-go-pear.bat           -- PEAR telepítő szkript  
|  
|-fdftk.dll  
|  
|-.  
|  
|-php-cgi.exe          -- CGI program  
|  
|-php-win.exe           -- szkripteket hajt végre parancsablak nélkül  
|  
|-php.exe               -- CLI program - KIZÁRÓLAG parancssori használatra  
|  
|-.  
|  
|-php.ini-dist          -- alapértelmezett php.ini beállítások  
|  
|-php.ini-recommended   -- ajánlott php.ini beállítások  
|  
|-php5activescript.dll  
|  
|-php5apache.dll  
|  
|-php5apache2.dll  
|  
|-.  
|  
|-php5ts.dll            -- alap PHP DLL  
|  
|-...
```

Figyelj a különbségekre és az egyezőségekre. A PHP 4 és PHP 5 is tartalmaz CGI és CLI programot, és szerver modulokat, de más könyvtárakban vannak és/vagy más nevük van. Míg a PHP 4 szervermoduljai a sapi könyvtárban van, a PHP 5-nek nincs ilyen könyvtára, és a PHP könyvtár gyökerében találhatóak meg. A PHP 5 kiterjesztések segéd-DLL-jei szintén nem külön könyvtárban vannak.

Megjegyzés: PHP 4 esetén minden fájlt, ami a `dll` és a `sapi` könyvtárban van, a főkönyvtárba kell helyezned (pl. `C:\php`).

Itt egy lista a PHP 4-el és PHP 5-el terjesztett szerver modulokról:

- `sapi/php4activescript.dll` (`php5activescript.dll`) - [ActiveScript motor](#), amely lehetővé teszi PHP beépítése Windows-os alkalmazásokba.
- `sapi/php4apache.dll` (`php5apache.dll`) - Apache 1.3.x modul.
- `sapi/php4apache2.dll` (`php5apache2.dll`) - Apache 2.0.x modul.
- `sapi/php4isapi.dll` (`php5isapi.dll`) - ISAPI modul ISAPI webszerverekhez mint pl. az IIS 4.0/PWS 4.0 vagy újabb.
- `sapi/php4nsapi.dll` (`php5nsapi.dll`) - Sun/iPlanet/Netscape szervermodul.
- `sapi/php4pi3web.dll` (nincs megefelelője PHP 5-ben) - Pi3Web szervermodul.

A szervermodulok jelentősen nagyobb hatékonyságot és további funkcionalitást biztosítanak a CGI binárisokhoz képest. A CLI verzió arra volt tervezve, hogy a PHP-t parancssori programozásra használhasd. A CLI-ről a [PHP parancssori alkalmazásáról szóló fejezetben](#) tudhatsz meg többet.

Figyelem

A SAPI modulok jelentősen tökeletesítve voltak a 4.1-es kiadásban, viszont a régebbi rendszerekben előfordulhat szerverhiba vagy egyéb szervermodulok hibája, mint pl. ASP.

A CGI és CLI binárisoknak szintén szükségük van a php4ts.dll (php5ts.dll) szervermodulra. Bizonyosodj meg róla, hogy ez a fájl elérhető a feltelepített PHP-dben. A DLL keresésének sorrendje:

- Az a könyvtár, ahonnan a php.exe meg lett hívva, vagy abban az esetben, ha SAPI modulként használod, a webszerver könyvtára (pl. C:\Program Files\Apache Group\Apache2\bin).
- Bármilyen könyvtár, amely a Windows PATH környezeti változóban szerepel.

A php4ts.dll / php5ts.dll fájlok elérhetővé tételehez három változat közül kell választanod: másold be a Window rendszerkönyvtárába, másold be a webszerver könyvtárába, vagy a PHP könyvtáradat (C:\php) add hozzá a PATH-hoz. A könnyebb kezelhetőség érdekében, mi az utóbbi változatot használjuk, mert így a jövőben könnyebb lesz frissíteni a PHP-t. A PHP könyvtár PATH-hoz adásáról a [megfelelő FAQ](#)-ban olvashatsz.

A következő lépés egy érvényes konfigurációs fájl (php.ini) létrehozása. Két ini fájl található meg a terjesztett zip fájlban, a php.ini-dist és a php.ini-recommended. Azt tanácsoljuk, hogy használ a php.ini-recommended fájlt, mert ebben a fájlban a beállításokat a hatékonyság és biztonság növelése érdekében állítottuk be. Ezt a jól dokumentált fajlt figyelmesen olvasd, mert a php.ini-dist fájlhoz viszonyított változások erőteljesen befolyásolhatják a PHP működését. Néhány példa: a [display_errors](#) értéke off valamint a [magic_quotes_gpc](#) értéke off. Azon felül, hogy ezeket elolvasd, tanulmányozd az [ini beállításokat](#) és minden opciót magad állíts be. Ha a legnagyobb biztonságot szeretnél elérni, ezt kell tenned, bár a PHP egész jól működik az alapbeállításokkal is. Másold be a kívánt ini fájlt egy olyan könyvtárba, ahol a PHP képes azt megtalálni, és persze nevezd át php.ini-re. A PHP a php.ini-t a [A konfigurációs állomány 9 fejezet](#) részben ismertet könyvtárakban keresi.

Ha Apache 2-t használsz, a legegyszerűbb módszer a PHPIniDir direktíva használata. (olvasd el a [Telepítés Apache 2 rendszerre](#) című oldalt), egyébként pedig a PHPRC környezeti változó a legjobb változás. Ez a folyamat az alábbi [FAQ bejegyzésben](#) van leírva.

Megjegyzés: Ha NTFS-t használisz Windows NT, 2000, XP vagy 2003-on, bizonyosodj meg arról, hogy a webszervert futtató felhasználónak van jog a php.ini olvasásához (pl. tedd olvashatóvá mindenki számára).

A következő lépések elhagyhatóak:

- Szerkeszd az új php.ini fájlodat. Ha tervezed, az [OmniHTTPD](#) használatát, ne kövesd a következő lépést. Állítsd be a [doc_root](#)-ot úgy, hogy a webszervered document_root-jára mutasson. Például:

```
doc_root = c:\inetpub\wwwroot // IIS/PWS esetén  
  
doc_root = c:\apache\htdocs // Apache esetén
```

- Válaszd ki azokat a kiterjeszéket, amelyeket a PHP indulásakor szeretnél betölteni. Olvasd el a [Kiterjesztések telepítése Windows-on](#) című részt, hogy megtudd, hogy állíts be egyet, és hogy mely kiterjesztések vannak beépítve. Ajánlatos előbb az újonnan telepített PHP-t kipróbálni mielőtt még a php.ini-ben aktiválnád a kiterjesztéket.
- PWS és IIS szervereken beállíthatod a [browscap](#) konfigurációs beállítást, hogy a Windows

9x/Me esetén a c:\windows\system\inetsrv\browscap.ini, NT/2000 esetén a c:\winnt\system32\inetsrv\browscap.ini XP esetén pedig a c:\windows\system32\inetsrv\browscap.ini fájlra mutasson. Egy naprakész browscap.ini megszerezéséért olvasd el a következő [FAQ](#)-t.

A PHP most telepítve van a rendszereden. A következő lépésekbe válassz egy webszervert, és vedd rá a PHP futtatására. Válassz egy webszervert a tartalomjegyzékből.

ActiveScript

This section contains notes specific to the ActiveScript installation.

ActiveScript is a windows only SAPI that enables you to use PHP script in any ActiveScript compliant host, like Windows Script Host, ASP/ASP.NET, Windows Script Components or Microsoft Scriptlet control.

As of PHP 5.0.1, ActiveScript has been moved to the [PECL](#) repository. Ezen PECL kiterjesztés DLL állományát letöltheted a [PHP Letöltések](#), vagy a <http://snaps.php.net/> címről.

Megjegyzés: You should read the [manual installation steps](#) first!

After installing PHP, you should download the ActiveScript DLL (php5activescript.dll) and place it in the main PHP folder (e.g. C:\php).

After having all the files needed, you must register the DLL on your system. To achieve this, open a Command Prompt window (located in the Start Menu). Then go to your PHP directory by typing something like cd C:\php. To register the DLL just type regsvr32 php5activescript.dll.

To test if ActiveScript is working, create a new file, named *test.wsf* (the extension is very important) and type:

```
<job id="test">

<script language="PHPScript">
    $WScript->Echo("Hello World!");
</script>

</job>
```

Save and double-click on the file. If you receive a little window saying "Hello World!" you're done.

Megjegyzés: In PHP 4, the engine was named 'ActivePHP', so if you are using PHP 4, you should replace 'PHPScript' with 'ActivePHP' in the above example.

Megjegyzés: ActiveScript doesn't use the default *php.ini* file. Instead, it will look only in the same directory as the .exe that caused it to load. You should create *php-activescript.ini* and place it in that folder, if you wish to load extensions, etc.

Microsoft IIS / PWS

Ez a fejezet az IIS (Microsoft Internet Information Server) szerverekre vonatkozó PHP telepítési útmutatókat tartalmazza.

Figyelem

Ha a PHP-t CGI felületen dolgoztatod, ez a szervereden bizonyos támadási felületeket nyit.

Kérlek, olvass el [CGI biztonság](#) fejezetünket, hogy megtudd, hogy tudod megvédeni magad ezen támadásokkal szemben.

Általános telepítési szempontok IIS-hez

- Először is olvasd el a [kézi telepítés lépéseit](#). Ne hagyd ki ezt a részt mert fontos információkat tartalmaz a PHP Windows-ra való telepítésével kapcsolatban.
- Ha CGI módot használsz, a php.ini-ben a [cgi.force_redirect](#) PHP direktívát 0-ra kell állítanod. A [cgi.force_redirect-el kapcsolatos FAQ-ban](#) fontos információkat találsz. Szintén CGI mód esetén a [cgi.redirect_status_env](#) direktívát érdemes beállítani. Amikor direktívákat szeretnél használni vigyázz, hogy ezek a direktívák a php.ini-ben ne legyenek kikommentezve.
- A PHP 4 CGI neve php.exe, míg PHP 5 esetén php-cgi.exe. A PHP 5 esetén, a php.exe CLI, és nem CGI.
- A Windows PATH környezeti változójához add hozzá a PHP könyvtárat, így a PHP DLL fájljai, futtatható állományai és a php.ini a PHP könyvtárában maradhat, anélkül, hogy a Windows rendszerkönyvtárat fel kellene forgatni. További információért lásd [a PATH beállításáról szóló GyIK-ot](#).
- Az IIS felhasználó (általában IUSR_MACHINENAME) jogosult kell legyen bizonyos fájlok és könyvtárak olvasására, mint például a php.ini, docroot, és a munkamenet tmp könyvtára.
- Figyelj oda, hogy a php.ini-ben az [extension_dir](#) és a [doc_root](#) PHP direktívák helyesen legyenek beállítva. Ezen direktívák értéke függ a rendszertől, ahova a PHP-t telepíted. PHP 4-ben az extension_dir extensions, míg PHP 5 esetén ez ext. PHP 5 esetén az extensions_dir értéke lehet például "c:\php\ext", az IIS doc_root értéke pedig "c:\Inetpub\wwwroot".
- A PHP kiterjesztések DLL állományai, mint például a php_mysql.dll és a php_curl.dll, a letöltött zip csomagban találhatók meg (nem a PHP telepítőben). PHP 5 esetén sok kiterjesztés a a PECL része, ezért a "PECL modulok gyűjteményéből" tölthetők le, ilyenek például a php_zip.dll és php_ssh2.dll állományok. [A PHP fájlokat innen töltheted le](#).
- Amikor megadod a futtatható állományt, jó ha a 'check that file exists' négyzet be van jelölve. Egy kis teljesítménycsökkenéssel az IIS (PWS) meg fogja vizsgálni, hogy a szkript fájl létezik-e, mielőtt meghívja a PHP-t, így a webszerver értelmes 404-es hibákat fog adni, ahelyett, hogy CGI hibákat dobna arról hogy a PHP-től nem kapott kimenetet.

Windows NT/200x/XP és IIS 4 vagy újabb

A PHP kétféle módon telepíthető, CGI binárisként és ISAPI modulként. Mindkét esetben el kell indítanod a Microsoft Management Console programot (lehet 'Internet Services Manager', a Windows NT 4.0 Option Pack ágban vagy a Control Panel=>Administrative Tools-ban Windows 2000/XP alatt). Majd jobbklikk a Web szerver csomópontra (ami valószínűleg 'Default Web Server'-ként jelenik meg), majd válaszd ki a 'Properties' menüpontot.

Ha a CGI binárist szeretnéd használni, akkor:

- A 'Home Directory', 'Virtual Directory', vagy 'Directory' alatt végezd el a következőket:

- A Execute Permissions-t (Futtatási jogosultság) állítsd 'Scripts only'-ra ('Csak szkriptek')
- Klikkelj a 'Configuration' gombra, majd válaszd ki az Application Mappings fület. Kattints az Add-ra, majd az Executable path-ot állítsd be a megfelelő CGI fájlra. Egy példa PHP 5 esetén: C:\php\php-cgi.exe. Kiterjesztésként .php -t adj meg. A 'Method exclusions' mezőt hagyd üresen, és jelüld be a 'Script engine' négyzetet. Ezután nyomj OK-t néhányszor.
- Állíts be megfelelő védelmet. Ez az Internet Service Manager-ben megvan, és ha az NT szervered NTFS fájlrendszerét használ, adj futtatási jogot IUSR_ számára arra a könyvtárra ahol a php.exe / php-cgi.exe található.

Ha ISAPI modulként akarod használni, akkor:

- Ha nem szeretnél PHP segítségével HTTP Autentikáviót végezni, hagyd ki ezt a lépést. Az ISAPI Filters alatt, hozz létre egy új ISAPI filtert. A filter neve legyen PHP, majd add meg a php4isapi.dll / php5isapi.dll fájl útvonalát.
- A 'Home Directory', 'Virtual Directory', vagy 'Directory' alatt végezd el a következőket:
- A Execute Permissions-t (Futtatási jogosultság) állítsd 'Scripts only'-ra ('Csak szkriptek')
- Klikkelj a 'Configuration' gombra, majd válaszd ki az Application Mappings fület. Kattints az Add-ra, majd az Executable path-ot állítsd be a megfelelő ISAPI DLL-re. Egy példa PHP 5 esetén: C:\php\php5isapi.dll. Kiterjesztésként .php -t adj meg. A 'Method exclusions' mezőt hagyd üresen, és jelüld be a 'Script engine' négyzetet. Ezután nyomj OK-t néhányszor.
- Állítsd le teljesen az IIS-t (NET STOP iisadmin)
- Indítsd el újra az IIS-t (NET START w3svc)

IIS 6 esetén (2003 Server), nyisd meg az IIS Manager-t, menj a Web Service Extensions-be, válaszd az "Add a new Web service extension"-t, írj be egy nevet, mint például PHP, válaszd az Add gombot és keresd meg vagy az ISAPI fájlt (php4isapi.dll vagy php5isapi.dll) vagy pedig a CGI-t (php.exe vagy php-cgi.exe), majd jelöld be a "Set extension status to Allowed"-et, végül OK.

Ahhoz, hogy az index.php-t használja alapértelmezett oldalként, a következőket kell tenned: A Documents fülön, válaszd az Add gombot. Írd be: index.php majd OK. Állítsd a sorrendet a Move Up és Move Down gombokkal. Ez hasonló az Apache DirectoryIndex beállításával.

A fenti lépésekkel minden olyan kiterjesztésre hajtsd végre, amelyet a PHP szkriptekhez szeretnél rendelni. A .php a legeltejedtebb, bár egyes régi programok esetén szükséges lehet a .php3.

Ha egy idő után 100%-os CPU kihasználtságot észlel, kapcsold ki a *Cache ISAPI Application* IIS beállítást.

Windows és PWS 4

A PWS 4 Nem támogatja az ISAPI-t, ezért csak a PHP CGI mód használható.

- Szerkeszd át a csatolt pws-php4cgi.reg / pws-php5cgi.reg fájlt (PHP 4 esetén a SAPI könyvtárban, PHP 5 esetén pedig a főkönyvtárban), hogy abban a php.exe / php-cgi.exe fájlra való hivatkozás a telepítésnek megfelelő legyen. A visszaperjeleket meg kell előzze egy másik visszaperjel, pl.: [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map] ".php"="c:\php\php.exe" (PHP 5 esetén C:\php\php-cgi.exe). Most olvaszd be a registry fájlt a rendszerbe, például kattelj rá kétszer.

- A PWS Manager-ben, egy jobbegérgomb kattintás arra a könyvtárra, amelyhez PHP támogatást szeretnél rendelni, és válaszd ki a Properties menüpontot. Jelöld be az 'Execute' négyzetet, és nyomd meg az OK-t.

Windows és PWS/IIS 3

Ezen szervereken a konfigurálás a disztribúciókban szereplő REG fájlok segítségével javasolt (PHP 4 esetén a SAPI könyvtáran szereplő pws-php4cgi.reg, PHP 5 esetén pedig a főkönyvtárban szereplő pws-php5cgi.reg). Szerkeszd ezt a fájlt, bizonyosodj meg arról, hogy a kiterjesztések és a PHP könyvtár megfelel a konfigurációnak, vagy kövesd az alábbi lépéseket, hogy elkészíts ezt saját kezűleg.

Figyelem

Az alábbi lépések a windows registry-ben való közvetlen szerkesztést igénylik. Egyetlen hiba használhatatlanná teheti a rendszered! mindenkorban készíts egy biztonsági másolatot, mielőtt bármi mást tennél. A PHP fejlesztői csapata nem tehető felelőssé, ha kárt teszel a registry-ben!

- Indítsd el a Regedit-et.
- Keresd meg a *HKEY_LOCAL_MACHINE /System /CurrentControlSet /Services /W3Svc /Parameters /ScriptMap* kulcsot.
- Az Edit menüben válaszd ki a *New->String Value* menüpontot.
- Írd be a fájl kiterjesztést, amit használni szeretnél, pl. *.php*
- Klikkelj kétszer az új értékre és írd be a *php.exe* elérési útját, PHP 4 esetén például *C:\php\php.exe "%s" %s*, PHP 5 esetén pedig *C:\php\php-cgi.exe "%s" %s*.
- Ismételd ezeket a lépéseket az összes kiterjesztésre, amit PHP szkripthez szeretnél használni.

A következő lépések nem befolyásolják a web szerver üzembehelyezését, és csak akkor van rá szükség, ha a *php* szkriptjeidet parancsorról is szeretnéd futtatni (pl. *c:\myscripts\test.php*) - vagy a fájlkezelőben duplakattintásra elindítani azokat. Ugord át ezeket a lépéseket, ha azt akarod, hogy duplakattintásra inkább a szövegszerkesztődbe töltődjenek be a PHP szkriptek.

- Keresd meg a: *HKEY_CLASSES_ROOT* kulcsot.
- Az Edit menüben válaszd a *New->Key* menüpontot.
- Nevezd el az új kulcsot az előzőekben megadott kiterjesztés nevére, pl. *.php*
- Válaszd ki az új kulcsot, aztán a jobb oldalon kattints kétszer a "default value" soron, és írd be, hogy *phpfile*.
- Ismételd az utóbbi lépést az összes kiterjesztésre, amit az előző részben beállítottál.
- Most hozz létre ismét egy új kulcsot (*New->Key*) a *HKEY_CLASSES_ROOT* alatt, és nevezd el *phpfile*-ra.
- Válaszd ki az új *phpfile* kulcsot, aztán a jobb oldalon kattints kétszer a "default value" sorra, és írd be, hogy *PHP Script*.
- Kattints jobb gombbal a *phpfile* kulcsra és válaszd ki a *New->Key* menüpontot. Nevezd az új kulcsot *Shell*-nek.
- Kattints jobb gombbal a *Shell* kulcsra és válaszd ki a *New->Key* menüpontot. Nevezd az új

kulcsot *open*-nek.

- Kattints jobb gombbal az *open* kulcsra és válaszd ki a *New->Key* menüpontot. Nevezd az új kulcsot *command*-nak.
- Válaszd ki az új *command* kulcsot, aztán a jobb oldalon kattints kétszer a "default value" soron, és írd be a *php.exe* elérési útját, pl. *c:\php\php.exe -q %1*. Ne felejtsd el a *%1*-et!
- Lépj ki a Regedit-ből.
- Ha Windows alatt PWS szervert használsz, indítsd újra a gépet, hogy újratöltsse a rendszer a registry-t.

PWS és IIS 3 használók így már rendelkeznek egy teljesen funkcionális rendszerrel. IIS 3 használóknak ajánlható Steven Genusa ötletes script map [konfiguráló eszköze](#).

Apache 1.3.x Microsoft Windows-on

Ez a rész a PHP Microsoft Windows rendszeren az Apache 1.3.x szerverhez történő telepítéséhez tartalmaz információkat. Az [Apache 2-re vonatkozó információkat](#) külön oldalon találod.

Megjegyzés: mindenekelőtt olvasd el a [kézi telepítés lépései](#)!

A PHP Apache 1.3.x-hez történő telepítésére Windows alatt két módszer létezik. Az egyik a CGI bináris használata (PHP 4 esetén *php.exe*, PHP 5 esetén pedig *php-cgi.exe*), a másik az Apache modul DLL. Mindkét esetben kell szerkesztened a *httpd.conf* fájlt, hogy tudasd az Apache-al, hogy használja a PHP-t, ezután újra kell indítanod a szervert.

Érdemes itt megjegyeznünk, hogy a SAPI modul Windows-os változata sokkal stabilabb lett, javasoljuk inkább ennek a használatát, mintsem a CGI binárisét, mivel az előbbi sokkal átlátszobbbb és biztonságosabb.

Bár létezik néhány variáció a PHP bekonfigurálására Apache-on, ezek elég egyszerűek ahhoz, hogy az újdonsúltek is használhassák. További konfigurációs direktívákért lásd az Apache dokumentációt.

Miután a konfigurációs fájlt módosítottad, ne felejtsd el úraindítani a szervert, például a **NET STOP APACHE**, majd a **NET START APACHE** parancsokkal, ha az Apache-t Windows kiszolgálóként futtattad, vagy használt a megszokott ikonokat.

Megjegyzés: Ne feledd, hogy amennyiben elérési ótvonalakat adsz meg az Apache konfigurációs állományában Windows alatt, minden hanyattperjelet () át kell alakítanod sima perjellel. Tehát a *c:\directory\file.ext* elérési utat így kell megadnod: *c:/directory/file.ext*!

Telepítés Apache modulként

A következő sorokat kell hozzáadnod az Apache *httpd.conf* fájlhoz:

Példa 6-3. PHP telepítése Apache 1.3.x modulként

Feltételezzük, hogy a PHP a *c:\php* könyvtárba van telepítve. Ha nem így van, akkor írd át az útvonalakat.

PHP 4 esetén:

```
# A LoadModule rész végére
```

```
# Ne felejtsd el ezt az állomány a sapi könyvtárból kimásolni!
LoadModule php4_module "C:/php/php4apache.dll"
```

```
# Az AddModule rész végére
AddModule mod_php4.c
```

PHP 5 esetén:

```
# A LoadModule rész végére
LoadModule php5_module "C:/php/php5apache.dll"
```

```
# Az AddModule rész végére
AddModule mod_php5.c
```

Mindkettő esetén:

```
# Ezt a sort az <IfModule mod_mime.c> feltételes ágon belül helyezd el
AddType application/x-httpd-php .php
```

```
# .phps fájlok szintaxis kiemeléséhez
AddType application/x-httpd-php-source .phps
```

Telepítés CGI binárisként

Ha a PHP comagot a C:\php\ könyvárba zippelted ki, mint ahogy [a kézi telepítés lépései leíró részben](#) írtuk, a CGI bináris konfigurálásához a következő sorokat kell beszúrnod:

Példa 6-4. A PHP és az Apache 1.3.x CGI-ként

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php

# PHP 4 esetén
Action application/x-httpd-php "/php/php.exe"

# PHP 5 esetén
Action application/x-httpd-php "/php/php-cgi.exe"

# add meg azt a könyvtárat, ahol a php.ini található
SetEnv PHPRC C:/php
```

A fenti listában látható második sor a httpd.conf jelenlegi verziójában megtalálható, de ki van kommentelve. Ne felejtsd el a c:/php/ helyébe a PHP tényleges útvonalát behelyettesíteni.

Figyelem

Ha a PHP-t CGI felületen dolgoztatod, ez a szervereden bizonyos támadási felületeket nyit. Kérlek, olvasd el [CGI biztonság](#) fejezetünket, hogy megtudd, hogy tudod megvédeni magad ezen támadásokkal szemben.

Ha szeretnéd a PHP forrásokat kiszínezve megjeleníteni, modul verzió esetén nincs erre megfélő módszer. Ha a CGI telepítést választottad, használhatod a [highlight_file\(\)](#) függvényt. Készíts egy PHP szkriptet, és írd bele ezt: <?php highlight_file('egy_php_script.php'); ?>.

Apache 2.0.x on Microsoft Windows

This section contains notes and hints specific to Apache 2.0.x installs of PHP on Microsoft Windows systems. We also have [instructions and notes for Apache 1.3.x users on a separate page](#).

Megjegyzés: You should read the [manual installation steps](#) first!

Apache 2.2.x Support: Users of Apache 2.2.x may use the documentation below except the appropriate DLL file is named `php5apache2_2.dll` and it only exists as of PHP 5.2.0. See also <http://snaps.php.net/>

Figyelem

Nem ajánljuk az Apache2 threaded MPM-jének alkalmazását éles környezetben. Ehelyett a prefork MPM használata, vagy az Apache 1.3-as változatának használata javallott. Ha kíváncsi vagy a miértekre, olvasd el a kapcsolódó FAQ bejegyzést: [Apache2 with a threaded MPM!](#)

You are highly encouraged to take a look at the [Apache Documentation](#) to get a basic understanding of the Apache 2.0.x Server. Also consider to read the [Windows specific notes](#) for Apache 2.0.x before reading on here.

PHP és Apache 2.0.x kompatibilitása: A PHP alábbi verziói biztosan működnek az Apache 2.0.x legújabb verziójával:

- PHP 4.3.0 vagy későbbi, elérhető itt: <http://www.php.net/downloads.php>.
- a legfrissebb stabil fejlesztői verzió. A forráskód letöltése: <http://snaps.php.net/php4-latest.tar.gz>, Windows binárisok letöltése: <http://snaps.php.net/win32/php4-win32-latest.zip>.
- prerelease verzió, letölthető itt: <http://qa.php.net/>.
- bármikor letöltheted a PHP-t [anonymous CVS](#)-el.

A PHP ezen verziói kompatibilisek az Apache 2.0.40-ás és későbbi verzióival.

Az Apache 2.0 SAPI-támogatása a PHP 4.2.0-ás verziójával kezdődött. A PHP 4.2.3 az Apache 2.0.39-el működik, ne használj más Apache verziót vele. Az ajánlott konfiguráció: PHP 4.3.0 vagy frissebb, és az Apache2 legfrissebb verziója.

A PHP minden említett verziója továbbra is működik az Apache 1.3.x verziójával.

Figyelem

Apache 2.0.x is designed to run on Windows NT 4.0, Windows 2000 or Windows XP. At this time, support for Windows 9x is incomplete. Apache 2.0.x is not expected to work on those platforms at this time.

Download the most recent version of [Apache 2.0.x](#) and a fitting PHP version. Follow the [Manual Installation Steps](#) and come back to go on with the integration of PHP and Apache.

There are two ways to set up PHP to work with Apache 2.0.x on Windows. One is to use the CGI binary the other is to use the Apache module DLL. In either case you need to edit your `httpd.conf` to configure Apache to work with PHP and then restart the server.

Megjegyzés: Ne feledd, hogy amennyiben elérési ótvonalakat adsz meg az Apache konfigurációs állományában Windows alatt, minden hanyattperjelet () át kell alakítanod sima perjellé. Tehát a `c:\directory\file.ext` elérési utat így kell megadnod: `c:/directory/file.ext`!

Installing as a CGI binary

You need to insert these three lines to your Apache `httpd.conf` configuration file to set up the

CGI binary:

Példa 6-5. PHP and Apache 2.0 as CGI

```
ScriptAlias /php/ "c:/php/"
AddType application/x-httpd-php .php

# For PHP 4
Action application/x-httpd-php "/php/php.exe"

# For PHP 5
Action application/x-httpd-php "/php/php-cgi.exe"
```

Figyelem

Ha a PHP-t CGI felületen dolgoztatod, ez a szervereden bizonyos támadási felületeket nyit. Kérlek, olvasd el [CGI biztonság](#) fejezetünket, hogy megtudd, hogy tudod megvédeni magad ezen támadásokkal szemben.

Installing as an Apache module

You need to insert these two lines to your Apache httpd.conf configuration file to set up the PHP module for Apache 2.0:

Példa 6-6. PHP and Apache 2.0 as Module

```
# For PHP 4 do something like this:
LoadModule php4_module "c:/php/php4apache2.dll"
# Don't forget to copy the php4apache2.dll file from the sapi directory!
AddType application/x-httpd-php .php

# For PHP 5 do something like this:
LoadModule php5_module "c:/php/php5apache2.dll"
AddType application/x-httpd-php .php

# configure the path to php.ini
PHPIniDir "C:/php"
```

Megjegyzés: Remember to substitute your actual path to PHP for the c:/php/ in the above examples. Take care to use either php4apache2.dll or php5apache2.dll in your LoadModule directive and *not* php4apache.dll or php5apache.dll as the latter ones are designed to run with [Apache 1.3.x](#).

Megjegyzés: If you want to use content negotiation, read [related FAQ](#).

Figyelem

Don't mix up your installation with DLL files from *different PHP versions*. You have the only choice to use the DLL's and extensions that ship with your downloaded PHP version.

Sun, iPlanet and Netscape servers on Microsoft Windows

This section contains notes and hints specific to Sun Java System Web Server, Sun ONE Web Server, iPlanet and Netscape server installs of PHP on Windows.

From PHP 4.3.3 on you can use PHP scripts with the [NSAPI module](#) to [generate custom directory listings and error pages](#). Additional functions for Apache compatibility are also available. For support in current web servers read the [note about subrequests](#).

CGI setup on Sun, iPlanet and Netscape servers

To install PHP as a CGI handler, do the following:

- Copy php4ts.dll to your systemroot (the directory where you installed Windows)
- Make a file association from the command line. Type the following two lines:

```
assoc .php=PHPScript
ftype PHPScript=c:\php\php.exe %1 %*
```
- In the Netscape Enterprise Administration Server create a dummy shellcgi directory and remove it just after (this step creates 5 important lines in obj.conf and allow the web server to handle shellcgi scripts).
- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/shellcgi, File Suffix:php).
- Do it for each web server instance you want PHP to run

More details about setting up PHP as a CGI executable can be found here:
<http://benoit.noss.free.fr/php/install-php.html>

NSAPI setup on Sun, iPlanet and Netscape servers

To install PHP with NSAPI, do the following:

- Copy php4ts.dll to your systemroot (the directory where you installed Windows)
- Make a file association from the command line. Type the following two lines:

```
assoc .php=PHPScript
ftype PHPScript=c:\php\php.exe %1 %*
```
- In the Netscape Enterprise Administration Server create a new mime type (Category: type, Content-Type: magnus-internal/x-httdp-php, File Suffix: php).
- Edit magnus.conf (for servers >= 6) or obj.conf (for servers < 6) and add the following: You should place the lines after *mime types init*.

```
Init fn="load-modules" funcs="php4_init,php4_execute,php4_auth_trans" shlib="c:/php
Init fn="php4_init" LateInit="yes" errorString="Failed to initialise PHP!" [php_ini]
```

(PHP >= 4.3.3) The *php_ini* parameter is optional but with it you can place your *php.ini* in your webserver config directory.
- Configure the default object in obj.conf (for virtual server classes [Sun Web Server 6.0+] in their vserver.obj.conf): In the *<Object name="default">* section, place this line necessarily after all 'ObjectType' and before all 'AddLog' lines:

```
Service fn="php4_execute" type="magnus-internal/x-httdp-php" [inikey=value inikey=va
```

(PHP >= 4.3.3) As additional parameters you can add some special *php.ini*-values, for example you can set a *docroot="/path/to/docroot"* specific to the context *php4_execute* is called. For boolean ini-keys please use 0/1 as value, not "On", "Off",... (this will not work correctly), e.g. *zlib.output_compression=1* instead of *zlib.output_compression="On"*
- This is only needed if you want to configure a directory that only consists of PHP scripts (same like a cgi-bin directory):

```
<Object name="x-httdp-php">
ObjectType fn="force-type" type="magnus-internal/x-httdp-php"
Service fn=php4_execute [inikey=value inikey=value ...]
</Object>
```

After that you can configure a directory in the Administration server and assign it the style *x-*

httpd-php. All files in it will get executed as PHP. This is nice to hide PHP usage by renaming files to .html.

- Restart your web service and apply changes
- Do it for each web server instance you want PHP to run

Megjegyzés: More details about setting up PHP as an NSAPI filter can be found here:
<http://benoit.noss.free.fr/php/install-php4.html>

Megjegyzés: The stacksize that PHP uses depends on the configuration of the webserver. If you get crashes with very large PHP scripts, it is recommended to raise it with the Admin Server (in the section "MAGNUS EDITOR").

CGI environment and recommended modifications in php.ini

Important when writing PHP scripts is the fact that Sun JSWS/Sun ONE WS/iPlanet/Netscape is a multithreaded web server. Because of that all requests are running in the same process space (the space of the webserver itself) and this space has only one environment. If you want to get CGI variables like *PATH_INFO*, *HTTP_HOST* etc. it is not the correct way to try this in the old PHP 3.x way with **getenv()** or a similar way (register globals to environment, *\$_ENV*). You would only get the environment of the running webserver without any valid CGI variables!

Megjegyzés: Why are there (invalid) CGI variables in the environment?

Answer: This is because you started the webserver process from the admin server which runs the startup script of the webserver, you wanted to start, as a CGI script (a CGI script inside of the admin server!). This is why the environment of the started webserver has some CGI environment variables in it. You can test this by starting the webserver not from the administration server. Use the command line as root user and start it manually - you will see there are no CGI-like environment variables.

Simply change your scripts to get CGI variables in the correct way for PHP 4.x by using the superglobal *\$_SERVER*. If you have older scripts which use *\$HTTP_HOST*, etc., you should turn on *register_globals* in *php.ini* and change the variable order too (important: remove "E" from it, because you do not need the environment here):

```
variables_order = "GPCS"
register_globals = On
```

Special use for error pages or self-made directory listings (PHP >= 4.3.3)

You can use PHP to generate the error pages for "404 Not Found" or similar. Add the following line to the object in *obj.conf* for every error page you want to overwrite:

```
Error fn="php4_execute" code=XXX script="/path/to/script.php" [inikey=value inikey=value]
```

where *XXX* is the HTTP error code. Please delete any other *Error* directives which could interfere with yours. If you want to place a page for all errors that could exist, leave the *code* parameter out. Your script can get the HTTP status code with *\$_SERVER['ERROR_TYPE']*.

Another possibility is to generate self-made directory listings. Just create a PHP script which displays a directory listing and replace the corresponding default Service line for *type="magnus-internal/directory"* in *obj.conf* with the following:

```
Service fn="php4_execute" type="magnus-internal/directory" script="/path/to/script.php" [
```

For both error and directory listing pages the original URI and translated URI are in the variables `$_SERVER['PATH_INFO']` and `$_SERVER['PATH_TRANSLATED']`.

Note about [nsapi_virtual\(\)](#) and subrequests (PHP >= 4.3.3)

The NSAPI module now supports the [nsapi_virtual\(\)](#) function (alias: [virtual\(\)](#)) to make subrequests on the webserver and insert the result in the webpage. The problem is, that this function uses some undocumented features from the NSAPI library.

Under Unix this is not a problem, because the module automatically looks for the needed functions and uses them if available. If not, [nsapi_virtual\(\)](#) is disabled.

Under Windows limitations in the DLL handling need the use of a automatic detection of the most recent `ns-httpdXX.dll`. This is tested for servers till version 6.1. If a newer version of the Sun server is used, the detection fails and [nsapi_virtual\(\)](#) is disabled.

If this is the case, try the following: Add the following parameter to `php4_init` in `magnus.conf/obj.conf`:

```
Init fn=php4_init ... server_lib="ns-httpdXX.dll"
```

where *XX* is the correct DLL version number. To get it, look in the server-root for the correct DLL name. The DLL with the biggest filesize is the right one.

You can check the status by using the [phpinfo\(\)](#) function.

Megjegyzés: But be warned: Support for [nsapi_virtual\(\)](#) is EXPERIMENTAL!!!

OmniHTTPd Server

This section contains notes and hints specific to [OmniHTTPd](#) on Windows.

Megjegyzés: You should read the [manual installation steps](#) first!

Figyelem

Ha a PHP-t CGI felületen dolgoztatod, ez a szervereden bizonyos támadási felületeket nyit. Kérlek, olvass el [CGI biztonság](#) fejezetünket, hogy megtudd, hogy tudod megvédeni magad ezen támadásokkal szemben.

You need to complete the following steps to make PHP work with OmniHTTPd. This is a CGI executable setup. SAPI is supported by OmniHTTPd, but some tests have shown that it is not so stable to use PHP as an ISAPI module.

Important for CGI users: Read the [faq on cgi.force_redirect](#) for important details. This directive needs to be set to *0*.

1. Install OmniHTTPd server.
2. Right click on the blue OmniHTTPd icon in the system tray and select *Properties*
3. Click on *Web Server Global Settings*
4. On the 'External' tab, enter: *virtual = .php | actual = c:\php\php.exe* (use `php-cgi.exe` if installing PHP 5), and use the Add button.
5. On the *Mime* tab, enter: *virtual = wwwserver/stdcgi | actual = .php*, and use the Add button.

6. Click *OK*

Repeat steps 2 - 6 for each extension you want to associate with PHP.

Megjegyzés: Some OmniHTTPd packages come with built in PHP support. You can choose at setup time to do a custom setup, and uncheck the PHP component. We recommend you to use the latest PHP binaries. Some OmniHTTPd servers come with PHP 4 beta distributions, so you should choose not to set up the built in support, but install your own. If the server is already on your machine, use the Replace button in Step 4 and 5 to set the new, correct information.

Sambar Server on Microsoft Windows

This section contains notes and hints specific to the [Sambar Server](#) for Windows.

Megjegyzés: You should read the [manual installation steps](#) first!

This list describes how to set up the ISAPI module to work with the Sambar server on Windows.

- Find the file called `mappings.ini` (in the config directory) in the Sambar install directory.
- Open `mappings.ini` and add the following line under `[ISAPI]`:

Példa 6-7. ISAPI configuration of Sambar

```
#for PHP 4  
*.php = c:\php\php4isapi.dll  
  
#for PHP 5  
*.php = c:\php\php5isapi.dll
```

(This line assumes that PHP was installed in `c:\php`.)

- Now restart the Sambar server for the changes to take effect.
-

Xitami on Microsoft Windows

This section contains notes and hints specific to [Xitami](#) on Windows.

Megjegyzés: You should read the [manual installation steps](#) first!

This list describes how to set up the PHP CGI binary to work with Xitami on Windows.

Important for CGI users: Read the [faq on cgi.force_redirect](#) for important details. This directive needs to be set to `0`. If you want to use `$_SERVER['PHP_SELF']` you have to enable the [cgi.fix_pathinfo](#) directive.

Figyelem

Ha a PHP-t CGI felületen dolgoztatod, ez a szervereden bizonyos támadási felületeket nyit. Kérlek, olvasd el [CGI biztonság](#) fejezetünket, hogy megtudd, hogy tudod megvédeni magad ezen támadásokkal szemben.

- Make sure the webserver is running, and point your browser to xitamis admin console (usually `http://127.0.0.1/admin`), and click on Configuration.
- Navigate to the Filters, and put the extension which PHP should parse (i.e. `.php`) into the

field File extensions (.xxx).

- In Filter command or script put the path and name of your PHP CGI executable i.e. C:\php\php.exe for PHP 4, or C:\php\php-cgi.exe for PHP 5.
 - Press the 'Save' icon.
 - Restart the server to reflect changes.
-

Fordítás forrásból

Mielőtt belevágunk, érdemes megválaszolni azt a kérdést, hogy: "Miért olyan nehéz a fordítás Windows alatt?". Két indokra vezethető vissza:

1. A Windows (még) nem rendelkezik fejlesztők olyan nagy lélekszámú csapatával, akik szabadon szeretnék megosztani egymás közt forrásaikat. Ennek közvetlen következménye, hogy az ilyen fejlesztésekhez szükséges, elengedhetetlen infrastruktúrális beruházások még nem történtek. Nagyjából a hozzáférhető eszközök mindegyikét Unix alól "hozták át", ezért nem kell meglepődni, ha ez az örökség időnként nagyon szembeötlő.
 2. Az itt következő uasítások legtöbbje "csináld és felejtsd el" jellegű, ezért dőlj hátra, és próbáld meg a lehető leghűségesebben követni azokat.
-

Követelmények

Ahhoz, hogy a PHP-t lefordítsd, szükséged lesz egy Microsoft fejlesztői környezetre. A Microsoft Visual C++ 6.0-ot ajánljuk. A letöltött fájlok kicsomagolásához szükséged lesz egy kitömörítő alkalmazásra (pl. Winzip). Ha még nincs unzip programod, letölthetsz egy ingyenes verziót az [InfoZip](#)-től.

Mielőtt elkezded, le kell töltened...

- ..a win32 eszközöket a PHP oldaláról a a <http://www.php.net/extra/win32build.zip> címről.
- ..a PHP által használt DNS név-feloldó forrását a http://www.php.net/extra/bindlib_w32.zip címről. Ez a win32build.zip-ban szereplő resolv.lib helyettesítője.
- Ha a PHP-t Apache modulként akarod fordítani, szükséged lesz az [Apache forrásokra](#) is.

Végül kell maga a PHP 4 forrása. A legutolsó fejlesztői változatra szert tehetsz [anonim CVS](#)-t vagy a [snapshot](#)-ot használva vagy a legfrissebb [forrás tar](#)-t letöltve.

Az egész összerakása

Miután letöltötted a szükséges csomagokat, ki kell tömörítened egy megfelelő helyre.

- Készíts egy munkakönyvtárat, ahová a kitömörített fájlokat teheted, pl: C:\work.
- Készíts egy win32build könyvtárat a munkakönyvtárad alá (C:\work) és ide csomagold ki a win32build.zip fájlt.
- A munkakönyvtáradba (C:\work) hozd létre a bindlib_w32 könyvtárat, majd a bindlib_w32.zip ide tömörítsd ki.
- A letöltött PHP forráskódöt a munkakönyvtáradba csomagold ki (C:\work).

Ezek után a könyvtárstruktúrád így kell kinézzen:

```
+--c:\work
|   |
|   +--bindlib_w32
|   |   |
|   |   +--arpa
|   |   |
|   |   +--conf
|   |   |
|   |   +--...
|   |
|   +--php-4.x.x
|   |   |
|   |   +--build
|   |   |
|   |   +--...
|   |   |
|   |   +--win32
|   |   |
|   |   +--...
|   |
|   +--win32build
|   |   |
|   |   +--bin
|   |   |
|   |   +--include
|   |   |
|   |   +--lib
```

Hozd létre a c:\usr\local\lib könyvtárakat. Másold a bison.simple állományt a c:\work\win32build\bin-ből a c:\usr\local\lib-be.

Megjegyzés: A [Cygwin](#)-t használók kihagyhatják az legutóbbi lépést. Egy megfelelően telepített Cygwin környezet tartalmazza a szükséges bison.simple és bison.exe fájlokat.

Az MVC ++ konfigurálása

A következő lépésekben az fordításhoz szükséges MVC ++ -t állítjuk be. Indítsd el a Microsoft Visual C++ -t, majd a menüből válaszd ki a Tools => Options pontot. A párbeszédablakban válaszd a directories fület. A legördülő listából sorjában válaszd ki az Executables, az Includes és Library files pontokat. Egy jellegzetes listabejegyzések valahogy így mutatnak:

- Executable files: c:\work\win32build\bin, Cygwin-t használóknak: cygwin\bin
 - Include files: c:\work\win32build\include
 - Library files: c:\work\win32build\lib
-

Build resolv.lib

Le kell fordítanod a resolv.lib könyvtárat. Dönts el, hogy debug szimbólumokkal (bindlib - Win32 Debug) vagy anélkül (bindlib - Win32 Release) szeretnél fordítani. Készítsd el a neked megfelelő konfigurációt:

- Grafikus felületet használóknak: indítsd el a VC++ programot, válaszd ki a File => Open Workspace menüpontot, keresd meg a c:\work\bindlib_w32 könyvtárat, válaszd ki a

bindlib.dsw-t. Ezután válaszd a Build=>Set Active Configuration-t, majd add meg a kívánt konfigurációt. Végül válaszd a Build=>Rebuild All menüpontot.

- Parancssort használóknak: bizonyosodj meg arról, hogy a C++ környezeti változók be vannak állítva vagy hogy futtattad a **vcvars.bat**-ot, majd hajtsd végre a következő parancsokat:

- `msdev bindlib.dsp /MAKE "bindlib - Win32 Debug"`
- `msdev bindlib.dsp /MAKE "bindlib - Win32 Release"`

Ezen a ponton kell rendelkezzél egy használható resolv.lib-bel akár a c:\work\bindlib_w32\Debug akár a Release alkönyvtárban. Másold át ezt a fájlt a c:\work\win32build\lib könyvtárba, felülírva a már létező ugyanolyan nevű fájlt.

Fordítás

A kezdésként legjobb egy CGI verziót fordítani.

- Grafikus felületet használóknak: indítsd el a VC++ -t, nyisd meg a php4ts munkaterületetet (File => Open Workspace és válaszd ki a c:\work\php-4.x.x\win32\php4ts.dsw-t). Ezután: Build=>Set Active Configuration, és válaszd ki a kívánt konfigurációt, *php4ts - Win32 Debug_TS* vagy *php4ts - Win32 Release_TS*, és végül: Build=>Rebuild All.
- Parancssort használóknak: győződj meg arról, hogy a szükséges C++ környezeti változók legyenek beállítva vagy a **vcvars.bat** le lett futtatva, ezután add ki valamelyik alábbi parancsot a c:\work\php-4.x.x\win32 könyvtárból:
 - `msdev php4ts.dsp /MAKE "php4ts - Win32 Debug_TS"`
 - `msdev php4ts.dsp /MAKE "php4ts - Win32 Release_TS"`
 - Ennél a pontnál a *Debug_TS* vagy a *Release_TS* alkönyvtárban kell lennie egy használható `php.exe` fájlnak.

Lehetséges a fordító folyamat kissébb testreszabása a main/config.win32.h szerkesztése által. Például megváltoztathatod a `php.ini` helyét, a beépített kiterjesztéseket, valamint a kiterjesztések alapértelmezett helyét.

Ezután készíthetsz egy CLI verziót, ami a [PHP parancssorból történő futtatására](#) van tervezve. A lépések hasonlóak a CGI verzió készítésének lépéseihez, kivéve, hogy a *php4ts_cli - Win32 Debug_TS* vagy a *php4ts_cli - Win32 Release_TS* projekt fájlt kell kiválasztanod. Egy sikeres fordítás után egy `php.exe`-t kell találnod a *Release_TS\cli* vagy a *Debug_TS\cli* könyvtárban.

Megjegyzés: Ha szeretnél használni PEAR-t és a kényelmes parancssori telepítőt, a CLI-SAPI kötelező. További információért a PEAR-ről és a telepítőről olvasd el a [PEAR dokumentációt](#).

Ahhoz, hogy a PHP-nek a Microsoft IIS-hez történő integrálásához szükséges SAPI modult (`php4isapi.dll`) elkészítsd, a konfigurációt a `php4isapi-whatever-config`-ra állítsd be, majd fordítsd le a kívánt dll-t.

Kiterjesztések telepítése Windows-on

A webszerver és a PHP telepítése után valószínűleg néhány kiterjesztést is telepíteni akarsz a szolgáltatásokhoz bővíteni az erőforrásokat. Azokat a kiterjesztéseket, amelyeket a PHP indulásakor szeretnél indítani, a `php.ini`-ben kell megadnod. A szkriptjeidben is betöltheted dinamikusan őket a `dl()` függvény segítségével.

A PHP kiterjesztések DLL-jei a `php_` előtaggal vannak ellátva.

A PHP Windows-os verziójába sok kiterjesztés *be van építve*. Ez annyit jelent, hogy ezen kiterjesztések betöltéséhez *nem* szükségesek további DLL-ek és az [extension](#) direktíva használata. A [Windows-os PHP kiterjesztések](#) táblázat olyan kiterjesztéseket tartalmaz, amelyek működéséhez szükségesek, vagy általában szükségesek további PHP DLL állományok. Itt pedig egy lista a beépített kiterjesztésekéről:

PHP 4-ben (PHP 4.3.11): [BCMath](#), [Caledar](#), [COM](#), [Ctype](#), [FTP](#), [MySQL](#), [ODBC](#), [Overload](#), [PCRE](#), [Session](#), [Tokenizer](#), [WDDX](#), [XML](#) és [Zlib](#)

PHP 5-ben (PHP 5.0.4) a következő kiterjesztésekkel bővül a lista: [DOM](#), [LibXML](#), [Iconv](#), [SimpleXML](#), [SPL](#) és [SQLite](#). Az alábbiak viszont már nincsenek beépítve: [MySQL](#) and [Overload](#).

Az alapértelmezett hely, ahol a PHP keresi a kiterjesztéseket, PHP 4 esetén a `c:\php4\extensions`, PHP 5 esetén pedig `c:\php5`. Hogy megváltoztasd ezt a beállítást, hogy megfeleljen a PHP konfigurációdnak, szerkeszd a `php.ini` fájlt:

- Meg kell változtatnod az [extension_dir](#) beállítást úgy, hogy arra a könyvtárra mutasson, ahol a kiterjesztések vannak vagy ahová a `php_* .dll` fájlokot raktad. Ne feledkezz meg az utolsó visszaperjelről sem. Példa:

```
extension_dir = c:/php/extensions/
```

- Aktiváld az kívánt kiterjesztéseket úgy, hogy a `php.ini`-ben kitörlök a megfelelő `extension=php_* .dll` sor elől a pontosvesszőt (`;`)

Példa 6-8. A [Bzip2](#) PHP kiterjesztés aktiválása Windows-on

```
// ezt a sort cseréld ki ...
;extension=php_bz2.dll

// ... erre
extension=php_bz2.dll
```

- Egyes kiterjesztések a működésükhez további DLL-eket igényelhetnek. Közülük pár megtalálható a disztribúciós csomagban PHP 4 esetén a `C:\php\dlls\` könyvtárban, PHP 5 esetén pedig a főkönyvtárban, mások pedig olyan DLL-eket igényelnek, amelyek nincsenek a csomagban, például az Oracle (`php_oci8.dll`). Ha PHP 4-et telepítesz, másold át a csomagban lévő DLL-eket a `C:\php\dlls` könyvtárból a főkönyvtárba (`C:\php`). Ne felejtsd el a `C:\php` könyvtárat a *PATH*-ba tenni. (ez a folyamat egy külön [FAQ bejegyzésben](#) van leírva).
- Ezen DLL közül egyesek nincsenek benne a PHP disztribúcióban. Részletekért lásd a megfelelő kiterjesztés dokumentációját. A PECL-el kapcsolatban olvasd még el a kézikönyv [PECL kiterjesztések telepítése](#) című részét. A PECL-ben egyre több PHP kiterjesztés található, ezeket a kiterjesztéseket [külön kell letölteni](#).

Megjegyzés: Ha a PHP-nek a szervermodul verzióját futtatod, ne felejtsd el újraindítani a webszervert, hogy a `php.ini`-n végzett változások érvénybe lépjene.

A következő táblázat leír néhány rendelkezésre álló kiterjesztést, és az esetlegesen igényelt DLL-eket.

Táblázat 6-1. PHP kiterjesztések

Kiterjesztés	Leírás	Megjegyzések
php_bz2.dll	bzip2 tömörítési függvények	Nincs
php_calendar.dll	Naptár függvények	Beépítve a PHP 4.0.3-tól
php_cpdf.dll	ClibPDF függvények	Nincs
php_crack.dll	Crack függvények	Nincs
php_ctype.dll	ctype függvények	Beépítve a PHP 4.3.0-tól
php_curl.dll	CURL , kliens URL könyvtári függvények	Szükséges: libeay32.dll, ssleay32.dll (a csomagban)
php_cybercash.dll	Cybercash fizetéssel kapcsolatos függvények	PHP <= 4.2.0
php_db.dll	DBM függvények	Ellenjavallt. Használ a DBA függvényeket helyettük (php_dba.dll).
php_dba.dll	DBA : DataBase (dbm-stílusú) absztrakciós réteg függvények	Nincs
php_dbase.dll	dBase függvények	Nincs
php_dbx.dll	dbx függvények	
php_domxml.dll	DOM XML függvények	PHP <= 4.2.0 esetén szükséges: libxml2.dll (a csomagban) PHP >= 4.3.0 esetén szükséges: iconv.dll (a csomagban)
php_dotnet.dll	.NET függvények	PHP <= 4.1.1
php_exif.dll	EXIF függvények	php_mbstring.dll . A php_exif.dll a php_mbstring.dll után kell betölteni a php.ini-ben.
php_fbsql.dll	FrontBase függvények	PHP <= 4.2.0
php_fdf.dll	FDF : Forms Data Format függvények.	Szükséges: fdftk.dll (a csomagban)
php_filepro.dll	filePro függvények	Csak olvasási hozzáférés
php_ftp.dll	FTP függvények	Beépítve a PHP 4.0.3-tól
php_gd.dll	GD könyvtári kép-függvények	A PHP 4.3.2-től eltávolítva. A truecolor függvények nem érhetők el a GD1-ben, helyettük használ a php_gd2.dll-t.
php_gd2.dll	GD könyvtári kép-függvények	GD2
php_gettext.dll	Gettext függvények	PHP <= 4.2.0 esetén szükséges: gnu_gettext.dll (a csomagban), PHP >= 4.2.3 esetén szükséges: libintl-1.dll, iconv.dll (a csomagban).
php_hyperwave.dll	HyperWave függvények	Nincs
php_iconv.dll	ICONV karakterkészlet	Szükséges: iconv-1.3.dll (a csomagban),

Kiterjesztés	Leírás	Megjegyzések
	konverzió	PHP >= 4.2.1 iconv.dll
php_ifx.dll	Informix függvények	Szükséges: Informix könyvtárak
php_iisfunc.dll	IIS menedzsment függvények	Nincs
php_imap.dll	IMAP POP3 és NNTP függvények	Nincs
php_ingres.dll	Ingres II függvények	Szükséges: Ingres II könyvtárak
php_interbase.dll	InterBase függvények	Szükséges: gds32.dll (a csomagban)
php_java.dll	Java függvények	PHP <= 4.0.6 esetén szükséges: jvm.dll (a csomagban)
php_ldap.dll	LDAP függvények	PHP <= 4.2.0 esetén szükséges: libasl.dll (a csomagban), PHP >= 4.3.0 esetén szükséges: libeay32.dll, ssleay32.dll (a csomagban)
php_mbstring.dll	Multi-Byte String függvények	Nincs
php_mcrypt.dll	Mcrypt tömörítő függvények	Szükséges: libmcrypt.dll
php_mhash.dll	Mhash függvények	PHP >= 4.3.0 esetén szükséges: libmhash.dll (a csomagban)
php_mime_magic.dll	Mimetype függvények	Szükséges: magic.mime (a csomagban)
php_ming.dll	Ming függvények Flash-hez	Nincs
php_msqli.dll	mSQL függvények	Szükséges: msqli.dll (a csomagban)
php_mssql.dll	MSSQL függvények	Szükséges: ntwdplib.dll (a csomagban)
php_mysql.dll	MySQL függvények	PHP >= 5.0.0, szükséges: libmysql.dll (libmysqli.dll, PHP <= 5.0.2) (a csomagban)
php_myqli.dll	MySQLi függvények	PHP >= 5.0.0, szükséges: libmysqli.dll (a csomagban)
php_oci8.dll	Oracle 8 függvények	Szükséges: Oracle 8.1+ client libraries
php_openssl.dll	OpenSSL függvények	Szükséges: libeay32.dll (a csomagban)
php_oracle.dll	Oracle függvények	Szükséges: Oracle 7 kliens könyvtárak
php_overload.dll	Objektum túlterhelési függvények	Beépítve a PHP 4.3.0-tól
php_pdf.dll	PDF függvények	Nincs
php_pgsql.dll	PostgreSQL függvények	Nincs
php_printer.dll	Printer függvények	Nincs
php_shmop.dll	Osztott memória függvények	Nincs

Kiterjesztés	Leírás	Megjegyzések
php_snmp.dll	SNMP get and walk függvények	Csak NT esetén!
php_soap.dll	SOAP függvények	PHP >= 5.0.0
php_sockets.dll	Socket függvények	Nincs
php_sybase_ct.dll	Sybase függvények	Szükséges: Sybase kliens könyvtárak
php_tidy.dll	Tidy függvények	PHP >= 5.0.0
php_tokenizer.dll	Tokenizer függvények	Beépítve a PHP 4.3.0-tól
php_w32api.dll	W32api függvények	Nincs
php_xmlrpc.dll	XML-RPC függvények	PHP >= 4.2.1 esetén szükséges: <code>iconv.dll</code> (a csomagban)
php_xslt.dll	XSLT függvények	PHP <= 4.2.0 esetén szükséges: <code>sablot.dll</code> , <code>expat.dll</code> (a csomagban). PHP >= 4.2.1 esetén szükséges: <code>sablot.dll</code> , <code>expat.dll</code> , <code>iconv.dll</code> (a csomagban).
php_yaz.dll	YAZ függvények	Szükséges: <code>yaz.dll</code> (a csomagban)
php_zip.dll	Zip fájl függvények	Csak olvasási hozzáférés
php_zlib.dll	ZLib tömörítő függvények	Beépítve a PHP 4.3.0-tól

7. Fejezet. PECL kiterjesztések telepítése

Bevezetés a PECL telepítésébe

A PHP kiterjesztések számos módon telepíthetők. A [PECL](#) PHP kiterjesztések gyűjteménye, amelyek a [PEAR](#) struktúrában találhatóak meg. A következő részek bemutatják ezen kiterjesztések telepítésének módját.

Ezek az utasítások `/your/phpsrcdir/-rel` jelölik azt a könyvtárat, ahol a PHP forráskód található, `extname`-el pedig a PECL kiterjesztés nevét. Ezek az utasítások feltételezik a [pear parancs](#) ismeretét.

A megosztott kiterjesztések a `php.ini`-ben az [extension](#) direktíva használatával telepíthetők. Lásd még az [extensions_dir](#) direktívát és a [dl\(\)](#) függvényt. A továbbiakban leírt telepítési módok nem konfigurálják automatikusan a PHP-t, hogy betöltsse ezeket a kiterjesztésekét, ezt kézzel kell megtenni.

PHP modulok fordításánál fontos a szükséges eszközök (autoconf, automake, libtool, stb.) megfelelő verziójainak használata. Olvasd el az [Anonymous CVS útmutatót](#), hogy többet megtudj a szükséges eszközökről és verziókról.

PECL kiterjesztések letöltése

Számos módja van a PECL kiterjesztések letöltésének, mint például:

- <http://pecl.php.net>

Itt olyan információkat is találsz, mint például a változás napló, kiadási információk,

követelmények, javítások, stb. Bár nem minden PECL kiterjesztés rendelkezik honlappal, a legtöbb igen.

- *pear download extname*

A forráskódok letöltésre a [pear parancsot](#) is használhatod. Megadhatod azt is melyik verziót akarod letölteni.

- CVS

Minden PECL állomány megtalálható a CVS-ben. A <http://cvs.php.net/pecl/> címen találd ennek a webes felületét. Ha egyenesen a CVS-ből szeretnél letölteni, hajtsd végre a következő parancsokat (a *cvsread* felhasználó jelszava *phpfi*):

```
$ cvs -d:pserver:cvsread@cvs.php.net:/repository login  
$ cvs -d:pserver:cvsread@cvs.php.net:/repository co pecl/extname
```

- Letöltés Windows-al

A Windows felhasználók letölthetik a lefordított PECL binárisokat a [PHP letöltés](#) oldalon a *Collection of PECL modules* címszó alatt vagy egy [PECL Snapshot](#)-ot (pillanatfelvételt). Ha szeretnéd a PHP-t Windows alatt fordítani, olvasd el a [Win32 Build README](#)-t.

PECL Windows felhasználók részére

Mint bármilyen más PHP kiterjesztés DLL-t, a telepítéshez tess a PECL kiterjesztés DLL-eket az [extension_dir](#) könyvtárba és hivatkozz rá a *php.ini*-ben. Például:

```
extension=php_extname.dll
```

Ezután indítsd újra a webszertvert.

Megosztott PECL kiterjesztések fordítása PEAR-rel

A PEAR-rel könnyen készíthetsz megosztott PHP kiterjesztéseket a [pear parancs](#) használatával. Hajtsd végre a következőt:

```
$ pear install extname
```

Ez letölti az *extname* kiterjesztés forrását, majd lefordítja. Ennek eredménye egy *extname.so* állomány, amelyet aztán beírhatsz a *php.ini*-be.

Amennyiben a rendszer *preferred_state* (előnyben részesített állapot) változója magasabbra van állítva, mint amilyen az *extname*-ból rendelkezésre áll, például ha stabilra van állítva de a kiterjesztés még csak béta állapotban van, vagy módosítsd a *preferred_state*-et a *pear config-set* parancssal, vagy pedig add meg a PECL kiterjesztés egy bizonyos verzióját, valahogyan így:

```
$ pear install extname-0.1.1
```

A pear minden esetben a [extension-dir](#)-be fogja másolni az *extname.so*-t. A *php.ini*-t ennek megfelelően állítsd be.

Megosztott PECL kiterjesztések fordítása phpize-vel

Ha a pear használata nem lehetséges, mint például megosztott PECL kiterjesztések CVS-ből történő fordítása, egy megosztott kiterjesztés létrehozása kézzel is elvégezhető a *phpize* parancssal. A pear parancs alapvetően megtesz ezt, de kézzel is elvégezhető. Feltéve, hogy a forrásfájl neve *extname.tgz*, amely az aktuális könyvtárba lett letöltve, tekintsd a következőt:

```
$ pear download extname
$ gzip -d < extname.tgz | tar -xvf -
$ cd extname
$ phpize
$ ./configure && make
```

Ha minden rendben zajlik, akkor ez létrehoz egy extname.so állományt, és az extname/-en belül található modules/ és/vagy .libs/ könyvtárba teszi. Helyezd ezt a megosztott kiterjesztést (extname.so) a PHP [kiterjesztés könyvtárába](#), és állítsd be a php.ini-t megfelelően.

PECL kiterjesztések fordítása PHP-be statikusan

Ha szeretnéd a kiterjesztést a PHP-be statikusan belefordítani, tudd a kiterjesztés forrását a PHP forrásában található ext / könyvtárba. Például:

```
$ cd /your/phpsrcdir/ext
$ pear download extname
$ gzip -d < extname.tgz | tar -xvf -
$ mv extname-x.x.x extname
$ rm package.xml
```

Ez az alábbi könyvtárat eredményezi:

```
/your/phpsrcdir/ext/extname
```

Most fordítsd a PHP a szokásos módon:

```
$ cd /your/phpsrcdir
$ ./buildconf --force
$ ./configure --help
$ ./configure --with-extname --enable-someotherext --with-foobar
$ make
$ make install
```

Megjegyzés: Ahhoz, hogy a 'buildconf'-ot futtasd, szükséged lesz autoconf 2.13 és automake 1.4+ eszközökre (az autoconf későbbi verzióval is működhet, de nem támogatott).

Az, hogy az *--enable-extname* vagy a *--with-extname* alakot kell használni, függ a kiterjesztéstől. Tipikusan olyan kiterjesztések esetén, amelyek nem igényelnek külső könyvtárakat, az *--enable-* kell használni. Hogy megbizonyosodhass róla, a buldconf után futtasd ezt:

```
$ ./configure --help | grep extname
```

8. Fejezet. Problémák?

Olvasd el a FAQ-t

Néhány probléma bizony gyakran előfordul. A leggyakrabban előforduló gondok és válaszok a [PHP FAQ](#)-ban olvashatóak, amely része e kézikönyvnek.

Egyéb problémák

Ha még mindig elakadsz, talán valaki a PHP telepítési levelezési listán tud segíteni. Jól teszed, ha először megnézed az archívumot, hátha már valaki megválaszolt egy hasonló kérdést. Az archívum

elérhető a support oldalon a <http://www.php.net/support.php> címen. Az angol PHP telepítési levelezőlistára való feliratkozáshoz küldj egy üres levelet a php-install-subscribe@lists.php.net címre. A levelezőlista címe: php-install@lists.php.net. A magyar PHP levelezőlistára való feliratkozáshoz küldj egy subscribe témamegjelöléssel rendelkező levelet a w1-phplista-request@weblabor.hu címre. A levelezőlista címe: w1-phplista@weblabor.hu.

Ha segítséget szeretnél kapni valamelyik levelezőlistán, légy szíves próbálj meg precíz lenni, és minden fontos részletet adj meg a környezetről (operációs rendszer, PHP verziószám, web szerver, miként használod a PHP-t - CGI-ként vagy modulként, stb.). Ezenkívül adj meg elég PHP kódot, hogy a többiek reprodukálni és tesztelni tudják a problémát.

Hibajelentések

Ha úgy gondolod, hogy programhibát találtál a PHP-ben, légy szíves jelentsd a fejlesztőknek (angolul). Lehet, hogy a PHP fejlesztői semmit sem sejtenek felőle, és ha te nem jelentet be, előfordulhat, hogy nem lesz kijavítva. Hibákat a bug-követő rendszeren regisztrálhatsz, melynek címe: <http://bugs.php.net/>. Kérünk, hogy ne küldj hibajelentéseket levelezőlistára vagy személyes levélként. A hibajelentő rendszer alkalmas új szolgáltatás kérésére is.

Mielőtt bármilyen hibajelentést beküldenél, olvasd el a [Hogyan jelentsünk egy hibát](#) című dokumentumot!

9. Fejezet. Futásidejű beállítások

A konfigurációs állomány

A konfigárós fájl (PHP 3-ban `php3.ini`, PHP 4-től pedig egyszerűen `php.ini`) beolvasása a PHP indulásakor történik meg. A szervermodul verziókban ez csak egyszer történik meg, amikor a szerver elindul. A CGI és CLI változatok esetén minden hívás esetén megtörténik

A `php.ini`-t sorrendben a következő helyeken keresi:

- SAPI modul specifikus helyen (Apache 2 esetén a `PHPIniDir` direktíva, CGI és CLI esetén `-c` parancssori opció, NSAPI esetén `php_ini` paraméter, THTTPD esetén `PHP_INI_PATH` környezeti változó határozza meg)
- `HKEY_LOCAL_MACHINE\SOFTWARE\PHP\IniFilePath` (Windows Registry)
- A `PHPRC` környezeti változó
- Aktuális könyvtár (CLI esetén)
- A webszerver könyvtára (SAPI modulok esetén), vagy a PHP könyvtára (egyéb esetben Windows-on)
- Windows könyvtár (`C:\windows` vagy `C:\winnt`) (Windows esetén), vagy a `--with-config-file-path` fordítási opció

Ha a `php-SAPI.ini` fájl létezik, (ahol SAPI a használt SAPI, vagyis a fájlnév például `php-cli.ini` vagy `php-apache.ini`), ezt fogja a `php.ini` helyett használni. A SAPI neve a `php_sapi_name()` függvénytel kérhető le.

Megjegyzés: Az Apache webszerver induláskor a gyökérkönyvtárra vált, ezért a PHP a gyökérkönyvtárból olvassa be a `php.ini`-t, ha létezik.

A kiterjesztések által használt `php.ini` direktívák a megfelelő kiterjesztés oldalain vannak

dokumentálva. Az [alapvető direktívák listája](#) a függelékben található. Valószínűleg nincs minden PHP direktíva dokumentálva ebben a kézikönyvben. A PHP verzióban alkalmazható direktívák teljes listáját megtalálod az elég jól dokumentált `php.ini` fájlban, vagy [a legfrissebb php.ini-t](#) a CVS-ben találod meg.

Példa 9-1. `php.ini` példa

```
; egy nem idézőjelek között lévő pontosvessző (;) után lévő szöveg
; az adott sorban figyelmen kívül lesz hagyva
[php] ; a szekció jelölők (szögletes zárójelek között lévő szövegek)
        ; szintén figyelmen kívül lesznek hagyva
; A logikai értékek lehetnek
;    true, on, yes
; vagy false, off, no, none
register_globals = off
track_errors = yes

; a stringeket macskakörmök közé teheted
include_path = ".:/usr/local/lib/php"

; a visszaperjelek normális karaktereknek számítanak
include_path = ".;c:\php\lib"
```

Konfigurációs beállítások megváltoztatása

PHP Apache modulként

Ha a PHP-t Apache modulként futtatód, a konfigurációs beállításokat az Apache konfigurációs fájljaiban (`httpd.conf`) és a `.htaccess` fájlokban elhelyezett direktívákkal is elvégezheted. Ehhez szükséged lesz "AllowOverride Options" vagy "AllowOverride All" jogosultságra.

A PHP 4 és PHP 5 esetében számos Apache direktíva létezik, amely lehetővé teszi, hogy megváltoztasd a PHP konfigurációt az Apache konfigurációs fájljaiból. Ahhoz hogy megtudd, mely direktívák tartoznak a **PHP_INI_ALL**, **PHP_INI_PERDIR**, vagy **PHP_INI_SYSTEM** kategóriákba, nézd meg a [php.ini direktívákat felsoroló függeléket](#)

Megjegyzés: A PHP 3 esetén, vannak olyan Apache direktívák, amelyek megfelelnek a `php3.ini`-ben szereplő direktívák neveinek, annyiban különböznek, hogy a nevek a "php3_" előtaggal vannak ellátva.

`php_value` név érték

Beállítja a megadott direktíva értékét. Használható **PHP_INI_ALL** és **PHP_INI_PERDIR** típusú direktívák esetén. Ha egy előzőleg beállított értéket szeretnél törölni, használd értékként a `none`-t.

Megjegyzés: Ne használd a `php_value`-t logikai értékek beállítására, helyette a `php_flag`-et használhatod (lásd alább).

`php_flag` név `on|off`

Logikai konfigurációs direktíva beállítására szolgál. Csak **PHP_INI_ALL** és **PHP_INI_PERDIR** típusú direktívák esetén alkalmazható.

`php_admin_value` név érték

Beállítja az adott direktíva értékét. Ez *nem használható* .htaccess fájlokban. A php_admin_value-val beállított direktívák nem bírálhatóak felül .htaccess vagy virtualhost direktívákkal. Egy előzőleg beállított érték törlésére használ értékként a *none*-t.

`php_admin_flag` *név on|off*

Logikai konfigurációs direktíva beállítására szolgál. Ez *nem használható* .htaccess fájlokban. A php_admin_flag beállított direktívák nem bírálhatóak felül .htaccess vagy virtualhost direktívákkal.

Példa 9-2. Példa Apache konfigurációra

```
<IfModule mod_php5.c>
    php_value include_path ".:/usr/local/lib/php"
    php_admin_flag safe_mode on
</IfModule>
<IfModule mod_php4.c>
    php_value include_path ".:/usr/local/lib/php"
    php_admin_flag safe_mode on
</IfModule>
<IfModule mod_php3.c>
    php3_include_path ".:/usr/local/lib/php"
    php3_safe_mode on
</IfModule>
```

Figyelem

A PHP konstansok nem léteznek PHP-n kívül. A httpd.conf-ban például nem használhatsz PHP konstansokat, mint például az **E_ALL** vagy **E_NOTICE**, az **error_reporting** direktíva beállításához, mivel azoknak nem lesz értelme, ezért 0-nak értékelődnek ki. Ehelyett használ a megfelelő bitmaszk értékeket. Ezek a konstansok `php.ini`-ben használhatóak.

A PHP konfiguráció megváltoztatása a Windows registry segítségével

Ha a PHP-t Windows-on futtatod, a konfigurációs értékek megváltoztathatók könyvtáránként a Windows registry használatával. A konfigurációs értékek a *HKLM\SOFTWARE\PHP\Per Directory Values* registry kulcsban vannak tárolva a könyvtáraknak megfelelő alkulcsokban. A *c:\inetpub\wwwroot* könyvtár konfigurációs értékei a *HKLM\SOFTWARE\PHP\Per Directory Values\c:\inetpub\wwwroot* kulcsban lesznek tárolva. A könyvtár beállításai érvényesek lesznek minden olyan szkriptre ami az adott könyvtárban, vagy annak bármely alkönyvtárában található. Egy kulcs alatt szereplő érték neve egy PHP konfigurációs direktíva nev kell legyen, az érték pedig a string. Az értékekben szereplő PHP konstansok nem lesznek értelmezve. Csak `PHP_INI_USER` konfigurációs értékek állíthatók be ily módon, `PHP_INI_PERDIR` értékek nem.

Egyéb interfészek a PHP-hez

Attól függetlenül, hogyan futtatod a PHP, bizonyos értékeket a szkripted futásidéjében is megváltoztathatsz az **ini_set()** függvénytellyel. További információért lásd az **ini_set()** függvény dokumentációját.

Ha szeretnéd megtudni a rendszer teljes konfigurációs beállításainak listáját érakekkel együtt, hív meg a **phpinfo()** függvényt és vizsgáld a kapott oldalt. Az sajátos konfigurációs direktívák értékét futásidőben a **ini_get()** vagy a **get_cfg_var()** függvény segítségével állapíthatod meg.

III. A nyelv alapjai

Tartalom

10. [Alapvető szintaxis](#)
 11. [Típusok](#)
 12. [Változók](#)
 13. [Állandók](#)
 14. [Kifejezések](#)
 15. [Operátorok](#)
 16. [Vezérlési szerkezetek](#)
 17. [Függvények](#)
 18. [Osztályok, objektumok \(PHP 4-ben\)](#)
 19. [Osztályok és objektumok \(PHP 5\)](#)
 20. [Kivételek \(Exceptions\)](#)
 21. [Referenciák](#)
-

10. Fejezet. Alapvető szintaxis

Escape szekvencia HTML-ben

Amikor a PHP feldolgoz egy fájlt, akkor a nyitó és a záró tag-eket keresi, amelyek megmondják a PHP-nek, hogy kezdje el ill. fejezze be a közöttük lévő kódot értelmezni. Ez a kódértelmező mód teszi lehetővé azt, hogy a PHP kódokat mindenféle dokumentumba be tudjuk ágyazni, mivel minden, ami a nyitó és záró tag-eken kívül esik, a PHP értelmező figyelmen kívül hagy. PHP kódot legtöbbször HTML dokumentumokba ágyazva fogsz látni, mint ebben a példában is.

```
<p>Ezt figyelmen kívül hagyja.</p>
<?php echo 'Ezt viszont értelmezi.'; ?>
<p>Ezt szintén figyelmen kívül hagyja.</p>
```

Bonyolultabb struktúrákat is alkalmazhatsz:

Példa 10-1. Haladó escape-elés

```
<?php
if ($kifejezes) {
    ?>
    <strong>Ez igaz.</strong>
    <?php
} else {
    ?>
    <strong>Ez hamis.</strong>
    <?php
}
?>
```

Ez az elvártnak megfelelően működik, mivel a PHP amikor ?> záró tag-et talál, egyszerűen elkezdi a kimenetre írni ami ezután következik, míg nem talál egy másik nyitó tag-et. Az adott példa természetesen nem egy túl hasznos alkalmazást mutat be, de mikor nagy szövegrészleteket akarsz kiiratni, akkor a PHP módból való kilépés sokkal hatékonyabb, mint [echo\(\)](#)-val vagy [print\(\)](#)-el kiiratni az egészet.

Négy különböző nyitó és záró tag pár létezik. Kettő közülük, a <?php ?> és a <script language="php"> </script>, mindig rendelkezésre állnak. A másik kettő a rövid tag-ek és az ASP-szerű tag-ek, ezek ki és be kapcsolhatók a php.ini konfigurációs fájlban. Míg egyesek a rövid tag-eket és az ASP tag-eket kényelmesnek tartják, ezek kevésbé hordozhatóvá teszik a programokat,

ezért használatuk nem ajánlott.

Megjegyzés: Ha XML-be, vagy XHTML-be ágyazol PHP kódot, akkor a <?php ?> tag-eket kell használnod, hogy ne téj el a szabványtól.

Példa 10-2. PHP nyitó és záró tag-ek

```
1.  <?php echo 'ha XHTML vagy XML dokumentumokat akarsz szolgáltatni, tudd így'; ?>
2.  <script language="php">
    echo 'egyes szerkesztők (mint pl FrontPage) nem szeretik
          a feldolgozó utasításokat';
</script>
3.  <? echo 'ez a legegyszerűbb, egy SGML processing utasítás'; ?>
<?= kifejezes ?> Ez egy rövidítése ennek: "<? echo kifejezes ?>"
```

4. <% echo 'Használhatsz ASP-stílusú tag-eket'; %>
<%= \$valtozo; # Ez egy rövidítése ennek: "<% echo . . ." %>

Míg az első és a második példában tárgyalt tag-ek minden rendelkezésre állnak, az elsőt alkalmazzák széleskörben, és az is az ajánlott.

A rövid tag-ek (harmadik példa) csak akkor használhatók, ha engedélyezve vannak a [short_open_tag](#) php.ini konfigurációs fájl direktívával, vagy ha a PHP az --enable-short-tags kapcsolóval volt fordítva.

Megjegyzés: Ha PHP 3-at használsz, akkor a rövid tag-eket a [short_tags\(\)](#) függvénytelivel engedélyezheted. *Ez csak PHP 3-ban lehetséges!*

Az ASP-szerű tag-ek (negyedik példa) csak akkor használhatók, ha engedélyezve vannak az [asp_tags](#) php.ini konfigurációs fájl direktívával.

Megjegyzés: Az ASP tag-ek lehetősége a PHP 3.0.4-től létezik.

Megjegyzés: A rövid nyitójelölések használatát kerülni kell újrafelhasználandó, széles körű terjesztésre szánt könyvtárak vagy programok fejlesztésekor, illetve olyan alkalmazások megírásakor, amelyek üzemeltetése felett a program írójának nincs ellenőrzése. Ennek oka, hogy a rövid nyitójelölések nem minden konfigurációban használhatók, így csökkenti a hordozhatóságot. Hordozható, újrafelhasználható komponensek esetén *ne* használd a rövid nyitójelöléseket!

Utasítások elválasztása

Hasonlóan a C-hez vagy a Perl-hez, a PHP megköveteli, hogy minden utasítás pontosvesszővel végződön. A PHP záró tag automatikusan magába foglal egy pontosvesszöt is, ezért nem szükséges a PHP rész utolsó utasítását pontosvesszővel zárnod. A záró tag magába foglal egy azonnali soremelést is.

```
<?php
    echo 'Ez egy teszt';
?>

<?php echo 'Ez egy teszt' ?>
```

Megjegyzés: A záró tag a fájl végén opcionális, egyes esetekben az elhagyása hasznos, pl. kimenet pufferelés és [include\(\)](#) vagy [require\(\)](#) együttes használata esetén.

Megjegyzések - kommentek

A PHP támogatja a 'C', 'C++' és Unix shell-szerű (Perl stílusú) megjegyzéseket. Például:

```
<?php
    echo 'Ez egy teszt'; // egysoros c++ szerű megjegyzés
    /* többsoros komment
       még egy sor megjegyzés */
    echo 'ez egy másik teszt';
    echo 'egy végső teszt'; # egy shell-szerű komment
?>
```

Az egysoros megjegyzések csak a sor végéig vagy az aktuális PHP kódblokk végéig tart attól függően, melyik van előbb. Ez azt jelenti, hogy a // ?> sorban az ez után álló HTML kód kiírásra kerül, a ?> zárójelölés visszaállítja a HTML módot, és ezt a // nem tudja befolyásolni. Amennyiben az [asp_tags](#) php.ini beállítás engedélyezve van, a fentiek az asp féle kódblokkzáró címkekre is vonatkozik.

```
<h1>Ez egy <?php # echo 'egyszerű';?> példa.</h1>
<p>A fenti cím azt írja, hogy 'Ez egy példa.'</p>
```

A 'C'-szerű kommentek az első */ karakterpárig tartanak. Vigyázz, nehogy egymásbaágazd a 'C'-szerű megjegyzéseket, amely könnyen megtörténhet, ha nagy kódrészleteket kommentezel ki.

```
<?php
/*
   echo 'Ez egy teszt'; /* Ez a megjegyzés gondot fog okozni */
*/
?>
```

11. Fejezet. Típusok

Bevezető

A PHP nyolc primitív típust támogat.

A négy skalár típus:

- [boolean](#) (logikai)
- [integer](#) (egész szám)
- [float](#) (lebegőpontos szám, más néven [double](#))
- [string](#) (karakterlánc, karaktersorozat)

A két összetett típus:

- [array](#) (tömb)
- [object](#) (objektum)

Végül két speciális típus:

- [resource](#) (erőforrás)
- [NULL](#)

Ez a kézikönyv bevezet még néhány [pszeudo-típust](#) csak az olvashatóság kedvéért:

- [mixed](#) (vegyes)

- **number** (szám)
- **callback** (visszahívható)

Találhatsz még referenciaikat a "double"-ra. Tekintsd a double-t ugyanolyannak, mint a float. A két különböző név csupán történelmi okokból létezik.

A változó típusát rendszerint nem a programozó adja meg [persze van beleszólása...], hanem a PHP futási időben határozza meg a változó környezetétől függően.

Megjegyzés: Ha egy [kifejezés](#) értékére és/vagy típusára vagy kiváncsi, akkor használ a [var_dump\(\)](#) függvényt.

Megjegyzés: Ha csak a típusára van szükség könnyen olvasható formában, akkor a [gettype\(\)](#)-ot kell alkalmazni. Típusellenőrzésre viszont *ne* ezt, hanem az [is_type](#) függvényeket kell használni a programokban. Néhány példa:

```
<?php
$bool = TRUE;      // boolean
$str  = "ize";    // string
$int  = 12;        // integer

echo gettype($bool); // kiírja, hogy "boolean"
echo gettype($str); // kiírja, hogy "string"

// Ha ez integer, megnöveljük négygyel
if (is_int($int)) {
    $int += 4;
}

// Ha a $bool változó string típusú, kiírjuk
// (ez nem fog kiírni semmit)
if (is_string($bool)) {
    echo "String: $bool";
}
?>
```

Ha egy változó típusát egy adott típusra kell konvertálnunk, [castolhatjuk](#) a változót, vagy alkalmazzuk rá a [settype\(\)](#) függvényt.

A változó különbözőképp értékelődhet ki bizonyos helyzetekben, attól függően, hogy az adott pillanatban milyen típusú. Bővebb leírásért lásd még a [Bűvészkedés a típusokkal](#) című részt! Érdekesek lehetnek még számodra a [típus összehasonlító táblázatok](#), ezek példát adnak különféle típus összehasonlításra.

Logikai adattípus

Ez a legegyszerűbb típus. Egy [boolean](#) igazságértéket fejez ki. Lehet vagy **TRUE** (igaz), vagy **FALSE** (hamis).

Megjegyzés: A logikai adattípus a PHP 4-esben került bevezetésre.

Szintaxis

Egy logikai érték megadásához használhatod a **TRUE** vagy **FALSE** szavakat, szükség szerint. Egyik jelentése sem függ a kis- és nagybetűs írásmódtól.

```
<?php  
$igaz = True; // a logikai igaz értéket adjuk az $igaz változónak  
?>
```

Tipikus valamelyen [operátor](#) használatakor kapsz **boolean** típusú értéket, amit egy [vezérlési szerkezetben](#) fel tudsz használni.

```
<?php  
// A == operátor egyenlőséget vizsgál  
// majd boolean-t ad vissza  
if ($akcio == "verzio_kirasa") { // a == operátor boolean értékkel tér vissza  
    echo "Ez az 1.23-as változat";  
}  
  
// ez nem szükséges...  
if ($elvalaszto_kirasa == TRUE) {  
    echo "<hr>\n";  
}  
  
// ...mivel egyszerűen ez is működik  
if ($elvalaszto_kirasa) {  
    echo "<hr>\n";  
}  
?>
```

Logikai értékké alakítás

Ha kifejezetten **boolean** típusúvá szeretnél alakítani egy értéket, használd a (*bool*) vagy a (*boolean*) típusátalakítást. A legtöbb esetben azonban nem kell ezt alkalmaznod, mivel az érték automatikusan átalakul, ha egy operátor, függvény, vagy vezérlési szerkezet **boolean** típusú argumentumot vár.

Lásd még a [Búvészkedés a típusokkal](#) című részt.

Amikor **boolean** típusúvá kell alakítani egy értéket, az alábbiak **FALSE** értéket adnak:

- a [boolean](#) típusú **FALSE**
- az [integer \(egész\)](#) típusú 0 (nulla)
- a [float \(lebegőpontos\)](#) 0.0 (nulla)
- az üres [string](#), és a "0" [string](#)
- egy elemeket nem tartalmazó [array \(tömb\)](#)
- egy attribútumokat nem tartalmazó [object \(objektum\)](#) (csak PHP 4)
- a speciális [NULL](#) érték (beleétve a nem beállított - hivatkozást megelőzően nem definiált - változókat)

Minden más érték **TRUE** lesz (beleértve bármilyen [resource \(eőforrás\)](#) típusú értéket).

Figyelem

A -1 is **TRUE** lesz, mint minden más nem nulla (akár negatív, akár pozitív) szám!

```
<?php  
var_dump((bool) "");           // bool(false)  
var_dump((bool) 1);            // bool(true)  
var_dump((bool) -2);           // bool(true)  
var_dump((bool) "ize");        // bool(true)  
var_dump((bool) 2.3e5);         // bool(true)
```

```
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");  // bool(true)
?>
```

Egész számok

Egy [integer](#) a következő halmaz része: $Z = \{\dots, -2, -1, 0, 1, 2, \dots\}$.

Lásd még a [Tetszőleges pontosságú egészek](#), [Lebegőpontos számok](#) és [BCMath tetszőleges pontosságú matematikai függvények](#) című részeket.

Szintaxis

Az egészek megadhatók decimális (10 alapú), hexadecimális (16 alapú) vagy oktális (8 alapú) számként, opcionális előjellel (- vagy +).

Ha az oktális formát választod, a számot egy 0 (nulla) jeggyel kell kezdened, ha a hexadecimálisat, akkor 0x-el.

Példa 11-1. Egész értékek

```
<?php
$a = 1234; # decimális szám
$a = -123; # negatív szám
$a = 0123; # oktális szám (megegyezik a 83 decimális számmal)
$a = 0xA; # hexadecimális szám (megegyezik a 26 decimális számmal)
?>
```

Az egész literálok lehetséges alakja formálisan a következő:

```
decimal      : [1-9][0-9]*
               | 0

hexadecimal : 0[xx][0-9a-fA-F]+

octal        : 0[0-7]+

integer      : [+]?decimal
               | [+]?hexadecimal
               | [+]?octal
```

Az egészek maximális mérete operációs rendszertől függ, az átlagos érték a két milliárd (32 bites előjeles egész). A PHP nem támogatja az előjelnélküli egészeket.

Figyelem

Ha egy érvénytelen számjegy szerepel egy oktális egészben, a szám hátralevő része figyelmen kívül lesz hagyva.

Példa 11-2. Az oktális számok furcsasága

```
<?php
var_dump(01090); // 010 octal = 8 decimal
?>
```

Egészek értelmezési határának túllépése

Az [integer](#) típus értelmezési tartományán kívül eső egész értékek [float](#) típussá alakulnak. Ha

valamely művelet eredménye kívül esik a [integer](#) értelmezési tartományán, akkor az eredmény automatikusan [float](#) típusúvá konvertálódik.

```
<?php
$nagy_szam = 2147483647;
var_dump($nagy_szam);
// kimenete: int(2147483647)
$nagyobb_szam = 2147483648;
var_dump($nagyobb_szam);
// kimenete: float(2147483648)

// ez nem működik hexadecimálisan megadott egészekre:
var_dump( 0x100000000 );
// kimenete: int(2147483647)

$millio = 1000000;
$nagy_szam = 50000 * $millio;
var_dump($nagy_szam);
// kimenete: float(50000000000)
?>
```

Figyelem

Sajnálatosan meg kell említenünk, hogy a PHP 4.0.6-ban ez az átalakítás nem működött minden pontosan negatív számok használatakor, például: ha a `-50000 * $millio` műveletet eredménye: `-429496728`. Ha minden operandus pozitív, nincs semmi probléma.

Ezt a hiba ki lett javítva a PHP 4.1.0-ben.

PHP-ben nincs egész osztás. Az `1/2` művelet a `0.5` float (lebegőpontos) értéket eredményezi. Alkalmazhatsz típuskonverziót integer-re lefele kerekítéshez, vagy használhatod a [round\(\)](#) függvényt.

```
<?php
var_dump(25/7);           // float(3.5714285714286)
var_dump((int)(25/7));    // int(3)
var_dump(round(25/7));    // float(4)
?>
```

Egész értékké alakítás

Ha kifejezetten [integer](#) típusúvá szeretnél alakítani egy értéket, használd az `(int)` vagy az `(integer)` típusátalakítást. A legtöbb esetben azonban nem kell ezt alkalmaznod, mivel az érték automatikusan átalakul, ha egy operátor, függvény, vagy vezérlési szerkezet [integer](#) típusú argumentumot vár. Az [intval\(\)](#) függvénytel is alakíthat sz integer-ré.

Lásd még a [Búvészkedés a típusokkal](#) című részt.

Átalakítás [boolean \(logikai\)](#) értékekről

A **FALSE** (hamis) érték a `0` (nulla) egész számmá alakul, a **TRUE** (igaz) érték az `1` (egy) egész számot adja.

Átalakítás lebegőpontos értékekről

Lebegőpontos számok egész értékké alakításakor a szám tört része elvész, azaz *lefelé kerekítés*

történik.

Ha a lebegőpontos szám az egész tartományon kívül esik (általában $+/- 2.15e+9 = 2^{31}$), az eredmény nem definiált, mivel a lebegőpontos szám nem rendelkezik elég precizitással, hogy pontos egész eredményt kapj. Sem warning, sem notice szintű hiba nem lép fel ebben az esetben!

Figyelem

Soha ne alakíts egy ismeretlen törtszámot **integer** típusúvá, mivel ez időnként nem várt eredményekhez vezethet.

```
<?php  
echo (int) ( (0.1+0.7) * 10 ); // kiírja, hogy 7!  
?>
```

Lásd a [a lebegőpontos számok pontosságának problémái](#) című részt.

Átalakítás karakterláncokról

Lásd a [Stringek átalakítása számmá](#) című részt.

Átalakítás más típusokról

Figyelem

Az egésszé alakítás viselkedése más típusokra nem definiált. Jelenleg ezek az átalakítások megegyeznek azzal, mintha először **logikai**, majd utána egész értékké alakítottad volna a kiindulási értéket. Erre a viselkedésre azonban *nem* szabad építeni, mivel minden figyelmeztetés nélkül megváltozhat.

Lebegőpontos számok

A lebegőpontos számok ("float", "valós szám") az alábbi szintaxisok bármelyikével hozhatóak létre:

```
<?php  
$a = 1.234;  
$b = 1.2e3;  
$c = 7E-10;  
?>
```

Formálisan:

```
LNUM          [0-9]+  
DNUM          ([0-9]*[\.]{LNUM}) | ({LNUM}{\.}[0-9]*)  
EXPONENT_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

A lebegőpontos számok mérete platformfüggő, de a maximális érték körülbelül 1.8e308, durván 14 tizedesjegy pontossággal (64 bites IEEE formátum).

Lebegőpontos számok pontossága

Elég gyakran előfordul, hogy egyszerű decimális törtek, mint a 0.1 és a 0.7 nem konvertálhatóak pontosan bináris megfelelőikre kis veszeség nélkül. Ez zavaró eredményekhez vezethet, például $\text{floor}(0.1+0.7)*10$ tipikusan 7 értékkel tér vissza, az elvárt 8 helyett, a belső ábrázolás miatt, ami valójában 7.999999999....

Ez azzal a tényel van kapcsolatban, hogy lehetetlen megjeleníteni néhány törtet véges számú

decimálisan számjeggyel. Például `1/3` decimálisan `0.3333333...` értékké válik.

Ezért soha ne bízz meg a törtszámok eredményeiben az utolsó jegyig, és soha se hasoníts össze két lebegőpontos számot pontos egyenlőségre. Ha nagyobb pontosságú számokra van szükséges, használ a [tetszőleges pontosságú matematikai függvényeket](#) vagy a [gmp](#) függvényeket.

Konverzió float-ra

A stringek float-ra történő konvertálásáról a [Stringek átalakítása számmá](#) című részben találsz információt. Más típusú értékek esetén a konverzió pontosan úgy történik, mintha előbb integer-re konvertálódna, majd az eredmény float-ra. További információért lásd még a [Egész értékké alakítás](#) című részt. A PHP 5-ös verziójától ha objektumot float-tá alakítasz, figyelmeztetés (notice) szintű hibajelzés keletkezik.

Stringek

A [string](#) karakterek sorozata. PHP-ben egy karakter pontosan egy bytenak felel meg, így 256 különböző karakter lehetséges. Ez azt is jelenti, hogy a PHP-nek jelenleg nincs beépített Unicode támogatása. Néhány Unicode támogatásért lásd a [utf8_encode\(\)](#) és a [utf8_decode\(\)](#) függvényeket.

Megjegyzés: Nem okozhat problémát a stringek körében, hogy túl hosszúvá válnak. Semmiféle korlát nem létezik a PHP által kezelt stringek méretére, ezért nem kell tartani a hosszú stringektől.

Szintaxis

A stringeket háromféleképpen lehet létrehozni.

- [aposztróffal](#)
- [idézőjellel](#)
- [heredoc szintaxissal](#)

String létrehozása aposztróffal

A legkönnyebben úgy adhatunk meg egy egyszerű stringet, hogy aposztrófok (' karakterek) közé tesszük.

Ha a stringben egy aposztófot szeretnél elhelyezni, és azzal nem a string végét szeretnéd jelezni, más nyelvekhez hasonlóan egy visszaperjel karaktert kell alkalmaznod az aposztóf előtt (!). Ha egy aposztróf, vagy a string vége előtt el szeretnél helyezni egy visszaperjelet, meg kell dupláznod azt. Figyelj arra, hogy ha bármilyen más karakter előtt teszel visszaperjelet, a visszaperjel meg fog jelenni a stringben. Emiatt gyakran nincs szükség magának a visszaperjelnek a duplázására.

Megjegyzés: PHP 3 használatakor azonban egy `E_NOTICE` szintű figyelmeztést kapsz, ha ezt kihasználod.

Megjegyzés: A többi móddal ellentétben, az aposztrófos stringben a [változók](#) értékei és az escape szekvenciák *nem* helyettesítődnek be.

```
<?php
echo 'Ez itt egy egyszerű string';

echo 'A stringekbe újsor karaktereket is építhetsz,
ilyen formában. Ez így szépen
működik.';

// Kimenet: Arnold egyszer azt mondta: "I'll be back"
echo 'Arnold egyszer azt mondta: "I\'ll be back"';

// Kimenet: You deleted C:\*.*?
echo 'You deleted C:\\*.*?';

// Kimenet: You deleted C:\*.*?
echo 'You deleted C:\*.*?';

// Kimenet: Nem helyettesítődik be \n az újsor karakter
echo 'Nem helyettesítődik be \n az újsor karakter';

// Kimenet: A $változók $szintén nem helyettesítődnek be
echo 'A $változók $szintén nem helyettesítődnek be ';
?>
```

String létrehozása idézőjellel

Ha egy stringet idézőjelek ("") közé helyezünk, a PHP több speciális jelölés feldolgozására lesz képes:

Táblázat 11-1. Speciális jelölések idézőjeles stringben

jelölés	jelentése
\n	újsor (LF vagy 0x0A (10) ASCII kódú karakter)
\r	kocsivissza (CR vagy 0x0D (13) ASCII kódú karakter)
\t	vízszintes tabulátor (HT vagy 0x09 (9) ASCII kódú karakter)
\	visszaperjel
\\$	dollárjel
\"	idézőjel
\[0-7]{1,3}	egy karaktersorozat, ami oktális számokra illeszkedik
\x[0-9A-Fa-f]{1,2}	egy karaktersorozat, ami hexadecimális számokra illeszkedik

Ha bármilyen más karakter elé visszaperjelet írsz, ebben az esetben is ki fog írórni a visszaperjel.

A legfontosabb jellemzője az idézőjeles stringeknek az, hogy a változók behelyettesítésre kerülnek. Lásd a [változók behelyettesítése](#) című részt további részletekért.

String létrehozása heredoc szintaxis

Egy másfajta módszer a stringek megadására a heredoc szintaxis ("`<<<`"). [ez itt megint nem elírás, kedves unix-shell programozó!] A `<<<` jelzés után egy azonosítót kell megadni, majd a stringet, és végül az azonosítót mégegyszer, ezzel zárva le a stringet.

A lezáró azonosítónak mindenféle sor legelső karakterén kell kezdődni. Ugyancsak figyelni

kell arra, hogy ez az azonosító is az általános PHP elemek elnevezési korlátai alá tartozik: csak alfanumerikus karaktereket és aláhúzást tartalmazhat, és nem kezdődhet számjegy karakterrel.

Figyelem

Nagyon fontos, hogy odafigyelj arra, hogy a lezáró azonosítót tartalmazó sor ne tartalmazzon semmi mást, csupán *esetleg* egy pontosvessző (:) karaktert. Ez még pontosabban azt is jelenti, hogy az azonosító *nem lehet beljebb kezdve*, és nem szabad semmilyen szóköz vagy tabulátor karaktert sem elhelyezni a pontosvessző előtt vagy után. Fontos, hogy az záróazonosító előtt szerepelnie kell egy újsor karakternek az operációs rendszernek megfelelően. Ez például Macintosh-on \r.

Ha ezt a szabály megsérte és a záróazonosító nem "tiszta", akkor nem lesz záróazonosítónak tekitve, így a PHP tovább fogja keresni. Ebben az esetben, ha nem talál érvényes záróazonosítót, akkor szintaktikai hibát fog adni, ahol a hiba sorának száma a szkript végére fog mutatni.

Nem megengedett a heredoc szintaxis használata az osztályok attribútumainak inicializálásakor. Ilyen esetben más string-szintaxist kell használnod.

Példa 11-3. Helytelen példa

```
<?php
class ize {
    public $bigyo = <<<EOT
bigyo
EOT;
}
?>
```

A heredoc az idézőjeles stringekhez hasonlóan működik, az idézőjelek nélkül. Ez azt jelenti, hogy nem kell visszaperjellel jelülni az idézőjeleket a szövegben, de a fenti speciális jelölések használhatóak. A változók értékei behelyettesítődnek, de az idézőjeles karaktersorozatokhoz hasonlóan gondosan kell ügyelni a komplex változó hivatkozások megadására.

Példa 11-4. "Heredoc" string példa [VAS: Vége A Stringnek de helyette lehet bármilyen szöveg]

```
<?php
$str = <<<VAS
Példa egy stringre, amely
több sorban van, és heredoc
szintaxisú
VAS;

/* Komplexebb példa, változókkal. */
class ize
{
    var $ize;
    var $valami;

    function ize()
    {
        $this->ize = 'Valami';
        $this->valami = array('elso', 'masodik', 'harmadik');
    }
}

$size = new ize();
$nev = 'Béla';

echo <<<VAS
```

```
A nevem "$nev". Kiírok egy értéket: $size->ize.  
Most kiírok egy tömbelemet: {$size->valami[1]}.  
EZ nagy 'A' kell, hogy legyen: \x41  
VAS;  
?>
```

Megjegyzés: A heredoc a PHP 4-esben került a nyelvbe.

Változók behelyettesítése

Ha egy stringet idézőjelek között, vagy heredoc szintaxissal adsz meg, a jelölt **változók** értékei behelyettesítésre kerülnek.

Kétféleképpen lehet megadni egy változót: az [egyszerű](#) és a [komplex](#) formátummal. Az egyszerű forma a leggyakoribb és legkényelmesebb. Lehetőséget ad egy skalár, tömb érték vagy egy objektum tulajdonság beillesztésére.

A komplex szintaxis a PHP 4-es változatában került a nyelvbe, és a jelölésben használatos kapcsos zárójelekkel ismerhető fel.

Egyszerű szintaxis

Ha dollár (\$) jelet talál a PHP egy stringben, mohón megkeresi az összes ezt követő azonosítóban is használható karaktert, hogy egy érvényes változónevet alkosszon. Használj kapcsos zárójeleket, ha pontosan meg szeretnéd határozni, meddig tart egy változó.

```
<?php  
$ingatlan = 'ház';  
echo "kertes $ingatlan kerítéssel"; // működik, szóköz nem lehet változónévben  
echo "páros $ingatlanszám"; // nem működik, az 's' és további karakterek lehetnek változók  
echo "páros ${ingatlan}szám"; // működik, mert pontosan meg van adva a név  
echo "páros {$ingatlan}szám"; // működik, mert pontosan meg van adva a név  
?>
```

Hasonlóképpen meg lehet adni tömbindexet vagy objektum tulajdonságot is. A tömbindexek esetében a záró szögletes zárójel (/) jelöli az index végét, az objektum tulajdonságoknál az egyszerű skalárok szabályai érvényesek, habár objektum tulajdonágok esetén nem használható a fenti trükk.

```
<?php  
// Ezek a példák a tömbök stringen belüli használatára vonatkoznak.  
// Stringen kívül minden indexet idézőjelbe a tömb string típusú indexeit  
// és ne használj {kapcsos zárójeleket} stringen kívül.  
  
// Irassunk ki minden hibát  
error_reporting(E_ALL);  
  
$gyumolcsok = array('eper' => 'piros', 'alma' => 'zöld');  
  
// Működik, de láss az $size[valami] karakterláncon kívüli problémáját  
echo "Az alma $gyumolcsok[alma]."; // ez másképpen használandó karaktersorozatokon kívül.  
  
// Működik  
echo "Az alma {$gyumolcsok['alma']}.";  
  
// Működik, de a PHP egy alma nevű konstanst keres először  
// mint ahogy alább le van írva  
echo "A alma is {$gyumolcsok[alma]}.";
```

```
// Nem működik, kapcsos zárójeleket kellene használni. Szintaktikai hibát epirosményez.  
echo "A alma is $gyumolcsok['alma']."'.";  
  
// Működik  
echo "A alma is " . $gyumolcsok['alma'] . "..";  
  
// Működik  
echo "A négyzet $negyzet->szelesseg méter széles.";  
  
echo "A négyzet $negyzet->szelesseg00 centiméter széles."; // nem működik  
// A megoldás érdekében lásd a komplex szintaxis szakaszt!  
?>
```

Bármely ennél komplexebb helyettesítéshez a komplex szintaxis használatos.

Komplex (kapcsos zárójeles) szintaxis

Ezt nem azért nevezzük komplexnek, mert a szintaxis komplex, hanem azért, mert így komplex kifejezések helyettesítésére nyílik lehetőség.

Gyakorltlag bármilyen változó érték behelyettesíthető ezzel a szintaxissal. Egyszerűen úgy kell megadni az értéket adó kifejezést, mintha a stringen kívül dolgoznál vele, és utána { és } jelek közé kell zárni. Mivel a '{' jel megadására nincs speciális jelölés, ez a forma csak akkor fog működni, ha a { után közvetlenül egy \$ jel található. (Használhatod a "{\$" vagy "\{\$" jelöléseket, ha a stringben a "{\$" sorozatot szeretnéd beilleszteni, és nem változóhelyettesítést adsz meg). Néhány példa, ami tisztázhatja a félreérteket:

```
<?php  
// Irassunk ki minden hibát  
error_reporting(E_ALL);  
  
$oriasi = 'fantasztikus';  
  
// Nem működik, kimenet: Ez { fantasztikus}  
echo "Ez { $oriasi}";  
  
// Működik, kimenet: Ez fantasztikus  
echo "Ez {$oriasi}";  
echo "Ez ${oriasi}";  
  
// Működik  
echo "Ez a négyzet {$negyzet->szelesseg}00 cm széles.";  
  
// Működik  
echo "Ez működik: {$arr[4][3]}";  
  
// Ez hibás ugyanazért, amiért $size[bigyo] hibás stringen  
// kívül. Más szóval, működik, de mivel a PHP az ize nevű  
// konstanst keresi először, E_NOTICE szintű hibát  
// fog adni (undefined constant).  
echo "Ez wrong: {$arr[ize][3]}";  
  
// Működik. Amikor többdimenziós tömböket használysz, stringen  
// belül mindenig vedd körük kapcsos zárójelekkel  
echo "Ez működik: {$arr['ize'][3]}";  
  
// Működik.  
echo "Ez működik: " . $arr['ize'][3];
```

```
echo "Még így is írhatod: {$obj->values[3]->nev}";  
echo "Ez a $nev nevű változó értéke: {${$nev}}";  
?>
```

String karaktereinek elérése és módosítása

A string karaktereire nullától számított indexekkel lehet hivatkozni, a string neve után megadott kapcsos zárójelek között.

Megjegyzés: Kompatibilitási okokból a tömbökönél használatos szögletes zárójelek is alkalmazhatóak a karakterek eléréséhez. Ez a tömb jelzés azonban nem javasolt a PHP 4-től.

Példa 11-5. Néhány string példa

```
<?php  
$str = 'Ez egy teszt.';  
$first = $str{0};  
  
// Most meg a harmadik  
$harmadik = $str{2};  
  
// Az utolsó karakter kiolvasása  
$str = 'Ez még mindig egy teszt.';  
$utolso = $str[strlen($str)-1];  
  
// Utolsó karakter módosítása  
$str = 'Értem';  
$str[strlen($str)-1] = 'd';  
?>
```

Hasznos függvények és operátorok

Stringeket a '.' (pont) stringösszefűző operátorral tudsz összefűzni. A '+' (összeadás) operátor nem alkalmas erre a feladatra. Lásd a [String operátorok](#) című részt további információkért.

Számos hasznos függvény létezik stringek manipulálásához.

Lásd a [string függvények](#) című részt általános függvényekért, a reguláris kifejezés függvényeket fejlett kereséshez és cserékhöz (két formában: [Perl](#) és [POSIX kiterjesztett](#)).

Külön függvények léteznek [URL stringekhez](#), és stringek kódolásához/dekódolásához ([mcrypt](#) és [mhash](#) kódolások).

Végül ha ezek között sem találod, amit keresel, lásd a [karakter típus függvényeket](#).

Konverzió stringgé

Stringgé konvertálni a *(string)* módosítóval vagy a [strval\(\)](#) függvénytel tudsz. A string konverzió automatikusan végrehajtódik olyan környezetben, ahol string értékre van szükség. Ez [echo\(\)](#) és [print\(\)](#) függvények használatánál megtörténik, vagy olyankor, amikor változó értékét stringgel hasonlítod össze. A [Típusok](#) és [Bűvész kedés a típusokkal](#) részek elolvasása segíthet a következők megértésében. Lásd még a [settype\(\)](#) függvényt.

A [boolean](#) **TRUE** érték az "1" stringgé, a **FALSE** pedig a "" (üres string) stringgé konvertálódik. Ily

módon alakítgathatsz oda-vissza a logikai és szöveges adatok között.

Egy egész ([integer](#)) vagy egy lebegőpontos szám ([float](#)) úgy konvertálónak, hogy az eredményül kapott string a szám számjegyeiből áll (beleértve az exponenciális részt lebegőpontos számok esetén).

A tömbök mindig az "Array" stringgé konvertálónak, tehát így nem tudod kiiratni egy tömb tartalmát [echo\(\)](#) vagy [print\(\)](#) függvényel. Egy elem kiírásához használj valami ehhez hasonlót: `echo $arr['ize']`. Lásd lejjebb a teljes tartalom kiírására vonatkozó javaslatokat.

A objektumok mindig az "Object" stringgé konvertálónak. Ha egy attribútumának az értékét szeretnéd kiírni, olvasd el a lenti bekezdéseket. Ha azt szeretnéd kideríteni, hogy egy objektum melyik osztálynak a példánya, használ a [get_class\(\)](#) függvényt. A PHP 5-ös verziójától a konverzió az objektum [__toString\(\)](#) metódusának meghívásával történik.

Az erőforrások ([resource](#)) minden "Resource id #1" formájú strigekké alakítódnak át, ahol *I* az erőforrás egyedi azonosító száma, amelyet a PHP oszt ki futásidőben. Ha az erőforrás típusát szeretnéd megtudni, használ a [get_resource_type\(\)](#) függvényt.

A [NULL](#) minden az üres stringgé konvertálódik.

Amint fent olvashattad, tömbök, objektumok és erőforrások kiírása nem szolgáltat semmi hasznos információt magukról az értékekéről. A hibakezelés (debug) esetén használható értékek kiírására egy jobb módszer megismeréséhez tanulmányozd a [print_r\(\)](#) és a [var_dump\(\)](#) függvényeket.

A PHP értékeket alakíthatod úgy is stringgé, hogy örökre el tudsz tárolni azokat. Ezt a módszert szerializációnak nevezik, és a [serialize\(\)](#) függvény alkalmazható erre. A PHP értékeket XML struktúrába is tudsz szerializálni, abban az esetben, ha van a PHP konfigurációban [WDDX](#).

String konvertálása számmá

Amikor egy string számként értékelődik ki, az eredmény típusa a következőképp kerül meghatározásra.

A string lebegőpontosként értékelődik ki, ha a '.', 'e', vagy 'E' karakterek bármelyikét tartalmazza. Egyébként egésként értékelődik ki.

Az érték meghatározása a string első karakterei alapján történik. Ha a string érvényes számmal kezdődik, ez az érték kerül felhasználásra. Máskülönben az érték 0 lesz [nulla, nem nagy O betű]. Érvényes számnak tekintendő, amelynek opcionális előjelét egy vagy több számjegy követi (melyben lehet egy tizedespont), végül lehet exponense. Az exponens egy 'e' vagy 'E' karakter, melyet egy vagy több szám követ.

```
<?php
$size = 1 + "10.5";                                // $size most float (11.5)
$size = 1 + "-1.3e3";                               // $size most float (-1299)
$size = 1 + "bob-1.3e3";                            // $size most integer (1)
$size = 1 + "bob3";                                 // $size most integer (1)
$size = 1 + "10 Kicsi Pingvin";                   // $size most integer (11)
$size = 4 + "10.2 Kicsi Pingvinke" // $size most integer (14)
$size = "10.0 disznó " + 1;                         // $size most integer (11)
$size = "10.0 disznó " + 0.5;                      // $size most float (10.5) [szóval 10 disznó, meg egy fé?
?>
```

Ha többet szeretnél megtudni erről a konverzióról, akkor nézd meg a Unix manuált `strtod(3)`. [=RTFM]

Ha szeretnéd kipróbálni valamelyik fejezetbeli példát, elmásolhatod innen a példákat és beszúrhatsz az alábbi sort, hogy lásd mi is történik.

[Ahhoz nem árt azért előbb érteni a beszúrt sort. A beszúrandó string elején a "\" azért van, hogy a \$ize helyére NE a ize változó értéke kerüljön, hanem a \\$ hatására a \$ karakter íródjon ki, majd a ize karaktertörlő, egy egyenlőségjel, és utána a \$ize változó értéke. Az első "\" hatására ugyanis a köv. karakter ("\$") elveszti különleges jelentését, és kimásolódik a kimenetre, majd jön a "ize=" szöveg, ez is úgy, ahogy van a kimenetre kerül, a fordító nem is találkozott változóval, amit be kellene helyettesíteni. Aztán jön a \$ize, ezt már felismeri a fordító, és a változó értékét írja ki]:

```
<?php  
echo "\$size==$size; típusa " . gettype($size) . "<br />\n";  
?>
```

Ne reméld, hogy egy karaker kódját megkapod, ha egésszé konvertálod (mint ahogyan például C-ben tennéd). Használ a [ord\(\)](#) és a [chr\(\)](#) függvényeket a karakterkódok és a karakterek közötti konverzióra.

Tömbök

A PHP tömbjei rendezett leképezéseket valósítanak meg. A leképezés értékeit rendel *kulcsokhoz*. Ez a típus sokféleképpen használható, mint egy hagyományos tömb, egy lista (vektor), hash tábla, szótár, kollekció, halmaz, sor, és mások. Mivel egy újabb PHP tömb szerepelhet értékként, könnyen szimulálhatsz fákat.

Az említett struktúrák leírása ezen kézikönyv kereteibe nem fér bele, de legalább egy példát mutatunk mindegyikre. További információkért külső forrásokat kell igénybe venned erről az igen széles témáról.

Szintaxis

Tömb létrehozása az [array\(\)](#) nyelvi elemmel

Egy [array](#) (tömb) típusú változót az [array\(\)](#) nyelvi elemmel tudsz létrehozni, amely számos vesszővel elválasztott *kulcs* => érték párt vár.

```
array( [kulcs =>] érték  
      , ...  
      )  
// a kulcs lehet integer vagy string  
// az érték bármilyen érték lehet  
  
<?php  
$arr = array("ize" => "bigyo", 12 => true);  
  
echo $arr["ize"]; // bigyo  
echo $arr[12]; // 1  
?>
```

A *kulcs* lehet [integer](#) vagy [string](#) típusú. Ha a kulcs egy [integer](#) szabályos reprezentációja, akkor egészként értelmezi. (azaz a "8"-at 8-ként értelmezi, a "08"-at pedig "08"-nak). Ha a *kulcs*-nak valós értéket adsz meg, akkor azt egésszé ([integer](#)) alakítja. PHP-ben nincs külön indexelt és asszociatív tömb típus, csak egyfélé tömb típus van, amely tartalmazhat mind egész mind string indexeket.

Az érték bármilyen típusú lehet.

```
<?php  
$arr = array("tombocske" => array(6 => 5, 13 => 9, "a" => 42));
```

```
echo $arr["tombocske"][6];      // 5
echo $arr["tombocske"][13];     // 9
echo $arr["tombocske"]["a"];    // 42
?>
```

Ha nem adsz meg kulcsot egy adott értékhez, akkor annak a kulcsa az egész típusú indexek maximuma + 1 lesz. Ha olyan kulcsot adsz meg, amelyhez már létezik hozzárendelt érték, az érték felül lesz írva.

```
<?php
// Ez a tömb ugyanaz mint...
array(5 => 43, 32, 56, "b" => 12);

// ...ez a tömb
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

Figyelem

Az indexgenerálás fent leírt algoritmus a PHP 4.3.0-tól megváltozott. Ha egy olyan tömbhöz adsz hozzá index nélküli elemet, amelyben a legnagyobb index értéke nagatív, az új elem indexe nulla (0) lesz. Azelőtt az új index a legnagyobb index + 1 lett úgyanúgy mint pozitív indexek esetén.

Ha **TRUE**-t adsz meg kulcsként az 1 egész típusú értéket fogja jelenteni, a **FALSE** pedig a 0-át. Ha kulcsként **NULL**-t adsz meg, üres stringként lesz értelmezve. Az üres string használata kulcsként létrehoz (vagy felülír) egy üres string kulcsot és az értéket; eltérően az üres szögletes zárójelektől.

Nem használhatsz tömböt vagy objektumot kulcsként. Ha mégis megpróbáld, egy *Illegal offset type* figyelmeztetést fogsz kapni.

Létrehozás/módosítás a szögletes zárójeles formával

Meglévő tömbök is módosíthatóak konkrét értékek megadásával.

Ezt úgy tudod megtenni, hogy a tömb neve után szögletes zárójelekben megadod a kulcsot, amit módosítani szeretnél. Ha elhagyod a kulcsot, és csak egy üres szögletes zárójel párt ("[]") adsz meg a változó neve után, a tömb végére illeszhetsz elemet.

```
$tomb[kulcs] = érték;
$tomb[] = érték;
// a kulcs lehet integer (egész szám) vagy string
// az érték bármi lehet
```

Ha a **\$tomb** nem létezik, ezzel létrejön. Tehát ez egy újabb alternatíva tömbök létrehozására. Ha módosítani szeretnél egy elemet, rendelj hozzá egy kulccsal azonosított elemhez egy új értéket. Ha meg szeretnél szüntetni egy kulcs/érték párt, használd az **unset()** függvényt.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;      // itt ez ugyanaz mint a $arr[13] = 56;

$arr["x"] = 42;  // Ez hozzáad egy új elemet
                 // a tömbhöz "x" kulccsal

unset($arr[5]); // Ez eltávolítja az elemet a tömbből

unset($arr);     // Ez az egész tömböt törli
?>
```

Megjegyzés: Mint ahogy fentebb említettük, ha a szögeletes zárójelek között nem adsz meg kulcsot, akkor a létező indexek maxumuma + 1 lesz az új elem kulcsa. Ha még nincs egész index, akkor ez 0 (nulla) lesz. Ha olyan kulcsot adsz meg, amelyhez már van érték rendelve, az érték felül lesz írva.

Figyelem

Az indexgenerálás fent leírt algoritmusá PHP 4.3.0-tól megváltozott. Ha egy olyan tömbhöz adsz hozzá index nélküli elemet, amelyben a legnagyobb index értéke nagatív, az új elem indexe nulla (0) lesz. Azelőtt az új index a legnagyobb index + 1 lett úgyanúgy mint pozitív indexek esetén.

Az automatikus indexeléshez használд legnagyobb szám *nem kell abban a pillanatban létezzen a tömbben*. Egyszerűen csak kellett valamikor létezett a tömbben a legutóbbi újraindexelés óta. A következő példa szemlélteti ezt:

```
<?php
// Létrehozunk egy egyszerű tömböt
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Most kitörlünk minden elemet, de magát a tömböt nem
foreach ($array as $i => $ertek) {
    unset($array[$i]);
}
print_r($array);

// Hozzáadunk egy elemet (az új kulcs 5 lesz az elvárt 0 helyett)
$array[] = 6;
print_r($array);

// Újraindexeljük:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

A fenti példa a következő kimenetet adja:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)
```

Hasznos függvények

Létezik néhány függvény a tömbökkel való munka megkönnyítésére. Lásd a [tömb függvények](#) című részt.

Megjegyzés: Az [unset\(\)](#) függvénnnyel lehet egy értéket a tömbből törölni. Figyelj arra, hogy a tömb nem lesz újraindexelve! Ha csak a "szokásos egész indexet" használod (nullától kezdve egyessével növelve), az újraindexelés hatását a [array_values\(\)](#) függvénnnyel érheted el.

```
<?php
$a = array(1 => 'egy', 2 => 'kettő', 3 => 'három');
unset($a[2]);
/* ennek eredményeképpen $a így fog kinézni:
   $a = array(1 => 'egy', 3 => 'három');
   és NEM így:
   $a = array(1 => 'egy', 2 => 'három');
 */

$b = array_values($a);
// Itt a $b így néz ki: array(0 => 'egy', 1 => 'három');
?>
```

A [foreach](#) vezérlési szerkezet kifejezetten a tömbök számára jött létre. Egy egyszerű módszert ad tömbökön való végiglépkedésre.

Mit tehetünk, és mit nem a tömbökkel

Miért nem jó az `$size[bigyo]` forma?

Minden string típusú kulcsot idézőjelek vagy aposztrófok közé kell tenni tömbbeli elemre történő hivatkozásnál, például: `$size['bigyo']` és nem `$size[bigyo]`. Vajon miért nem jó `$size[bigyo]` alak? Talán láttad ezt a szintaxist régi PHP programokban:

```
<?php
$size[bigyo] = 'ellenség';
echo $size[bigyo];
// stb.
?>
```

Ez hibás, és mégis működik. Akkor mégis miért nem jó? Azért, mert a `bigyo` itt valójában egy nem definiált állandót jelöl, és nem a `'bigyo'` sztringet (figyelj az aposztrófokra). A későbbi PHP verziókban már előfordulhatnak olyan előre definiált állandók, amelyeknek ugyanaz lesz a neve, mint ami a saját kódodban sztringként szerepel. Azért működik, mert ezeket a `csupasz stringeket` (idézőjel nélküli string, amelyeknek nem felel meg egy ismert szimbólum) a PHP a nevükkel azonos tartalmú sztringgé konvertálja. Például nincs definiálva `bigyo` nevű konstans, ezért a PHP helyettesíteni fogja a `'bigyo'` stringgel.

Megjegyzés: Ez nem azt jelenti, hogy *mindig* idézőjelek közé kellene raknod a kulcsot. Ne használj idézőjelet, amikor a kulcs [konstans](#) vagy [változó](#), mivel ez megakadályozná a PHP-t abban hogy ezeket értelmezze.

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);
```

```
// Egyszerű tömb:  
$array = array(1, 2);  
$count = count($array);  
for ($i = 0; $i < $count; $i++) {  
    echo "\n$i vizsgálata: \n";  
    echo "Rossz: " . $array['$i'] . "\n";  
    echo "Jó: " . $array[$i] . "\n";  
    echo "Rossz: {$array['$i']} \n";  
    echo "Jó: {$array[$i]} \n";  
}  
?>
```

Megjegyzés: A fenti példa a következő kimenetet adja:

```
0 vizsgálata:  
Notice: Undefined index: $i in /path/to/script.html on line 9  
Rossz:  
Jó: 1  
Notice: Undefined index: $i in /path/to/script.html on line 11  
Rossz:  
Jó: 1  
  
1 vizsgálata:  
Notice: Undefined index: $i in /path/to/script.html on line 9  
Rossz:  
Jó: 2  
Notice: Undefined index: $i in /path/to/script.html on line 11  
Rossz:  
Jó: 2
```

Több példa ennek megmutatására:

```
<?php  
// Irassunk ki minden hibát  
error_reporting(E_ALL);  
  
$arr = array('gyümölcs' => 'alma', 'zöldség' => 'répa');  
  
// Helyes  
print $arr['gyümölcs']; // alma  
print $arr['zöldség']; // répa  
  
// Helytelen. Működik de E_NOTICE szintű hibát ad,  
// mert nincs gyümölcs nevű konstans definiálva  
//  
// Notice: Use of undefined constant gyümölcs - assumed 'gyümölcs' in...  
print $arr[gyümölcs]; // alma  
  
// Definiálunk egy konstanst, hogy lássuk mi folyik itt.  
// A gyümölcs nevű konstanshoz a 'zöldség' értéket rendeljük.  
define('gyümölcs', 'zöldség');  
  
// Notice the difference now  
print $arr['gyümölcs']; // alma  
print $arr[gyümölcs]; // répa  
  
// A következő rendben van, mert string belséjében van.  
// A konstansokat nem keresi stringen belül, tehát itt nincs E_NOTICE  
print "Hello $arr[gyümölcs]"; // Hello alma  
  
// Kivétel: a stringben kapcsos zárójelekkel körülvett tömbök
```

```
// esetén figyeli a konstansokat
print "Hello {$arr['gyümölcs']}"; // Hello répa
print "Hello {$arr['gyümölcs']}"; // Hello alma

// Ez nem működik, szintaktikai hibát eredményez, mint például:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or T_NUM_STRING'
// Ez természetesen szuperglobális tömbök esetén is érvényes.
print "Hello $arr['gyümölcs']";
print "Hello $_GET['ize']";

// Egy másik lehetőség az összefűzés (konkatenáció)
print "Hello " . $arr['gyümölcs']; // Hello alma
?>
```

Ha beállítod az [error_reporting\(\)](#)-ot hogy írja ki az **E_NOTICE** szintű hibákat (vagy ha **E_ALL**-ra állítod), akkor látni fogod ezeket a hibákat. Alapértelmezésben az [error_reporting](#) úgy van beállítva, hogy ezeket ne mutassa.

Ahogy a [szintaxisról szóló részben](#) tárgyaltuk, a szögletes zárójelek között ('/ and '/') kifejezés kell álljon. Ez azt jelenti, hogy írhatsz ilyeneket is:

```
<?php
echo $arr[fuggveny($bigyo)];
?>
```

Ez a példa bemutatja, hogyan használhatsz függvény visszatérési értéket tömbindexként. A PHP úgyszintén ismeri a konstansokat. Bizonyára hallottál már az *E_** konstansokról.

```
<?php
$hiba_leiras[E_ERROR] = "Fatális hiba történt";
$hiba_leiras[E_WARNING] = "A PHP figyelmeztetést adott";
$hiba_leiras[E_NOTICE] = "Informális megjegyzés";
?>
```

Figyeld meg, hogy az *E_ERROR* egy érvényes azonosító, mint a *valami* az előző példában. De a legutóbbi példa ugyanaz, mintha ezt írnánk:

```
<?php
$hiba_leiras[1] = "Fatális hiba történt";
$hiba_leiras[2] = "A PHP figyelmeztetést adott";
$hiba_leiras[8] = "Informális megjegyzés";
?>
```

mivel az *E_ERROR* konstans értéke *1*, stb.

As we already explained in the above examples, *\$size/bigyo]* still works but is wrong. It works, because *bigyo* is due to its syntax expected to be a constant expression. However, in this case no constant with the name *bigyo* exists. PHP now assumes that you meant *bigyo* literally, as the string "*bigyo*", but that you forgot to write the quotes.

De miért nem jó ez?

Valamikor a jövőben a PHP fejlesztői hozzáadhatnak egy új konstanst vagy kulcsszót a nyelvhez, vagy bevezethetsz egy új konstanst az alkalmazásodba, és akkor bajba kerülsz. Például jelenleg sem használhatóak az *empty* és *default* szavak, mivel ezek speciális kulcsszavak.

Ha ezek az érvek nem győznek meg: ez a szintaxis egyszerűen nem javasolt, és bármikor megszünhet működni.

Megjegyzés: Mint ahogy említettük, a dupla-idézőjelekkel megadott **string**-ben helyes, ha a tömbindexet nem veszed körül idézőjelekkel, így a "*\$size/bigyo]*" helyes.

Részletekért lásd a fenti példákat, valamint a [változók behelyettesítése](#) című részt.

Konverzió tömbbé

Ha az **integer**, **float**, **string**, **boolean** vagy **resource** típusokat konvertálod tömbbé (**array**), egyelemű tömböt kapsz eredményül (indexe 0), amely az eredeti elem értékét tartalmazza.

Ha objektumot konvertálsz tömbbé, a tömb elemei az objektum attribútumai lesznek, a kulcsok pedig a megfelelő attribútumok nevei lesznek.

Ha a **NULL** értéket konvertálod tömbbé, üres tömböt kapsz eredményül.

Tömbök összehasonlítása

Lehetőség van a tömbök összehasonlítására. Erre használhatod a [array_diff\(\)](#) függvényt valamint a [tömb operátorokat](#).

Példák

A tömb típus a PHP-ben nagyon sokoldalú, ezért összegyűjtöttünk néhány példát, hogy megmutassuk a tömbök erejét.

```
<?php
// ez
$a = array( 'szín' => 'piros',
            'íz'      => 'édes',
            'alak'    => 'kerek',
            'név'     => 'alma',
                    4           // 0 lesz a kulcsa
        );

// teljesen meggyezik ezzel
$a['szín'] = 'piros';
$a['íz']   = 'édes';
$a['alak'] = 'kerek';
$a['név']  = 'alma';
$a[]       = 4;           // 0 lesz a kulcsa

$b[] = 'a';
$b[] = 'b';
$b[] = 'c';
// a következő tömböt adja: array(0 => 'a', 1 => 'b', 2 => 'c')
// vagy egyszerűen: array('a', 'b', 'c')
?>
```

Példa 11-6. Az array() használata

```
<?php
// Tömb, mint tulajdonság hozzárendelés
$map = array( 'verzió'      => 4,
              'rendszer'    => 'Linux',
              'nyelv'       => 'angol',
              'short_tags' => true
        );

// szigorúan számokat tartalmazó tömb
```

```
$tomb = array( 7,
                8,
                0,
                156,
                -10
            );
// ez ugyanaz, mint array (0 => 7, 1 => 8, ...)

$valtогatas = array(          10, // 0. indexű/kulcsú
                      5    => 6,
                      3    => 7,
                      'a'  => 4,
                           11, // kulcsa 6 (a legnagyobb egész kulcs 5 volt)
                      '8'  => 2, // kulcsa 8 (egész!)
                      '02' => 77, // kulcsa '02'
                      0    => 12 // a 10 értéket felülírjuk 12-vel
            );
// üres tömb
$ures = array();
?>
```

Példa 11-7. Kollekció

```
<?php
$szinek = array('piros', 'kék', 'zöld', 'sárga');

foreach ($szinek as $szin) {
    echo "Szereted a(z) $szin színt?\n";
}

?>
```

A fenti példa a következő kimenetet adja:

```
Szereted a(z) piros színt?
Szereted a(z) kék színt?
Szereted a(z) zöld színt?
Szereted a(z) sárga színt?
```

Figyelj arra, hogy jelenleg közvetlenül nem lehet módosítani a tömb elemeinek értékét ezzel a ciklussal. A problémát a következőképpen tudod megkerülni:

Példa 11-8. Kollekció

```
<?php
foreach ($szinek as $kulcs => $szin) {
    // nem működik (nem módosítja a tömböt):
    // $szin = strtoupper($szin);

    // működik (módosítja a tömböt):
    $szinek[$kulcs] = strtoupper($szin);
}
print_r($szinek);
?>
```

A fenti példa a következő kimenetet adja:

```
Array
(
    [0] => PIROS
    [1] => KÉK
    [2] => ZÖLD
    [3] => SÁRGA
)
```

Ebben a példában egy egytől számosztott tömböt készítünk.

Példa 11-9. Egytől kezdődő index

```
<?php  
$elsonegyed = array(1 => 'Január', 'Február', 'Március');  
print_r($elsonegyed);  
?>
```

A fenti példa a következő kimenetet adja:

```
Array  
(  
    [1] => 'Január'  
    [2] => 'Február'  
    [3] => 'Március'  
)
```

Példa 11-10. Tömb feltöltése

```
<?php  
// egy tömb felöltése a <link linkend="ref.dir">könyvtárban</link> található filenevekkel  
$konyvtar = opendir('.');  
while (false !== ($filenev = readdir($konyvtar))) {  
    $filenevek[] = $filenev;  
}  
closedir($konyvtar);  
?>
```

A tömbök rendezettek. A sorrendet számos függvénytel megváltoztathatod. Lásd a [tömb függvények](#) című részt további információkért. A tömbben szereplő elemek számát a [count\(\)](#) függvénytel tudod lekérdezni.

Példa 11-11. Tömbök rendezése

```
<?php  
sort($filenevek);  
print_r($filenevek);  
?>
```

Mivel a tömb egy értéke bármilyen lehet, értékként akár egy másik tömböt is megadhatsz. Ilyen formában készíthetsz rekurzív vagy többdimenziós tömböket.

Példa 11-12. Rekurzív és többdimenziós tömbök

```
<?php  
$gyumolcsok = array("gyümölcsök" => array("a" => "narancs",  
                                              "b" => "banán",  
                                              "c" => "alma"  
                                              ),  
      "számok"     => array (1,  
                               2,  
                               3,  
                               4,  
                               5,  
                               6  
                               ),  
      "lyukak"      => array (      "első",  
                               5 => "második",  
                               "harmadik"  
                               )  
      );  
  
// Néhány példa a fenti tömb elemeinek hivatkozására  
echo $gyumolcsok["lyukak"][5]; // kiírja, hogy "második"  
echo $gyumolcsok["gyümölcsök"]["a"]; // kiírja, hogy "narancs"  
unset($gyumolcsok["lyukak"][0]); // eltávolítja az "első"-t
```

```
// Készítünk egy új többszintű tömböt
$gyumolcsle["alma"]["zöld"] = "finom";
?>
```

Jegyezd meg, hogy tömb értékül adása minden másolást jelent. Használd a referencia operátort, ha a tömböt referencia szerint szeretnéd átadni.

```
<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // az $arr2 megváltozott,
             // az $arr1 még mindig array(2, 3)

$arr3 = &$arr1;
$arr3[] = 4; // itt az $arr1 és az $arr3 megegyeznek
?>
```

Objektumok

Objektumok létrehozása

Egy objektum létrehozására a *new* operátor való, amely az adott objektumtípus egy példányát hivatott létrehozni [mivel lehet, hogy hivatkozni akarunk rá, rendszerint másolni szoktuk az objektumot (ill. a címét) egy változóba]

```
<?php
class semmi // egy objektumosztály létrehozása, semmi az osztály neve
{
    function do_semmi ()
    {
        echo "Csinálom a semmit.";
    }
}

$bigyo = new semmi; // $bigyo most egy semmi típusú objektum
$bigyo->do_semmi(); // a $bigyo objektum do_semmi()
                      // metódusát (függvényét) hívja
?>
```

Alaposabb információkért nézd meg az [Osztályok és objektumok](#) című részt.

Konverzió objektummá

Ha egy objektumot konvertálsz objektummá, nem változik meg. Ha más típusú értéket konvertálsz objektummá, akkor létrejön egy új példány keletkezik a beépített *stdClass* osztályból. Ha az érték **NULL** volt, az új példány üres lesz. minden más érték esetén egy *scalar* nevű attribútum fogja tartalmazni az értéket.

```
<?php
$obj = (object) 'ciao';
echo $obj->scalar; // kimenet: 'ciao'
?>
```

Erőforrások

Az erőforrás egy olyan speciális változó, ami egy külső erőforrásra tartalmaz referenciát. Az

erőforrásokat speciális függvények hozzák létre és használják. Lásd az ide tartozó [függeléket](#) a különböző erőforrás típusokhoz tartozó függvények listájával.

Megjegyzés: Az erőforrás típus a PHP 4-ben került a nyelvbe.

Lásd még a [get_resource_type\(\)](#) függvényt.

Konverzió erőforrásra

Mivel az erőforrások speciális azonosítók amelyek nyitott állományokra, adatbázis-kapcsolatokra, képvászon-területekre és hasonlókra mutatnak, nem konvertálhatsz bármilyen értéket erőforrássá.

Erőforrások felszabadítása

A PHP 4-es Zend Engine-jében bevezetett hivatkozás-számlálási technika következtében a használatlan erőforrásokat automatikusan detektálja a PHP (mint a Java). Ebben az esetben minden erőforrás által használt összes más erőforrás felszabadításra kerül a szemétgyűjtő algoritmusban. Emiatt ritkán szükséges, hogy felszabadítsuk a foglalt memóriát valamelyen `free_result` jellegű függvénygel.

Megjegyzés: Az állandó adatbázis kapcsolatok speciálisak abban az értelemben, hogy *nem* szűnnek meg a szemétgyűjtés során. Lásd még az [állandó kapcsolatok](#) című részt.

NULL

A speciális **NULL** érték jelzi, hogy egy változó nem tartalmaz értéket. A **NULL** a **NULL** egyetlen lehetséges értéke.

Megjegyzés: A NULL típus a PHP 4-ben került bevezetésre.

Egy változó **NULL**-nak tekintendő, ha

- a **NULL** állandó értéke lett hozzárendelve.
- ha még semmilyen érték nem lett hozzárendelve.
- ha az [unset\(\)](#) függvény törölte.

Szintaxis

Csak egy értéket vehet fel a **NULL** típus, ami a kisbetűs és nagybetűs írásmódra nem érzékeny **NULL** kulcsszó.

```
<?php  
$valtozo = NULL;  
?>
```

Lásd még: [is_null\(\)](#) és [unset\(\)](#)!

E dokumentációban szereplő pszeudo-típusok

mixed

A *mixed* olyan paramétert jelöl, amely több típust elfogad (de nem feltétlenül bármit).

Például a [gettype\(\)](#) bármilyen típust elfogad, viszont a [str_replace\(\)](#) csak stringet és tömböt fogad el.

number

A *number* jelöli azt, ha egy paraméter [integer](#) vagy [float](#) típusú lehet.

callback

Egyes függvények, mint például a [call_user_func\(\)](#) vagy a [usort\(\)](#) felhasználó által definiált "visszahívható" függvényt fogad el paraméterként. A visszahívható függvények nemcsak egyszerű függvények lehetnek, hanem objektum-metódusok vagy statikus osztály-metódusok.

A PHP függvénynek csak egyszerűen át kell adni a nevét. Bármilyen beépített vagy felhasználó által definiált függvényt megadhatsz kivéve a következőket: [array\(\)](#), [echo\(\)](#), [empty\(\)](#), [eval\(\)](#), [exit\(\)](#), [isset\(\)](#), [list\(\)](#), [print\(\)](#) és [unset\(\)](#).

Egy objektum-metódust tömbként kell megadni úgy, hogy a 0 indexű elem az objektumot tartalmazza, az 1 indexű elem pedig a metódus nevét.

Statikus osztálymetódusok is átadhatók anélkül, hogy példányosítanánk. Ebben az esetben a 0 indexnél az osztály nevét kell megadni objektum helyett.

Példa 11-13. Példák visszahívható függvény használatára

```
<?php
// egy példa visszahívandó függvény
function sajat_visszahivhato_fuggveny() {
    echo 'Hello világ!';
}

// egy példa visszahívandó metódus
class Osztalyocskam {
    function sajatVisszahivhatoMetodus() {
        echo 'Hello Világ!';
    }
}

// 1. típus: Egyszerű visszahívható függvény alkalmazása
call_user_func('sajat_visszahivhato_fuggveny');

// 2. típus: Statikus osztálymetódus visszahívása példányosítás nélkül
call_user_func(array('Osztalyocskam', 'sajatVisszahivhatoMetodus'));

// 3. típus: Objektum metódusának visszahívása
$obj = new Osztalyocskam();
call_user_func(array(&$obj, 'sajatVisszahivhatoMetodus'));
?>
```

Bűvész kedés a típusokkal

A PHP nem követeli meg (nem támogatja) az explicit típusdefiníciót a változók deklaláráskor; egy változó típusát a környezet határozza meg, amiben a változót használjuk. Vagyis ha egy stringet rendelünk \$var-hoz, akkor \$var string lesz. Ha ezután egy egészet rendelünk hozzá, \$var egész lesz.

Egy példa a PHP automatikus típuskonverziójára az összeadás '+' operátora. Ha bármely operandusa lebegőpontos, akkor mindegyik operandusból lebegőpontos lesz, és az eredmény is az lesz. Máskülönben, ha az operandusok egészek, az eredmény is egész lesz. Figyeljünk azonban arra, hogy az operandusok típusát NEM változtatja meg; csupán a kiértékelés módját határozza meg.

```
<?php  
$size = "0";           // $size egy string (ASCII 48)  
$size += 2;            // $size most egész (2)  
$size = $size + 1.3;   // $size most lebegőpontos (3.3)  
$size = 5 + "10 Kis Pingvin"; // $size egész (15)  
$size = 5 + "10 Kismalac"; // $size egész (15)  
?>
```

Ha az utolsó két példa furcsának tűnik, nézd meg a [String konvertálása számmá](#) című részt.

Ha egy változót adott típussal szeretnél használni egy kifejezésben, nézd meg a [Típus konverziókat](#). Ha a változó típusát akarod megváltoztatni, használd a [settype\(\)](#) függvényt.

Ha az ebben a fejezetben található példákat ki szeretnéd próbálni, használd a [var_dump\(\)](#) függvényt a típusok és értékek kiírására.

Megjegyzés: Egy tömb típusának automatikus konverziója jelenleg nem definiált.

```
<?php  
$a = "1";    // $a egy string  
$a[0] = "f"; // Mi van a string offsettel? Mi történik?  
?>
```

Mivel a PHP (történeti okokból) támogatja, hogy a stringet, mint karakterek tömbjét kezeljük, és "beleindexeljünk" a fenti példa problémához vezet: \$a-ból most tömb legyen, aminek az eleme "f", vagy a string első karaktere legyen "f"?

Több információért lásd a [String karaktereinek elérése és módosítása](#) című részt.

A PHP jelen verziói a második értékadást string offszet meghatározásaként értelmezik, így az \$a értéke "f" lesz, mindenmellett, hogy ennek a konverziónak az eredményét határozatlannak tekintjük. A PHP 4 bevezette a kapcsos zárójeles szintaxist a string karaktereinek elérésére, használd ezt a szintaxist a fent bemutatott helyett.

```
<?php  
$a     = "abc"; // $a string  
$a{1} = "f";   // $a itt "afc"  
?>
```

Típuskonverziók

PHP-ben a típuskonverzió úgy működik, mint C-ben: a kívánt típus nevét zárójelbe írjuk az elé a változó elő, amelyet cast-olni (konvertálni) akarunk.

```
<?php  
$size  = 10;           // $size egész
```

```
$bigyo = (boolean) $size; // $bigyo logikai  
?>
```

A megengedett típusok:

- (int), (integer) - egésszé konvertál
- (bool), (boolean) - logikai értékké konvertál
- (real), (double), (float) - lebegőpontos számmá konvertál
- (string) - stringgé konvertál
- (array) - tömbbé konvertál
- (object) - objektummá konvertál

Megjegyzés: szóközök és tabulátorok megengedettek a string belsejében, tehát az alábbiak ugyanazt csinálják:

```
<?php  
$size = (int) $bigyo;  
$size = ( int ) $bigyo;
```

Megjegyzés: Egy változót úgy is striggé konvertálhatsz, hogy dupla idézőjelek közé teszed.

```
<?php  
$size = 10; // $size is an integer  
$str = "$size"; // $str is a string  
$fst = (string) $size; // $fst is also a string  
  
// Ez azt írja ki, hogy "azonosak"  
if ($fst === $str) {  
    echo "azonosak";  
}  
?>
```

Nem minden tiszta, hogy mi történik, ha típusok közt cast-olunk [implicit cast]. További információkért lásd az alábbi fejezeteket:

- [Logikai értékké alakítás](#)
- [Egész értékké alakítás](#)
- [Lebegőpontos értékké alakítás](#)
- [Stringgé alakítás](#)
- [Tömbbé alakítás](#)
- [Objektummá alakítás](#)
- [Erőforrássá alakítás](#)
- [Típuskonverziós táblázat](#)

12. Fejezet. Változók

Alapok

PHP-ban a változókat egy dollárjel utáni változónév jelöli. A változónevek érzékenyek kis- és nagybetűk különbözőségére.

A változónevekre a PHP más jelzőivel azonos szabályok vonatkoznak. Egy érvényes változónév betűvel vagy aláhúzással kezdődik, amit tetszőleges számú betű, szám vagy aláhúzás követ. Reguláris kifejezéssel kifejezve ez a következőt jelenti: '[a-zA-Z_]\x7f-\xff][a-zA-Z0-9_]\x7f-\xff]*'

Megjegyzés: Ebben az esetben egy betű lehet az angol abc egy betűje a-z-ig és A-Z-ig, valamint a 127-től 255-ig terjedő (0x7f-0xff) ASCII kódú karakterek.

Lásd még a [Változókkal kapcsolatos függvényeket](#).

```
<?php
$var = 'Géza';
$Var = 'János';
echo "$var, $Var"; // kiírja, hogy "Géza, János"

$4site      = 'ez nem jó'; // nem érvényes, mert számmal kezdődik
$_4site     = 'ez ok'; // érvényes, aláhúzással kezdődik
$täyte     = 'mansikka'; // érvényes, az 'ä' a kiterjesztett ASCII 228-as karaktere
$tükörfúrógép = 'árvíztűrő'; // érvényes, ellenőrizheted egy ASCII táblában
?>
```

PHP 3-ban a változóhoz minden értékek tartoznak. Vagyis ha egy kifejezést rendelünk egy változóhoz, az eredeti kifejezés egészének értéke másolódik a célváltozóba. Ez azt jelenti, hogy ha például egy változó értékét egy másikhoz rendeljük, egyikük megváltozása sincs hatással a másikra. Nézd át [Kifejezések](#) c. fejezetet, ahol az ilyen jellegű hozzárendelések több információ található.

PHP 4-től lehetőség van egy másik hozzárendelési módra: [változó referencia szerinti hozzárendelésre](#). Ez azt jelenti, hogy az új változó egyszerűen hivatkozik (más szóval "alias lesz", vagy "rá mutat") az eredetire. Az új változón végzett változtatások az eredetit is érintik és fordítva. Ez azt is jelenti, hogy nem történik másolás; ekképpen a hozzárendelés gyorsabban történik meg. Igaz ugyan, hogy ez a sebességnövekedés csak "feszes" ciklusokban vagy nagy [tömbök](#) ill. [objektumok](#) átadásakor jelentkezik.

Referencia szerinti értékkedáshoz egyszerűen & jelet kell az átadandó változó neve elő írni. Az alábbi kód - például - kiírja kétszer, hogy 'Nemem Bob':

```
<?php
$size = 'Bob'; // 'Bob' hozzárendelése $size-hoz
$bigyo = &$size; // Hivatkozás $size-ra $bigyo-ban.
$bigyo = "Nemem $bigyo"; // $bigyo megváltoztatása...
echo $bigyo;
echo $size; // $size is megváltozott
?>
```

Fontos megjegyezni, hogy csak megnevezett változókra lehet referenciát létrehozni.

```
<?php
$size = 25;
$bigyo = &$size; // Ez egy érvényes hozzárendelés.
$bigyo = &(24 * 7); // Érvénytelen referencia egy névtelen kifejezésre.

function test()
{
    return 25;
}

$bigyo = &test(); // Érvénytelen.
?>
```

Előre definiált változók

A PHP egy számos előre definiált változót biztosít az bármely futó szkript számára. Sokat ezek közül nem lehet teljes pontosságal dokumentálni, mert függnek a a futtató szervertől, a használt verziótól, a konfigurálástól, és egyéb tényezőktől. Néhány ilyen változó nem elérhető, ha a PHP parancssorból fut. Ezen változókról találsz egy listát [Fenntartott előredefiniált változók](#) c. részben.

Figyelem

A PHP 4.2.0 és későbbi verzióiban a [register_globals](#) direktíva alapértelmezett értéke *off*. Ez a PHP-nek egy jelentős változása. A register_globals *off*-ra állítása befolyásolja az előredefiniált változók készletét a globális hatáskörben. Például a *DOCUMENT_ROOT* lekéréséhez a *\$_SERVER['DOCUMENT_ROOT']* formát kell használnod *\$DOCUMENT_ROOT* helyett, vagy <http://www.pelda.hu/teszt.php?id=3> URL esetén *\$_GET['id']*-t az *\$id* helyett, vagy *\$_ENV['HOME']* alakot a *\$HOME* helyett.

Több információért olvasd el a [register_globals](#)-ról szóló bejegyzést, a biztonságról szóló fejezetben a [Register Globals használatáról](#) szóló részt, valamint a PHP [4.1.0](#) és [4.2.0](#) verzióinak bejelentését.

A PHP-ben rendelkezésre álló fenntartott előredefiniált változók, mint például a [szuperglobális tömbök](#) használata ajánlottabb.

A 4.1.0-s verziótól a PHP további előredefiniált tömböket biztosít, amelyek webszerverektől (ha van), környezetből, és felhasználótól származó változókat tartalmaznak. Ezek az új tömbök elég sajátosak, mivel automatikusan globálisak, azaz automatikusan rendelkezésre állnak minden hatáskörben. Ezért gyakran nevezik ezeket 'autoglobális' vagy 'szuperglobális' tömböknek. (A felhasználónak nincs lehetősége szuperglobális változók definiálására.) A szuperglobális tömbök alább vannak felsorolva; viszont a PHP előredefiniált változóinak tartalma és természetüknek tárgyalása a [Fenntartott előredefiniált változók](#) című részben olvasható, ahol a régebbi előredefiniált változók (*\$HTTP_*_VARS*) létezéséről is tudomást szerezhetsz. Az 5.0.0 változattól kezdődően az [előre definiált hosszú PHP változók](#) létrehozása kikapcsolható a [register_long_arrays](#) direktíva segítségével.

Változó változók: A szuperglobális változók nem használhatók [változó változókként](#) függvényekben és osztályok metódusaiban.

Megjegyzés: Annak ellenére, hogy egyidőben létezhet a szuperglobális és a *HTTP_*_VARS* tömb, ezen nem azonosak, tehát az egyiknek a módosítása nem változtatja meg a másikat is.

Ha bizonyos változók nincsenek megjelölve a [variables_order](#)-ben akkor a megfelelő PHP előredefiniált tömbök üresek.

PHP szuperglobális változók

[\\$GLOBALS](#)

Minden olyan változóhoz tartalmaz egy referenciát, amely pillanatnyilag a szkript globális hatáskörében létezik. Ennek a tömbnek a kulcsai a globális változók nevei. A *\$GLOBALS* tömb a PHP 3-tól létezik.

[\\$_SERVER](#)

Olyan változók amelyeket a webszerver állított be vagy amelyek szorosan összefüggenek a

szkript futtatási környezetével. Analógiája a régebbi `$HTTP_SERVER_VARS` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott).

[\\$_GET](#)

HTTP GET kérés által létrejövő változók. Analógiája a régebbi `$HTTP_GET_VARS` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott).

[\\$_POST](#)

HTTP POST kérés által létrejövő változók. Analógiája a régebbi `$HTTP_POST_VARS` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott).

[\\$_COOKIE](#)

HTTP sütikból (cookie) származó változók. Analógiája a régebbi `$HTTP_COOKIE_VARS` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott).

[\\$_FILES](#)

HTTP post fájlfeltöltés által létrejövő változók. Analógiája a régebbi `$HTTP_POST_FILES` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott). További információ a [POST típusú fájlfeltöltésről szóló fejezetben](#).

[\\$_ENV](#)

A környezetből származó változók. Analógiája a régebbi `$HTTP_ENV_VARS` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott).

[\\$_REQUEST](#)

HTTP GET, POST és COOKIE bementekből származó változók, azaz amelyekben nem lehet megbízni. A jelenléte és a változók tömbbe való besorolásának sorrendje a PHP [variables_order](#) konfigurációs direktívája szerint van definiálva. Ennek nincs közvetlen analógiája a PHP 4.1.0-nál előbbi verzióiban. Lásd még: [import_request_variables\(\)](#).

Figyelem

A PHP 4.3.0 óta a `$_FILES`-ban szereplő fájlinformációk nem szerepelnek a `$_REQUEST`-ben.

Megjegyzés: [Parancssori](#) futtatás esetén ez *nem* tartalmazza az `argv` and `argc` bejegyzésekét; ezek a `$_SERVER` tömbben szerepelnek.

[\\$_SESSION](#)

Olyan változók, amelyek a szkript munkamenetéhez vannak rendelve. Analógiája a régebbi `$HTTP_SESSION_VARS` tömbnek (amely még mindig rendelkezésre áll, de használata nem ajánlott). További információért lásd a [Munkamenet kezelő függvények](#) című részt.

Változók hatásköre

[Ezt a fejezetet érdemes elolvasni, még akkor is, ha profi vagy valamilyen programozási nyelvben, mert a PHP tartogat egy-két érdekes meglepetést...]

A változó hatásköre az a környezet, amelyben a változó definiált. A legtöbb esetben minden PHP változónak egyetlen hatásköre van. Ez az egyetlen hatáskör kiterjed az include és a require segítségével használt fájlokra is. Például:

```
<?php  
$a = 1;  
include 'b.inc';  
?>
```

Itt az `$a` változó elérhető lesz az beillesztett `b.inc` szkriptben is. A felhasználói függvényekkel a lokális függvényhatáskör kerül bevezetésre. Alapértelmezés szerint minden, függvényen belül használt változó ebbe a lokális függvényhatáskörbe tartozik, például:

```
<?php  
$a = 1; /* globális hatáskör */  
  
function Test ()  
{  
    echo $a; /* egy helyi változót vár */  
}  
  
Test();  
?>
```

Ez a szkript nem fog semmilyen kimenetet sem eredményezni, mivel az echo kifejezés az `$a` változónak egy helyi - függvényen belüli - változatára utal, és ebben a hatáskörben ehhez nem rendeltek értéket. Ez valamelyest különbözik a C nyelv filozófiájától, ahol a globális változók automatikusan elérhetők bármely függvényből, feltéve ha a függvényben újra nem definiáltad azt a változót. Ez problémák forrása lehet, ha az ember véletlenül megváltoztat egy globális változót. A PHP-ben a globális változókat global kulcsszóval kell deklarálni a függvényekben.

A global kulcsszó

Először nézzünk egy példát a *global* használatára:

Példa 12-1. A global használata

```
<?php  
$a = 1;  
$b = 2;  
  
function Osszead()  
{  
    global $a, $b;  
  
    $b = $a + $b;  
}  
  
Osszead();  
echo $b;  
?>
```

A fenti szkript kiírja, hogy "3". `$a` és `$b` global-ként való deklarálásával minden utalás ezekre a változókra a globális változót fogja érinteni. Nincs megkötve, hány globális változót kezelhet egy függvény.

Globális változók elérésének másik módja a PHP által definiált speciális `$GLOBALS` tömb használata. Az előbbi példával egyenértékű megoldás:

Példa 12-2. A `$GLOBALS` használata global helyett

```
<?php
$a = 1;
$b = 2;

function Osszead()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Osszead();
echo $b;
?>
```

A `$GLOBALS` asszociatív tömb, ahol a globális változó neve jelenti a kulcsot, és a változó értéke a tömbelem értéke. Vedd észre, hogy a `$GLOBALS` tömb minden hatáskörben létezik, mivel a `$GLOBALS` egy [szuperglobális változó](#). A következő példa a szuperglobális változók erejét szemlélteti.

Példa 12-3. Példa szuperglobális változók és a hatáskör szemléltetésére

```
<?php
function global_teszt()
{
    // A legtöbb előredefiniált változó nem "szuper" és
    // megkövetelik a 'global' használatát, hogy elérhetőek
    // legyenek függvények helyi hatáskörében.
    global $_HTTP_POST_VARS;

    echo $_HTTP_POST_VARS['nev'];

    // A szuperglobális változók elérhetőek bármilyen
    // hatáskörben, és nem szükséges a 'global' kulcsszó.
    // A szuperglobális változók a PHP 4.1.0-től használhatóak,
    // a $_HTTP_POST_VARS pedig már elavultnak számít.
    echo $_POST['nev'];
}
?>
```

Statikus változók használata

A változók hatáskörének másik fontos lehetősége a *static* (statikus) változó. A statikus változó csak lokális hatáskörben él - egy függvényen belül, de két függvényhívás között nem veszti el az értékét, a változó hatásköréből való kilépés esetén is megmarad az értéke:

Példa 12-4. Statikus változók szükségességét szemléltető példa

```
<?php
function Test()
{
    $a = 0;
    echo $a;
    $a++;
}
?>
```

Ez nagyon haszontalan függvény, mivel nem csinál mást, mint minden meghívásakor `$a`-t 0-ra állítja, aztán kiírja a 0-t. Az `$a++` teljesen felesleges, mert amint vége a függvény futásának az `$a`

változó megszűnik. Ahhoz, hogy ebből értelmes számlálófüggvény legyen - megmaradjon a számláló értéke -, az `$a` változót statikusnak kell deklarálni:

Példa 12-5. Példa statikus változók használatára

```
<?php
function Test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Most már valahányszor meghívódik a `Test()` függvény, kiírja `$a` értékét, majd azt megnöveli eggyel.

A statikus változókat a rekurzív függvények megvalósításában is felhasználhatjuk. Rekurzívnak nevezzük azt a függvényt, amely saját magát hívja meg. Ezt természetesen feltételhez kell kötni, nehogy végtelen rekurzióba kerüljön a vezérlés és meghatározatlan időre a függvényen belül csapdába esik. Mindig meg kell bizonyosodni arról, hogy megfelelő feltétel rendelkezésre áll a rekurzió befejezéséhez. A következő függvény rekurzívan elszámol 10-ig a statikus `$count` változó segítségével: [A static kulcsszó nagyon fontos!]

Példa 12-6. Statikus változók rekurzív függvényekkel

```
<?php
function Test()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test();
    }
    $count--;
}
?>
```

Megjegyzés: Statikus változókat a fenti példákban szereplő módon lehet deklárnai. Ha olyan értéket próbálsz értékül adni neki, amely egy kifejezés eredménye, az parse error-t fog okozni.

Példa 12-7. Statikus változók deklarálása

```
<?php
function ize() {
    static $int = 0;           // jó
    static $int = 1+2;         // rossz (mivel ez egy kifejezés)
    static $int = sqrt(121);  // rossz (szintén kifejezés)

    $int++;
    echo $int;
}
?>
```

Referenciák globális és statikus változókkal

A PHP 4-et működtető Zend Engine 1, a [statikus](#) és [globális](#) változó módosítókat [referenciákkal](#) implementálja. Például egy valódi globális változó beemelve egy függvény hatáskörébe a *global* utasítással tulajdonképpen létrehoz egy referenciát a globális változóhoz. Ez váratlan

viselkedésmódhoz vezethet, amelyet a következő példa illusztrál:

```
<?php
function test_global_ref() {
    global $obj;
    $obj = &new stdclass;
}

function test_global_noref() {
    global $obj;
    $obj = new stdclass;
}

test_global_ref();
var_dump($obj);
test_global_noref();
var_dump($obj);
?>
```

A példában szereplő kód végrahajtása a következő kimenetet produkálja:

```
NULL
object(stdClass) (0) { }
```

Hasonló viselkedésmód jellemzi a *static* utasítást. A referenciák nem statikusan vannak tárolva:

```
<?php
function &get_instance_ref() {
    static $obj;

    echo 'Statikus objektum: ';
    var_dump($obj);
    if (!isset($obj)) {
        // Hozzárendel egy referenciát a statikus változóhoz
        $obj = &new stdclass;
    }
    $obj->property++;
    return $obj;
}

function &get_instance_noref() {
    static $obj;

    echo 'Statikus objektum: ';
    var_dump($obj);
    if (!isset($obj)) {
        // Hozzárendeli az objektumot a statikus változóhoz
        $obj = new stdclass;
    }
    $obj->property++;
    return $obj;
}

$obj1 = get_instance_ref();
$still_obj1 = get_instance_ref();
echo "\n";
$obj2 = get_instance_noref();
$still_obj2 = get_instance_noref();
?>
```

A példa végrahajtása a következő kimenetet eredményezi:

```
Statikus objektum: NULL
Statikus objektum: NULL

Statikus objektum: NULL
Statikus objektum: object(stdClass) (1) {
    ["property"]=>
        int(1)
}
```

A példa azt szemlélteti, hogy amikor hozzárendelsz egy referenciát egy statikus változóhoz, az nem lesz megjegyezve amikor újra meghívod a `&get_instance_ref()` függvényt.

Változó változók

Néha kényelmes változó változók használata. Ez olyan változó jelent, amelynek a nevét dinamikusan lehet beállítani. A normál változót így adunk értéket:

```
<?php
$a = 'hello';
?>
```

A változó változó veszi egy változó értékét, amelyet egy másik változó értékének tekinti. A fenti példában a `hello`, egy változó neveként használható, a `$a` elé még egy `$`-t írva.

```
<?php
$$a = 'világ';
?>
```

Ekkor már két változó van a PHP szimbólumtáblájában: `$a`, amelynek tartalma "hello", és `$hello`, amelynek a tartalma "világ". Ennélfogva a következő programsor:

```
<?php
echo "$a ${$a}";
?>
```

pontosan ugyanazt csinálja, mintha ezt írtuk volna:

```
<?php
echo "$a $hello";
?>
```

Mindkettő kiírja, hogy: **hello világ**.

Annak érdekében, hogy változó változókat tömbökkel együtt is használhassuk, fel kell oldani a következő kétértelműséget. A `$$a[1]` kifejezés kiértékelésekor a feldolgozónak tudnia kell, hogy ez a `$a[1]` értékét tekintse a hivatkozott változó neveként, vagy `$$a`-t - és ekkor és ennek a tömbnek 1. indexű elemére történt a hivatkozás. Az első esetben `$$a[1]`, míg a másodikban `$$a[1]` írandó.

Figyelem

Jó, ha észben tartod, hogy változó változókat nem használhatsz a PHP [szuperglobális tömbjeivel](#) függvényekben és osztályeljárásokban.

Változók a PHP-n kívülről

HTML űrlapok (GET és POST)

Egy HTML űrlap (form) elküldésével az űrlapból származó összes adat automatikusan elérhetővé válik a szkript számára a PHP segítségével. Többféle mód létezik ezen információk elérésére:

Példa 12-8. Egy egyszerű HTML form

```
<form action="ize.php" method="post">
    Nev: <input type="text" name="usernev" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Elküld" />
</form>
```

Az egyéni beállításaidtól függően több mód is létezik HTML formok által beküldött adatok elérésére. Íme néhány példa:

Példa 12-9. Egy egyszerű HTML POST űrlapból származó adatok elérése

```
<?php
// PHP 4.1.0-tól használható

echo $_POST['usernev'];
echo $_REQUEST['usernev'];

import_request_variables('p', 'p_');
echo $p_usernev;

// PHP 3-tól használható. PHP 5.0.0 óta ezek a hosszú előredefiniált
// változók letiltthatók a register_long_arrays direktívával.

echo $HTTP_POST_VARS['usernev'];

// Használható, ha a register_globals PHP direktíva értéke = on.
// A PHP 4.2.0 óta a register_globals alapértelmezett értéke = off.
// E mód használata nem ajánlott.

echo $usernev;
?>
```

Egy GET form használata hasonló, kivéve hogy a megfelelő GET előredefiniált változókat kell használnod. A GET a QUERY_STRING-re vonatkozik (az információ, ami az URL-ben a "?" után szerepel). Például a <http://www.pelda.hu/teszt.php?azon=3> tartalmaz egy GET adatot, amely `$_GET['azon']` módon érhető el. Lásd még: [\\$_REQUEST](#), [import_request_variables\(\)](#)

Megjegyzés: A [szuperglobális tömbök](#), mint például a `$_POST` vagy a `$_GET`, a PHP 4.1.0-től állnak rendelkezésre.

Mint láttuk, a PHP 4.2.0 előtt a [register_globals](#) alapértelmezett értéke *on* volt, PHP 3-ban pedig mindenkor be volt állítva. A PHP közössége azt javasolja mindenkinek, hogy ne alapozzon erre a direktívára, mert az *off*-ra állítását sokkal inkább részesítik előnyben; ennek megfelelően érdemes programozni.

Megjegyzés: A [magic_quotes_gpc](#) konfigurációs beállítás hatással van a GET, POST és COOKIE metódusokkal kapott értékekre. Bekapcsolva a (Ez a "PHP", biz'a!) szövegből automágikusan (Ez a \"PHP\", biz\'a!) lesz, hogy megkönnyítse az adatbázisba írást. Lásd még: [addslashes\(\)](#), [stripslashes\(\)](#) és [magic_quotes_sybase!](#)

A PHP megérti és kezeli a tömbökbe rendezett űrlapváltozókat. (Lásd [kapcsolódó Gy.I.K.!](#)) Hasznos lehet csoportosítani az összetartozó változókat, vagy az olyan űrlapelemeket, ahol több lehetőség közül nem csak egy választható. Példaképpen küldjünk el egy űrlapot saját magának, és elküldéskor jelenítsük meg az adatot.

Példa 12-10. Összetettebb űrlapváltozók

```
<?php
if (isset($_POST['muvelet']) && $_POST['muvelet'] == 'bekuldve') {
```

```
echo '<pre>';
print_r($_POST);
echo '<a href="'. $_SERVER['PHP_SELF'] .'">Próbáld újra</a>';

echo '</pre>';
} else {
?>
<form action=<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
    Név: <input type="text" name="szemelyes[nev]" /><br />
    Email: <input type="text" name="szemelyes[email]" /><br />
    Sör: <br />
    <select multiple name="sor[]">
        <option value="warthog">Warthog</option>
        <option value="guinness">Guinness</option>
        <option value="stuttgarter">Stuttgarter Schwabenbräu</option>
    </select><br />
    <input type="hidden" name="muvelet" value="bekuldve" />
    <input type="submit" name="submit" value="Küld!" />
</form>
<?php
}
?>
```

PHP 3-ban ilyen módon csak egydimenziós tömbök hozhatók létre, míg PHP 4-től nincs ilyen korlátozás.

IMAGE SUBMIT változónevek

Az űrlap elküldésekor megoldható, hogy gomb helyett képet használunk ilyesfélre jelölés segítségével:

```
<input type="image" src="kep.gif" name="elkuldb"/>
```

Ha a felhasználó a képre kattint, a kiszolgálóra a következő két változó jut el: `elkuldb_x`-et és `elkuldb_y`-t. Ezek tartalmazzák a kattintás képen belüli koordinátáit. A tapasztalatok biztos megjegyeznék, hogy a változónevek nem aláhúzást, hanem pontot tartalmaznak, de a PHP a pontot automatikusan aláhúzássá konvertálja.

HTTP sütik (cookie)

A PHP támogatja a [Netscape specifikációja](#) által definiált HTTP sütiket. A süti olyan mechanizmus, amely a távoli böngészőn tesz lehetővé adattárolást, és így lehetővé teszi a visszatérő felhasználók azonosítását. A sütiket a [`setcookie\(\)`](#) függvényteljesítménnyel lehet beállítani. A sütik a HTTP fejléc részei, így a SetCookie függvényt bármilyen kimenet generálása előtt kell meghívni. Ugyanezt a megkötést kell betartani a [`header\(\)`](#) függvényteljesítménnyel. A süti adatok a megfelelő tömbökben találhatóak, mint például a `$_COOKIE`, `$_HTTP_COOKIE_VARS` valamint a `$_REQUEST`. További részleteket és példákat a kézikönyv [`setcookie\(\)`](#) oldalán találsz.

Ha több adatot akarsz rendelni egy sütiváltozóhoz, egyszerűen tekintsd egy tömbnek. Például:

```
<?php
    setcookie("Sutikem[ize]", 'Teszt 1', time() + 3600);
    setcookie("Sutikem[bigyo]", 'Teszt 2', time() + 3600);
?>
```

Ez két különböző süti, viszont a szkriptedben a Sutikem egyetlen tömb lesz. Ha egy sütből szeretnél tárolni összetett adatokat, használhatod a [`serialize\(\)`](#) vagy az [`explode\(\)`](#) függvényeket.

Vigyázz, mert a süti felülírja az előző azonos nevű sütit, ha csak nem különbözik a path vagy a domain. Így pl. egy bevásárlókocsi megírásakor jó egy számlálót is elhelyezni, hogy elkerüljük a problémát.

Példa 12-11. Példa a [setcookie\(\)](#) használatára

```
<?php
if (isset($_COOKIE['szamlalo'])) {
    $szamlalo = $_COOKIE['szamlalo'] + 1;
} else {
    $szamlalo = 1;
}
setcookie('szamlalo', $szamlalo, time() + 3600);
setcookie("Kosar[$szamlalo]", $elem, time() + 3600);
?>
```

Pontok a bejövő változónevekben

Általában a PHP nem változtatja meg a változók neveit, amikor a szkript megkapja őket. A pont viszont nem érvényes karakter egy PHP változóneven belül. Az ok megértéséért nézd ezt:

```
<?php
$varname.ext; /* érvénytelen változónév */
?>
```

Most az elemző lát egy *\$varname* nevű változót, amelyet stringösszefűző operátor (.) követ, majd egy csupasz sztring: az ext - idézőjel nélküli string, amely nem egyezik semmilyen ismert vagy lefoglalt kulcsszóval. Ez nyilván nem a kívánt eredményt adná.

Emiatt fontos, hogy PHP automatikusan helyettesíti a pontokat aláhúzásjellel.

Változótípusok meghatározása

Mivel a PHP határozza meg a változók típusát és konvertálja őket (általában) szükség szerint, nem minden nyilvánvaló, hogy milyen típusú egy pillanatban egy adott változó. A PHP-nek számos függvénye van, amelyek egy változó típusát hivatottak eldönten, mint például: [gettype\(\)](#), [is_array\(\)](#), [is_float\(\)](#), [is_int\(\)](#), [is_object\(\)](#), és [is_string\(\)](#). Lásd még a [típusokról szóló fejezetet](#).

13. Fejezet. Állandók

Az állandó egy skalár érték azonosítója (neve). Mint ahogy az elnevezése is mutatja, a program futása során nem változik/hat meg az értéke. Kivételt képeznek a [mágikus konstansok](#), amelyek tulajdonképpen nem is konstansok. Az állandók alapesetben érzékenyek a kis- és nagybetűs írásmódra. Megállapodás szerint általában csupa nagybetűs neveket használunk az állandók neveiként.

Az állandók neveire a PHP más jelzőivel azonos szabályok vonatkoznak. Egy érvényes állandó-név betűvel vagy aláhúzással kezdődik, amit tetszőleges számú betű, szám vagy aláhúzás követ. Reguláris kifejezéssel kifejezve ez a következőt jelenti: /a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*

Példa 13-1. Helyes és helytelen konstans nevek

```
<?php
// Helyes konstans nevek
define("IZE", "valami");
define("IZE2", "valami más");
```

```
define("IZEÜBIGYO", "valami egészen más");

// Helytelen konstans nevek
define("2IZE",      "valami");

// Ez helyes, de jobb nem használni, mivel
// előfordulhat, hogy egyszer a PHP-nek lesz
// ilyen nevű mágikus konstansa, ami
// működésképtelené teszi a szkriptedet
define("__IZE__",    "valami");

?>
```

Megjegyzés: Ebben az esetben egy betű lehet az angol abc egy betűje a-z-ig és A-Z-ig, valamint a 127-től 255-ig terjedő (0x7f-0xff) ASCII kódú karakterek.

Hasonlóan a [szuperglobális változókhöz](#), az állandók bárhonnan elérhetőek. Az aktuális hatáskörtől (scope) függetlenül bárhol használhatók. A [változók hatásköre](#) című részben tobbet megtudhatsz a hatáskörökről.

Szintakszis

A [define\(\)](#) függvényel lehet állandót létrehozni, amelynek definiálása után később nem lehet törölni vagy megváltoztatni az értékét.

Csak skaláris adat ([boolean](#), [integer](#), [float](#) vagy [string](#) típusú) lehet egy konstans tartalma.

A konstans értékére a nevének megadásával lehet hivatkozni. A változókkal ellentétben *nem* szabad \$ jelet tenned a konstand neve elő. Egy konstans értékének lekérésére használhatod még a [constant\(\)](#) függvényt is, abban az esetben, ha a konstans nevét dinamikusan szeretnéd megkapni. A [get_defined_constants\(\)](#) függvényel lehet a definiált konstansok listáját megkapni.

Megjegyzés: A konstansok és a (globális) változók különböző névtérben vannak. Ez azt jelenti, hogy a **TRUE** és a **\$TRUE** két különböző dolgot jelent.

Ha egy definiálatlan konstanst próbálsz meg használni, a PHP a konstans nevét veszi karaktersorozatként értékül (KONSTANS és "KONSTANS"). Ilyen esetekben egy [E_NOTICE](#) szintű hiba keletkezik. Lásd még a kézikönyvnek azt a részét, ahol azt tárgyalja, miért nem jó az [Size\[bigyo\]](#) használata, abban az esetben, ha előtte nem definiáltad a *bigyo*-t konstansként. Ha szeretnéd megtudni, hogy egy konstans definiálva van, használ a [defined\(\)](#) függvényt.

A következők a fontosabb különbségek a változókhoz képest:

- Az állandók nevét nem kell dollár jellel \$ kezdeni.
- Az állandók akárhol definíálhatók, és akárhonnan elérhetők, a változók környezeti korlátozásaitól függetlenül.
- Az állandók nem módosíthatóak, és nem törölhetők, miután egyszer létrehozták azokat.
- Az állandók csak skaláris értékeket tartalmazhatnak.

Példa 13-2. Konstansok definiálása

```
<?php
define("KONSTANS", "Helló világ!");
echo KONSTANS; // kiírja, hogy "Helló világ!"
echo Konstans; // kiírja, hogy "Konstans" és hibát eredményez
?>
```

Mágikus konstansok

A PHP számos [előredefiniált állandót](#) biztosít a szkripteknek. Ezek többnyire a különböző kiterjesztések által meghatározott állandók, és csak akkor léteznek, ha az adott kiterjesztés is elérhető/használható akár dinamikusan betöltve vagy előre befordítva.

5 mágikus konstans létezik, amelyek változnak aszerint, hogy hol használják. Például a `__LINE__` értéke attól függ, hogy a szkript melyik sorában használják. Ezek a speciális állandók nem érzékenyek a kis- és nagybetűkre.

Táblázat 13-1. Néhány "mágikus" konstans

Név	Leírás
<code>__LINE__</code>	A fájl aktuális sorának száma.
<code>__FILE__</code>	Az állomány teljes elérési útvonala és neve. Ha include-al beillesztett fájlban használod, a beillesztett fájl nevét tartalmazza. A PHP 4.0.2 óta a <code>__FILE__</code> minden abszolút útvonalat tartalmazza, míg a régebbi verziókban egyes esetekben a relatív útvonalat tartalmazta.
<code>__FUNCTION__</code>	A függvény neve. (A PHP 4.3.0-ban került a nyelvbe.) A PHP 5-től a függvény nevét olyan formában adja meg, ahogy deklarálva volt (figyelembe veszi a kis- és nagybetűket). PHP 4-ben csupa kisbetűvel adja meg a függvény nevét.
<code>__CLASS__</code>	Az osztály neve. (A PHP 4.3.0-ban került a nyelvbe.) A PHP 5-től az osztály nevét olyan formában adja meg, ahogy deklarálva volt (figyelembe veszi a kis- és nagybetűket). PHP 4-ben csupa kisbetűvel adja meg az osztály nevét.
<code>__METHOD__</code>	Az osztály metódusának neve. (PHP 5.0.0-ban került a nyelvbe.) A metódus nevét olyan formában adja meg, ahogy deklarálva volt (figyelembe veszi a kis- és nagybetűket).

Lásd még a [get_class\(\)](#), [get_object_vars\(\)](#), [file_exists\(\)](#) és [function_exists\(\)](#) függvényeket.

14. Fejezet. Kifejezések

A kifejezések a PHP legfontosabb építőkövei. A PHP-ban majdnem minden, amit leírsz, egy kifejezés. A lehető legegyszerűbb és mégis pontos definíciója a kifejezésnek: "mindaz, amelynek értéke van".

A kifejezések legalapvetőbb formái az állandók és a változók. Az "\$a = 5" az '5'-öt az \$a változóhoz rendeli. Az '5'-nek nyilván 5 az értéke, vagyis más szavakkal az '5' olyan kifejezés, melynek értéke 5 (itt az '5' egy egész típusú állandó).

Ez után a hozzárendelés után \$a értéke 5 kell legyen, sőt, \$b = \$a esetén az eredménynek azonosnak kell lennie azzal, mint \$b = 5 esetén. Másképp megfogalmazva az \$a olyan kifejezés, amelynek értéke 5. Ha minden jól megy, pontosan ez fog történni.

Kissé bonyolultabb példák a kifejezésekre a függvények. Például tekintsük az alábbi függvényt:

```
<?php
function ize()
{
    return 5;
?>
```

Megelőlegezzük, hogy nem idegen Töled a függvények koncepciója - ha nem így lenne, akkor olvasd el a [függvényekről](#) szóló fejezetet. Amint az elvárható, a \$c = foo() lényegében azonos \$c =

5 beírásával. A függvények olyan kifejezések, amelyeknek az értéke a visszatérési értékük. Mivel `foo()` 5-tel tér vissza, a '`foo()`' kifejezés értéke 5. Általában a függvények nem arra használatosak, hogy egy bedrótozott számmal térjenek vissza, hanem valamilyen számolt származtatott értékkel.

Természetesen a változóknak a PHP-ben nem kell egésznek lenniük, és nagyon gyakran nem is azok. A PHP négy féle skalártípust támogat: egész ([integer](#)), lebegőpontos ([float](#)), sztring ([string](#)) és logikai ([boolean](#)) értékeket (skalárnak az számít, amit nem lehet kisebb darabokra bontani, eltérően - például - a tömböktől). A PHP két nem skalár típust is támogat: a tömb és objektum típust. Mindegyikükhez lehet értéket rendelni, és mindegyikük lehet egy függvény visszatérési értéke.

A PHP továbbfejleszti a kifejezés fogalmát, ahogy azt sok hasonló nyelv teszi. A PHP egy kifejezés-orientált nyelv, abból a szempontból, hogy majdnem minden kifejezés. Tekintsük a példát, amivel eddig foglalkoztunk: '\$a = 5'. Látható, hogy két értéket szerepel benne: az egész konstans '5' és \$a értéke, amelyet az 5 értékkel frissítettünk. Az igazság azonban az, hogy még egy érték is van a kifejezésben, a magáé a hozzárendelésé. A hozzárendelés maga kiértékeli az átadandó értéket, ebben az esetben az 5-öt. Ez azt jelenti gyakorlatilag, hogy a '\$a=5', a hatásától eltekintve egy kifejezés, amelynek az értéke 5. Ha ezért olyasmit írunk, hogy '\$b=(\$a=5)', akkor az olyasmi, mintha '\$a=5; \$b=5;-t írtunk volna (a pontosvessző jelöli az állítás végét). Mivel a hozzárendelés jobbról balra történik, ez írható zárójelek nélkül is: '\$b=\$a=5'.

A kifejezés-orientáltság másik jó példája az elő- és utónövekményes operátor. A PHP használóinak és sok más nyelvet használónak ismerős lehet a változó++ és változó-- jelölés. Ezek a [növelő](#) és [csökkentő operátorok](#). A PHP/FI 2-ben a '\$a++'-nek nincs értéke (mert nem kifejezés), és ezért nem is lehet semmihez hozzárendelni, vagy valamire használni. A PHP kiterjesztette az inkrementálás/dekrementálás lehetőségeit azzal, hogy ezeket is kifejezésekkel alakította, mint ahogyan a C nyelvben is megtették. A PHP-ben a C nyelvhez hasonlóan kétféle növekményes operátor van: elő- és utó(növekményes). Mindkettő megnöveli a változó értékét, tehát a változó szempontjából nézve nincs különbség, a kifejezés értékében annál inkább. A '++\$változó' formájú előnövekményes a megnövelt értéket adja vissza. (A PHP először növeli a változót, majd kiértékeli, innen jön az 'elő'). A '\$változó++' alakú utónövekményes a változó eredeti értéket adja. (A PHP először kiértékeli, aztán növeli a változót, innen jön az 'utó'). [A csökkentő operátorokkal értelemszerűen ugyanez a helyzet.]

A kifejezések egy gyakori típusai az [összehasonlító kifejezések](#). Ezen kifejezéseknek az értékei **FALSE** vagy **TRUE**. A PHP támogatja a > (nagyobb), >= (nagyobb, vagy egyenlő), == (egyenlő), != (nem egyenlő), < (kisebb) és a <= (kisebb, vagy egyenlő) operátorokat. A nyelv támogat még néhány szigorú ekvivalencia operátort: === (egyenlő és azonos típusú) és !== (nem egyenlő vagy nem azonos típusú). Ezeket a kifejezéseket általában feltételes kifejezésekben szokták használni. Ilyen pl. az *if* kifejezés.

Az utolsó példa, amivel itt foglalkozunk, a kombinált hozzárendelés-operátor kifejezés. Már tudhatod, hogy ha \$a-t egygel szeretnéd megnövelni, akkor egyszerűen írhatod, hogy '\$a++' vagy '++\$a'. Mi van akkor, hogy ha nem egygel, hanem 3-mal kell megnövelni? Lehet háromszor egymás után írni, hogy '\$a++', de ez nem valami hatékony és üdvözítő megoldás. Sokkal jobb gyakorlat az '\$a=\$a+3' használata. Ez kiértékeli az \$a + 3-at, és hozzárendeli az \$a-hoz, amelynek az lesz a végső eredménye, hogy \$a értéke megnő hárommal. A PHP-ben - több más C-szerű nyelvhöz hasonlóan - ennél rövidebb formában is megadható: '\$a+=3'. Ez pontosan azt jelenti, hogy "vedd \$a értékét, és adjál hozzá hármat, és írd vissza az eredményt \$a-ba". Amellett, hogy rövidebb és érthetőbb, még gyorsabb is. '\$a+=3' értéke, mint egy normál értékadásé a hozzárendelt érték. Ez persze NEM 3, hanem \$a + 3 (hiszen ezt az értéket rendeltük \$a-hoz). Az összes többi kétoperandusú operátort lehet ilyenformán hozzárendelő-operátorként használni, például '\$a-=5' (kivon 5-öt \$a-ból), '\$b*=7' (megszorozza \$b-t 7-tel), stb.

Létezik még egy kifejezés, amely elég különösnek tűnhet, ha csak nem láttad már más nyelvekben: a háromoperandusú feltételes operátor:

```
<?php  
$elso ? $masodik : $harmadik  
?>
```

Értéke: ha az első kifejezés értéke **TRUE**(nem nulla), akkor a második részkifejezés, egyébként a harmadik részkifejezés. Egy segítő példa:

```
<php  
$b = $todo == 'növel' ? $a+5 : $a-5; // $b $a+5 lesz, ha $todo értéke 'növel'  
// egyébként $b $a-5 lesz  
?>
```

Ez megegyezik az alábbi kóddal:

```
<php  
if ($todo == 'növel') $b = $a+5 else $b = $a-5;  
?>
```

Az alábbi példa segíthet az elő- és utónövekményes illetve csökkentő operátorok és kifejezéseik jobb megértésében:

```
<?php  
function duplaz($i)  
{  
    return $i*2;  
}  
  
$b = $a = 5;          /*ötöt rendelünk $a és $b változókhoz */  
  
$c = $a++;           /* utónövekményes, $c 5 lesz, mert  
először $a kiértékelődik, csak aztán nő */  
  
$e = $d = ++$b;     /* előnövekményes, $b növelt értéke (6)  
adódik tovább $d-nek és $e-nek */  
  
                    /* itt $d és $e hat */  
  
$f = duplaz($d++);  /* $d kétszeresének hozzárendelése mielőtt  
inkrementálódik, 2*6 = 12 lesz $f */  
  
$g = duplaz(++$e);  /* $e kétszeresének hozzárendelése miután  
inkrementálódik 2*7=14 lesz $g */  
  
$h = $g += 10;        /* először $g 10-zel nő, így 24 lesz az értéke,  
ez adódik tovább $h-nak, az is 24 lesz*/  
?>
```

Egyes kifejezések tekinthetők utasításoknak. Ebben az esetben egy utasítás 'kif ';' alakú, vagyis kifejezés utána pontosvesszővel. A '\$b=\$a=5;' -ben \$a=5 érvényes kifejezés, de nem érvényes utasítás önmagában. A teljes '\$b=\$a=5;' már igen.

Még egy említésre méltó dolog a kifejezések igaz értéke. Sok esetben főleg feltételes elágazásokban és ciklusokban, nem igazán érdekes, hogy a kifejezésnek mi az értéke, csak az, hogy igaz (**TRUE**) vagy hamis (**FALSE**). A **TRUE** és **FALSE** konstansok (nevük nem függ a kis- és nagybetűs írásmódtól) a PHP két lehetséges logikai értékei. Amikor szükséges, a kifejezések automatikusan logikai értékké alakulnak. Lásd a [típuskonverzióról szóló fejezetet](#) további részletekért.

A PHP a kifejezések egy teljes és hatékony megvalósítását biztosítja, és teljes dokumentációja meghaladja e kézikönyv kereteit. A fenti példák rávilágítanak, hogy mi is az a kifejezés, és hogyan kell létrehozni hasznos kifejezéseket. A kézikönyv hátralevő részében az *expr* alatt egy érvényes PHP kifejezést értünk.

15. Fejezet. Operátorok

Az operátor egy olyan doleg, aminek adsz egy vagy több értéket (vagy szakmai nyelven: kifejezést), ami kiad egy másik értéket (úgy hogy a kapott konstrukció kifejezéssé alakuljon). Tehát úgy gondolhatsz a függvényekre és az olyan konstrukciókra, amelyek értéket adnak vissza (pl. print), mint operátorok, azokra pedig, amelyek nem adnak vissza semmit (pl. echo), mint valami másfélé dolgokra.

Három félle operátor létezik. Az első fajta operátor az unáris, amely csak egy értékkel dolgozik, például a ! (a negációs operátor) vagy ++ (a növelő operátor). A második csoportot a bináris operátorok képviselik; ez a csoport foglalja magába a PHP által nyújtott operátorok közül a legtöbbet, találász ezekről egy listát lejjebb az [Operátorok precedenciája](#) című részben.

A harmadik kategória a ternáris operátor: ?: Ezt arra érdemes használni, hogy válasszunk két kifejezés közül egy harmadik kifejezéstől függően, inkább mintsem két mondat vagy két végrehajtási út között. A ternáris kifejezések bezárójelezése igen jó ötlet.

Operátorok precedenciája

Az operátorok precedenciája azt határozza meg, hogy milyen "szorosan" köt össze két kifejezést. Például az $I + 5 * 3$ kifejezésben, a kifejezés értéke 16, és nem 18, mert a szorzás operátorának, a ("*")-nak nagyobb precedenciája van, mint az összeadásénak ("+"). Zárójelek segítségével tetszőleges precedenciát lehet felállítani egy kifejezésen belül, ha szükséges. Például a $(I + 5) * 3$ eredménye 18 lesz.

Az alábbi táblázat az operátorokat precedenciájuk szerint csökkenő sorrendben tartalmazza.

Táblázat 15-1. Operátorok precedenciája

asszociativitás	operátorok	egyéb információ
nem köthető	new	new
jobbról balra	[array()
nem köthető	++ --	növelő/csökkentő operátorok
nem köthető	! ~ - (int) (float) (string) (array) (object) @	típusok
balról jobbra	* / %	aritmetikai operátorok
balról jobbra	+ - .	aritmetikai operátorok és string operátorok
balról jobbra	<< >>	bitorientált operátorok
nem köthető	< <= > >=	összehasonlító operátorok
nem köthető	== != === !==	összehasonlító operátorok
balról jobbra	&	bitorientált operátorok és referenciák
balról jobbra	^	bitorientált operátorok
balról jobbra		bitorientált operátorok
balról jobbra	&&	logikai operátorok
balról jobbra		logikai operátorok

asszociativitás	operátorok	egyéb információ
balról jobbra	? :	a ternáris operátor
jobbról balra	= += -= *= /= .= %= &= = ^= <<= >>=	értékadó operátorok
balról jobbra	and	logikai operátorok
balról jobbra	xor	logikai operátorok
balról jobbra	or	logikai operátorok
balról jobbra	,	sokféle használat

[Az asszociativitás talán megér egy kis magyarázatot. Ez azt határozza meg, hogy az adott szinten levő operátorok egymás utáni, zárójel nélküli használatát hogyan értelmezi a fordító. Egy példán keresztül talán érthetőbbé válik: 1-2+3 értelmezhető $(1-2)+3$ -nak (= 2), vagy $1-(2+3)$ -nak (= -4). Az előbbi kiértékelés, amely *balról jobbra asszociatív*, a helyes ebben az esetben. A második kiértékelés pedig *jobbról balra asszociatív* - ilyen lenne például a hatványozás, amely nincs a nyelvben implementálva (erre szolgál [pow\(\)](#)). Ha a $**$ lenne a hatványozás, akkor a $2^{**}3^{**}2$ helyesen 2 a 9-en = 512 lenne. Vannak nem köthető (nem asszociatív) operátorok is, ilyenek az összehasonlító operátorok. A PHP-ban tehát *nem* értelmes a $2<\$x<4$ kifejezés.]

A balról jobbra asszociativitás azt jelenti, hogy a kifejezés kiértékelése balról jobbra történik, a jobbról balra pedig ennek az fordítottját.

Példa 15-1. Asszociativitás

```
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

Használj zárójeleket, hogy olvashatóbb legyen a kódod!

Megjegyzés: Annak ellenére, hogy a `!` operátornak nagyobb a precedenciája, mint a `=` operátornak, a PHP elfogadja az ehhez hasonló kifejezéseket is: `if (!$a = ize())`, ebben az esetben az `ize()` által visszaadott érték `$a`-ba kerül bele.

Aritmetikai operátorok

Emlékszel az elemi aritmetikára a suliból? Ezek pont úgy működnek, mint ott.

Táblázat 15-2. Aritmetikai operátorok

Példa	Név	Eredmény
<code>-\$a</code>	Negáció	<code>\$a</code> ellentettje
<code>\$a + \$b</code>	Összeadás	<code>\$a</code> és <code>\$b</code> összege
<code>\$a - \$b</code>	Kivonás	<code>\$a</code> és <code>\$b</code> különbsége
<code>\$a * \$b</code>	Szorzás	<code>\$a</code> és <code>\$b</code> szorzata
<code>\$a / \$b</code>	Osztás	<code>\$a</code> és <code>\$b</code> hányadosa
<code>\$a % \$b</code>	Modulus	<code>\$a / \$b</code> maradéka

Az osztás operátor (`/`) minden lebegőpontos számot ad eredményül, mégha minden operandus

egész típusú vagy egész típusúvá alakított string.

Megjegyzés: Ha az $\$a$ negatív, akkor az $\$a \% \b maradék is negatív lesz.

Lásd még a kézikönyv [matematikai függvényekről](#) szóló oldalát.

Hozzárendelő operátorok

Az alapvető hozzárendelő operátor az " $=$ ". Elsőre azt hihetnénk, hogy ez az "egyenlő valamivel" jele. Valójában azt jelenti, hogy a bal oldali operandus [ami az egyenlőségjel bal oldalán áll] a jobb oldali kifejezést kapja értékül.

A hozzárendelő kifejezés értéke a bal oldalhoz rendelt érték. Vagyis a " $\$a = 3$ " értéke 3. Ez lehetőséget ad néhány trükkös dologra:

```
<?php  
  
$a = ($b = 4) + 5; // $a most 9, és $b 4  
  
?>
```

Az alapvető hozzárendelő operátoron felül vannak ún. "kombinált" operátorok is az összes [kétoperandusú aritmetikai](#) és sztring operátorok számára, amelyek lehetővé teszik, hogy használjunk egy változót egy kifejezésben, majd rögtön be is állítsuk a változót a kifejezés értékére. Például:

```
<?php  
  
$a = 3;  
$a += 5; // $a-t 8-ra állítja, mintha $a = $a + 5;-öt írtunk volna  
$b = "Csecs";  
$b .= "Emő"; // $b "Csecs Emő" lesz, egyenértékű párja: $b = $b . "Emő";  
  
?>
```

A hozzárendelés az eredeti változót az újba másolja érték szerint, így az egyiken elvégzett változtatások a másikat nem érintik. Ezt fontos tudni, például egy sokszor végrehajtott ciklus belsejében nagy tömbök másolásakor. A PHP 4 óta használható a `$var =&$othervar;` szintaxisú referencia szerinti érték hozzárendelés is, de ez PHP 3-ban nem működik. A 'referencia szerinti értékhozzárendelés' azt jelenti, hogy minden változó ugyanarra az adatra fog mutatni, és nem történik meg a változó értékének lemasolása. További információkat a referenciákról a [Referenciák részletesen](#) című fejezetben olvashatsz.

Bitorientált operátorok

A bitorientált operátorok teszik lehetővé, hogy egész típusú számokon belül bizonyos biteket beállítsunk, vagy lefedjünk (maszkolás). Ha viszont az operátoron minden két oldalán sztring típusú változó áll, akkor a bitorientált operátorok a sztringek karakterein dolgoznak úgy, hogy a karakterek ASCII kódjain végezik el a műveletet, és az eredményül adódó számot ASCII kóddal megadott karakterek értelmezi.

```
<?php  
echo 12 ^ 9; // '5' -öt ír ki  
  
echo "12" ^ "9"; // kiírja a visszaperjel karaktert (ASCII #8), mert  
// ('1' (ASCII #49)) ^ ('9' (ASCII #57)) = (ASCII #8)
```

```
echo "hallo" ^ "hello"; // eredmény: #0 #4 #0 #0 #0
// 'a' ^ 'e' = #4
?>
```

Táblázat 15-3. Bitorientált operátorok

Példa	Név	Eredmény
\$a & \$b	És	Ott lesz '1' az eredményben, ahol \$a és \$b mindegyikében az a bit '1'-es. minden más biten '0'.
\$a \$b	Vagy	Ott lesz '1' az eredményben, ahol \$a és \$b közül legalább az egyik azon a bitje '1'-es. minden más biten '0'.
\$a ^ \$b	Kizáró vagy	Ott lesz '1' az eredményben, ahol \$a és \$b közül csak pontosan az egyikben '1' állt. minden más biten '0'. [Más közelítésben ott lesz '1' az eredményben, ahol különböző bitek álltak \$a-ban és \$b-ben; megint más közelítésben \$a azon bitjei invertálódnak, amely helyeken \$b-ben '1' áll]
~ \$a	Nem	\$a összes bitjét invertálja
\$a << \$b	Eltolás balra	\$a bitjeit \$b számú bittel balra tolja (minden bitnyi eltolás 2-vel való szorzást jelent [amíg el nem fogynak a bitek, utolsó helyen előjelbit van ??])
\$a >> \$b	Eltolás jobbra	\$a bitjeit \$b számú bittel jobbra tolja (minden bitnyi eltolás 2-vel való egészosztást jelent. [Vigyázz, negatív számot inkább ne tolj jobbra!])

Figyelem

32 bites rendszeren ne végezz 32 bitnyinél nagyobb jobbratolást. Ne végezz balratolást olyankor, amikor az 32 bitnél hosszabb számot eredményezne.

Összehasonlító operátorok

Az összehasonlító operátorok, mint nevük is sugallja, két érték összehasonlítására szolgálnak. Érdekesek lehetnek még [a típus összehasonlítási táblázat](#), mivel mutat néhány példát különböző típusok összehasonlítására.

Táblázat 15-4. Összehasonlító operátorok

Példa	Név	Eredmény
\$a == \$b	Egyenlő	Igaz (TRUE), ha \$a és \$b értéke egyenlő
\$a === \$b	Azonos	Igaz (TRUE), ha \$a és \$b értéke egyenlő, és azonos típusúak (PHP 4-től)
\$a != \$b	Nem egyenlő	Igaz (TRUE), ha \$a és \$b értékei különbözök
\$a <> \$b	Nem egyenlő	Igaz (TRUE), ha \$a és \$b értékei különbözök
\$a !== \$b	Nem azonos	Igaz (TRUE), ha \$a és \$b értékei vagy típusai különbözök (csak PHP 4-től)
\$a < \$b	Kisebb mint	Igaz (TRUE), ha \$a szigorúan kisebb, mint \$b
\$a > \$b	Nagyobb mint	Igaz (TRUE), ha \$a szigorúan nagyobb, mint \$b
\$a <= \$b	Kisebb, vagy egyenlő	Igaz (TRUE), ha \$a kisebb, vagy egyenlő, mint \$b
\$a >= \$b	Nagyobb, vagy	Igaz (TRUE), ha \$a nagyobb, vagy egyenlő, mint \$b

Példa	Név	Eredmény
	egyenlő	

Ha egészet hasonlítasz stringgel, a string előbb [számmá konvertálódik](#). Ha két számot reprezentáló stringet hasonlítasz össze, egész számokként lesznek összehasonlítva. Ezek a szabályok a [switch](#) utasítás esetén is érvényesek.

```
<?php
var_dump(0 == "a"); // 0 == 0 -> true
var_dump("1" == "01"); // 1 == 1 -> true

switch ("a") {
case 0:
    echo "0";
    break;
case "a": // nem éri el az "a"-t, mert egyezett 0-val
    echo "a";
    break;
}
?>
```

Ha az operandusok típusa különböző, akkor az összehasonlítás az alábbi táblázat alapján történik (sorrendben).

Táblázat 15-5. Különböző típusú összehasonlítása

Első operandus típusa	Második operandus típusa	Eredmény
null vagy string	string	NULL konvertálása ""-é, numerikus vagy lexikális összehasonlítás
bool vagy null	bármi	Konvertálás bool -lá, FALSE < TRUE
objektum	objektum	A beépített osztályok definiálhatják saját összehasonlítási módjat, különböző osztályok összehasonlíthatatlanok, azonos osztályok esetén a tulajdonságokat a tömbökhez hasonló módon hasonlítja össze
string , erőforrás vagy numerikus	string , erőforrás vagy numerikus	A stringek és erőforrások átalakulnak számokká, majd a szokásos matematikai összehasonlítás
tömb	tömb	A kisebb elemszámú tömb a kisebb, ha az első operandus kulcsa nem létezik a második operandusban, akkor a tömbök összehasonlíthatatlanok, egyébként a értékről értékre történő összehasonlítás következik (lásd a lenti példát)
tömb	bármi	A tömb minden nagyobb
objektum	bármi	Az objektum minden nagyobb

Példa 15-2. A standard tömbösszehasonlítás másolata

```
<?php
// A tömbök ilyen módon lesznek összehasonlítva,
// standard összehasonlító operátorokkal
function standard_array_compare($op1, $op2)
{
    if (count($op1) < count($op2)) {
        return -1; // $op1 < $op2
```

```
> } elseif (count($op1) > count($op2)) {
>     return 1; // $op1 > $op2
> }
> foreach ($op1 as $key => $val) {
>     if (!array_key_exists($key, $op2)) {
>         return null; // uncomparable
>     } elseif ($val < $op2[$key]) {
>         return -1;
>     } elseif ($val > $op2[$key]) {
>         return 1;
>     }
> }
> return 0; // $op1 == $op2
}
?>
```

Lásd még: [strcasecmp\(\)](#), [strcmp\(\)](#), [tömboperátorok](#), [típusok](#).

A ternáris operátor

Egy másik feltételes operátor a "?:?" (ternáris) operátor.

Példa 15-3. Alapértelmezett érték beállítása

```
<?php
// Példa ternáris operátor használatára
$muvelet = (empty($_POST['muvelet'])) ? 'alap' : $_POST['muvelet'];

// A fenti azonos az alábbi az if/else utasítással:
if (empty($_POST['muvelet'])) {
    $muvelet = 'alap';
} else {
    $muvelet = $_POST['muvelet'];
}
?>
```

A *(expr1) ? (expr2) : (expr3)* kifejezés *kif2-t* értékeli ki, ha *kif1* értéke igaz (**TRUE**), és *kif3-at*, ha *kif1* értéke hamis (**FALSE**).

Megjegyzés: A ternáris operátor egy kifejezést alkot, ezért kiértékelése nem egy változót ad, hanem egy értéket. Ez azért fontos, mert ha egy változót referencia szerint akarsz átadni, és a *return \$var == 42 ? \$a : \$b;* alakot használod, a referencia szerinti átadás nem fog működni és az újabb PHP verziók esetén figyelmeztetést is kapsz.

Hibakezelő operátorok

A PHP egy hibakezelő operátort támogat, az at jelet (@ - kukac). PHP kifejezés elő írva a kifejezés által esetlegesen generált hibaüzenete(ke)t figyelmen kívül hagyja a rendszer.

Ha a [track_errors](#) szolgáltatás be van kapcsolva, bármilyen a kifejezés által generált hibaüzenet a [\\$php_errormsg](#) változóba kerül tárolásra. Ez a változó minden hiba esetén felülíródik, ezért használható információk kinyerése érdekében a kifejezést követően ezt minél hamarabb ellenőrizni kell.

```
<?php
/* Szándékos állomány megnyitási hiba */
$file = @file ('nem_letezo_allomany') or
die ("Nem lehet megnyitni, a hiba: '$php_errormsg'");
```

```
// bármilyen kifejezésre működik, nem csak függvényekre  
$ertek = @$tomb[$kulcs];  
// nem ad notice szintű hibát, ha a $kulcs kulcs nem létezik  
?>
```

Megjegyzés: A `@` operátor csak [kifejezésekre](#) működik. Egyszerű ökoliszabályként alkalmazandó, ha valaminek az értelmezett az értéke, akkor az elé a `@` operátor is oda tehető. Ekképpen például használható változók, függvények és [include\(\)](#) hívások, állandók neve előtt és sok más esetben. Nem használható azonban függvény és osztály definíciók vagy nyelvi szerkezetek (mint például `if` és `foreach` utasítások) előtt.

Lásd még: [error_reporting\(\)](#), valamint a [Hibakezelő és naplózó függvényeket](#).

Figyelem

Jelenleg a `"@"` hibakezelő operátor kikapcsolja azon kritikus hibák jelentését is, amelyek megszakítják a szkript futását. Más problémák mellett, ha függvényből érkező hibaüzenetek elnyelésére használod a `"@"` jelet, le fog állni a szkript futása, ha nem létezik a megadott függvény vagy elírtad annak nevét.

Vérehajtó operátorok

A PHP-ban létezik egy program-vérehajtó operátor: a visszaidézőjel (backtick) [aki tudja az igazi nevét, ne rejtsé véka alá!] (`). Ezek nem szimpla idézőjelek! A PHP megpróbálja a sztring tartalmát parancssorból futtatandó utasításként vérehajtani, amelynek a kimenete lesz az operátor értéke. Ez nem egyszerűen a kimenetre kerül, hanem hozzárendelhető egy változóhoz. A visszaidézőjel (backtick) operátor azonos a [shell_exec\(\)](#) függvényvel.

[Az alábbi kis példa az aktuális könyvtár tartalmát (hosszú lista, rejtett fájlok is) formázva írja ki (fix szélességű betűket használva, újsor karaktereket tiszteletben tartva)]

```
<?php  
$output = `ls -al`;  
echo "<pre>$output</pre>";  
?>
```

Megjegyzés: A vérehajtó operátor nem használható, ha a safe mode be van kapcsolva vagy amikor a [shell_exec\(\)](#) függvény le van tiltva.

Lásd még a kézikönyv [Programfuttató függvények](#) című részét, [popen\(\)](#) [proc_open\(\)](#) függvényeket, valamint a [Parancssori programozás a PHP-ben](#) című fejezetet.

Növelő/csökkentő operátorok

A PHP támogatja a C-szerű ún. elő- és utónövekményes ill. csökkentő operátorokat.

Táblázat 15-6. Növelő/csökkentő operátorok

Példa	Név	Hatás
<code>+\$a</code>	előnövekményes	Növeli \$a-t eggyel, majd visszaadja \$a értékét
<code>\$a++</code>	utónövekményes	Visszaadja \$a értékét, majd növeli \$a-t eggyel
<code>--\$a</code>	előcsökkentő	Csökkenti \$a-t eggyel, majd visszaadja \$a értékét
<code>\$a--</code>	utócsökkentő	Visszaadja \$a értékét, majd csökkenti \$a-t eggyel

Itt egy egyszerű példaprogram:

```
<?php
echo "<h3>Utónövekményes</h3>";
$a = 5;
echo "5-nek kell lennie: " . $a++ . "<br />\n";
echo "6-nak kell lennie: " . $a . "<br />\n";

echo "<h3>Előnövekményes</h3>";
$a = 5;
echo "6-nak kell lennie: " . ++$a . "<br />\n";
echo "6-nak kell lennie: " . $a . "<br />\n";

echo "<h3>Előcsökkentő</h3>";
$a = 5;
echo "5-nek kell lennie: " . $a-- . "<br />\n";
echo "4-nek kell lennie: " . $a . "<br />\n";

echo "<h3>Utócsökkentő</h3>";
$a = 5;
echo "4-nek kell lennie: " . --$a . "<br />\n";
echo "4-nek kell lennie: " . $a . "<br />\n";
?>
```

Aritmetikai műveletek karakterváltozókon való végzésénél a PHP a Perl szabályait követi, és nem a C szabályait. Például a Perlben 'Z'+1 kifejezé értéke 'AA' lesz, a C-ben pedig 'Z'+1 értéke '[' (`ord('Z') == 90`, `ord('[') == 91`). Megjegyzés: a karakterváltozók növelhetőek, de nem csökkenthetőek.

Példa 15-4. Aritmetikai műveletek karakterváltozókon

```
<?php
$i = 'W';
for ($n=0; $n<6; $n++) {
    echo ++$i . "\n";
}
?>
```

A fenti példa a következő kimenetet adja:

```
X
Y
Z
AA
AB
AC
```

Boolean változók növelésének, csökkentésének nincs hatása.

Logikai operátorok

Táblázat 15-7. Logikai operátorok

Példa	Név	Eredmény
\$a and \$b	És	Pontosan akkor igaz (TRUE), ha mind \$a mind \$b igazak (TRUE).
\$a or \$b	Vagy	Pontosan akkor igaz (TRUE), ha \$a és \$b között van igaz (TRUE).
\$a xor \$b	Kizáró vagy	Pontosan akkor igaz (TRUE), ha \$a és \$b közül pontosan egy igaz (TRUE).
! \$a	Tagadás	Pontosan akkor igaz (TRUE), ha \$a nem igaz (TRUE).

Példa	Név	Eredmény
\$a && \$b	És	Pontosan akkor igaz (TRUE), ha mind \$a mind \$b igaz (TRUE).
\$a \$b	Vagy	Pontosan akkor igaz (TRUE), ha \$a és \$b között van igaz (TRUE).

Az "és" és a "vagy" operátorok két variációjára a magyarázat az operátorok precedenciájában van.
(Lásd: [Operátorok Precedenciája](#).)

String operátorok

Két **string** operátor van. Az egyik az összefűzés operátor ('.'), amely bal és jobb oldali operandusának összefűzöttjével tér vissza. A második az összefűző-hozzárendelő operátor ('.='), amely hozzáfüzi a jobb oldalon szereplő szöveges értéket a bal oldali operandus végéhez. Olvasd el a [Hozzárendelő operátorok](#) című részt!

```
<?php
$a = "Para ";
$b = $a . "Zita"; // most $b értéke "Para Zita"

$a = "Dárdarázó ";
$a .= "Vilmos"; // most $a értéke "Dárdarázó Vilmos"
?>
```

Lásd még a kézikönyv [Stringek](#) és [String függvények](#) című részeit.

Tömb operátorok

Táblázat 15-8. Tömb operátorok

Példa	Név	Eredmény
\$a + \$b	Egyesítés	\$a és \$b egyesítése.
\$a == \$b	Egyenlő	TRUE ha \$a és \$b ugyanazokból az kulcs/érték párokból áll.
\$a === \$b	Azonos	TRUE ha \$a és \$b ugyanazokat az kulcs/érték párokat tartalmazza és ugyanolyan sorrendben
\$a != \$b	Nem egyenlő	TRUE ha \$a nem egyenlő \$b-vel.
\$a <> \$b	Nem egyenlő	TRUE ha \$a nem egyenlő \$b-vel.
\$a !== \$b	Nem azonos	TRUE ha \$a nem azonos \$b-vel.

A + a jobboldali tömböt a baloldalihoz fűzi úgy, hogy az ismétlődő indexen levő elem nem írja felül az eredeti elemet.

```
<?php
$a = array("a" => "alma", "b" => "banán");
$b = array("a" => "körte", "b" => "eper", "c" => "cseresznye");

$c = $a + $b; // $a és $b egyesítése
echo "\$a és \$b egyesítése: \n";
var_dump($c);
```

```
$c = $b + $a; // $b és $a egyesítése
echo "\$b és \$a egyesítése: \n";
var_dump($c);
?>
```

When executed, this script will print the following:

```
$a és $b egyesítése:
array(3) {
    ["a"]=>
        string(4) "alma"
    ["b"]=>
        string(5) "banán"
    ["c"]=>
        string(10) "cseresznye"
}
$b és $a egyesítése:
array(3) {
    ["a"]=>
        string(5) "körte"
    ["b"]=>
        string(4) "eper"
    ["c"]=>
        string(10) "cseresznye"
}
```

Az összehasonlítás számára a tömbelemek akkor egyenlőek, ha megegyezik a kulcsuk és az értékük.

Példa 15-5. Tömbök összehasonlítása

```
<?php
$a = array("alma", "banán");
$b = array(1 => "banán", "0" => "alma");

var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
?>
```

Lásd még a [Tömbök](#) és a [Tömbkezelő függvények](#) című részeket.

Típus operátorok

A PHP egyetlen típus operátora az *instanceof*. Az *instanceof* arra való, hogy megállapítsuk, hogy egy adott objektum eleme-e egy bizonyos [objektum osztály](#).

Az *instanceof* operátor a PHP 5-ben került bevezetésre. Azelőtt az [**is_a\(\)**](#) függvényt használták de használata ma nem ajánlott a *instanceof* létezése miatt.

```
<?php
class A { }
class B { }

$valami = new A;

if ($valami instanceof A) {
    echo 'A';
}
if ($valami instanceof B) {
    echo 'B';
}
?>
```

Mivel *\$valami* egy A típusú objektum, de nem B típusú csak az A-tól függő blokk fog

végrehajtódni:

A

Lásd még: [get_class\(\)](#), [is_a\(\)](#).

16. Fejezet. Vezérlési szerkezetek

Az összes PHP szkript utasítások sorozatából áll. Az utasítás lehet hozzárendelő utasítás, függvényhívás, ciklus, feltételes utasítás, vagy üres utasítás. Az utasítások általában pontosvesszővel végződnek. Ezenkívül az utasításokat csoportosítani lehet; utasításblokkba foglalhatók kapcsos zárójelek segítségével. Az utasításblokok maguk is utasítások. A különféle utasítástípusokat ebben a fejezetben tárgyaljuk.

if

Az *if* szerkezet az egyik legfontosabb szerkezete a legtöbb nyelvnek - így a PHP-nek is. A PHP a C-ben megismérthez hasonló *if* szerkezzettel bír:

```
if (kifejezés)
    utasítás
```

Amint a [kifejezésekkel szóló fejezetben](#) szerepel, a kifejezés logikai értéke értékelődik ki. Ha **kifejezés TRUE**, akkor a PHP végrehajtja az utasítást; ha **FALSE**, akkor figyelmen kívül hagyja. Arról, hogy mely értékek tekinthetők **FALSE**-nak, a [Logikai értékkel alakítás](#) c. fejezetben olvashatsz.

Az alábbi példa kiírja, hogy **a nagyobb, mint b**, ha **\$a** nagyobb, mint **\$b**:

```
<?php
if ($a > $b)
    echo "a nagyobb, mint b";
?>
```

Gyakran sok utasítást kell feltételhez kötve végrehajtani. Természetesen nem kell minden utasításhoz külön *if*-et írni. Az utasításokat utasításblokkba lehet összefogni. Az alábbi kód például kiírja, hogy **a nagyobb, mint b** ha **\$a** nagyobb, mint **\$b**, és utána hozzárendeli **\$a** értékét **\$b**-hez:

```
<?php
if ($a > $b) {
    echo "a nagyobb, mint b";
    $b = $a;
}
?>
```

A feltételes utasítások vég nélkül további *if* utasításokba ágyazhatók, amely a program különböző részeinek feltételes végrehajtását igen hatékonnyá teszi.

else

Gyakori, hogy egy bizonyos feltétel teljesülése esetén valamilyen utasítást kell végrehajtani, és valamilyen másik utasítást, ha nem teljesül a feltétel. Erre való az *else*. Az *else* kibővíti az *if* utasítást, hogy akkor hajson végre utasítást, amikor az *if* kifejezés **FALSE**-ként értékelődik ki. Az alábbi kód például kiírja, hogy **a nagyobb, mint b** ha **\$a** **\$b**-nél nagyobb, egyébként az **a**

NEM nagyobb, mint b üzenetet írja ki:

```
<?php  
if ($a > $b) {  
    echo "a nagyobb, mint b";  
} else {  
    echo "a NEM nagyobb, mint b";  
}  
?>
```

Az *else* utasítás csak akkor hajtódik végre, ha az *if* kifejezés és az összes *elseif* kifejezés is **FALSE** értékű. Az *elseif*-ről most olvashatsz.

elseif

Az *elseif*, amint azt a neve is sugallja, az *if* és az *else* kombinációja. [perlesek figyelem, itt *elseif*-nek hívják!!!] Az *else*-hez hasonlóan az *if* utasítást terjeszti ki, hogy különböző utasításokat hajtson végre abban az esetben, ha az eredeti *if* kifejezés értéke **FALSE** lenne. Azonban az *else*-sel ellentétben csak akkor hajtra végre az alternatív kódrészt, ha az *elseif* kifejezés **TRUE**. Az alábbi kód például - \$a értékétől függően - üdvözli Menő Manót, és Víz Eleket, vagy kiírja, hogy ismeretlen:

```
<?php  
if ($a == "Menő Manó") {  
    echo "Szervusz Menő Manó! Rég láttalak!";  
} elseif ($a == 'Víz Elek') { #szimpla idézőjel is használható  
    echo "Üdv Víz Elek!";  
} else {  
    echo "Szervusz, idegen. Hát téged mi szél hozott ide?";  
}  
?>
```

Egy *if* kifejezést több *elseif* követhet. Az első olyan *elseif* kifejezés hajtódik végre (ha van), amely értéke **TRUE**. A PHP-ban az 'else if' is (különírva) használható és ugyanúgy fog viselkedni, mint az 'elseif' (egybeírva). A szintaktikai jelentés 'kicsit' eltérő (ha ismered a C-t, nos ez pont úgy működik) de végülis ugyanaz lesz a végeredmény.

Az *elseif* ág csak akkor hajtódik végre, ha az öt megelőző *if* kifejezés, és az összes köztes *elseif* kifejezések **FALSE** értékűek, de az adott *elseif* kifejezése **TRUE**.

Vezérlési szerkezetek alternatív szintaxisa

A PHP bizonyos vezérlési szerkezeteihez egy alternatív szintaxist is nyújt; név szerint: az *if*, *while*, *for*, *foreach*, és *switch* számára. minden esetben az alternatív szintaxisnál a nyitó kapcsos zárójel helyett kettőspontot (:) kell írni, a záró zárójel helyett pedig a vezérlési szerkezetnek megfelelő *endif*, *endwhile*; *endfor*; *endforeach*; vagy *endswitch*; utasításokat értelemszerűen.

```
<?php if ($a == 5): ?>  
A most éppen 5.  
<?php endif; ?>
```

A fenti példában az "A most éppen 5." egy alternatív szintaxisú *if* kifejezésbe van ágyazva. A HTML rész csak akkor íródik ki, ha \$a egyenlő 5-tel.

Az alternatív szintaxis az *else*-re és az *elseif*-re is alkalmazható. Az alábbi példa egy *if* szerkezet, amiben van *elseif* és *else* is alternatív formában:

```
<?php
```

```
if ($a == 0.5):
    echo "a most fél.";
    echo "De vajon kitől?";
elseif ($a == 8):
    echo "Nekem nyolc, hogy mennyi az a.";
    echo "Úgyis megváltoztatom az értékét.";
    $a++;
else:
    echo "Ez így nem vicces, hogy a se nem fél, se nem nyolc";
endif;
?>
```

Lásd még a [while](#), [for](#), és [if](#) szerkezeteket további példák reményében!

while

A *while* ciklusok a PHP legegyszerűbb ciklusai. Éppen úgy viselkednek, mint a C nyelvbeli megfelelőik. A *while* általános szintaxisa:

```
while (kifejezés)
    utasítás
```

A *while* utasítás jelentése egyszerű. Azt mondja a PHP-nek, hogy mindaddig ismételje az utasítás(ok) végrehajtását, amíg a *while* kifejezés **TRUE**. Iterációnak nevezzük azt, amikor a PHP egyszer végrehajtja az utasítást/utasításblokkot egy ciklus részeként. A kifejezés értéke a ciklus kezdetekor értékelődik ki, tehát még ha az utasításblokk belsejében hamissá is válik a feltétel, a blokk végrehajtása akkor sem áll meg, csak az iteráció végén [feltéve ha közben megint meg nem változik a feltétel]. Amikor a *while* kifejezés értéke már az első vizsgálatkor **FALSE**, akkor az utasítás(blokk) egyszer sem kerül végrehajtásra.

Az *if* szerkezethez hasonlóan több utasítást csoportosítani lehet a *while* ciklusban kapcsos zárójelekkel, vagy az alternatív szintaxis használatával:

```
while (kifejezés):
    utasítás
    ...
endwhile;
```

Az alábbi példák ugyanazt csinálják - 1-től 10-ig kiírják a számokat:

```
<?php
/* 1. variáció */

$i = 1;
while ($i <= 10) {
    echo $i++; /* a kiírt érték $i, csak
                  utána növelünk
                  (post-inkrementáció) */
}

/* 2. variáció */

$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
 endwhile;
?>
```

do-while

A *do-while* ciklusok nagyon hasonlóak a *while* ciklusokhoz, a különbség minden össze annyi, hogy a kifejezés igaz volta itt az iteráció végén értékelődik ki, és nem az elején. A fő különbség a hagyományos *while* ciklushoz képest, hogy a *do-while* ciklus első iterációja garantáltan lefut (a kifejezés igazságértékét csak az iteráció végén ellenörzi), amely nem garantált a hagyományos *while* ciklusnál (itt a kifejezés igazságértéke az iteráció kezdetén kerül kiértékelésre, ha értéke kezdetben **FALSE**, akkor a ciklus végrehajtása azonnal befejeződik).

Csak egy szintaxisa van a *do-while* ciklusnak:

```
<?php  
$i = 0;  
do {  
    echo $i;  
} while ($i > 0);  
?>
```

A fenti ciklus pontosan egyszer fut le, mert az első iteráció után, amikor a kifejezés igazságértéke vizsgálatra kerül, kiderül, hogy **FALSE** (\$i nem nagyobb, mint 0) és a ciklus végrehajtása befejeződik.

Haladó C programozók már bizonyára jártasak a *do-while* ciklus másfajta használatában. Például utasításblokk közepén ki lehet lépni a blokkból, ha az utasításblokkot *do-while (0)*, közé tesszük, és *break* utasítást használunk. A következő kód részlet ezt szemlélteti:

```
<?php  
do {  
    if ($i < 5) {  
        echo "i nem elég nagy";  
        break;  
    }  
    $i *= $factor;  
    if ($i < $minimum_limit) {  
        break;  
    }  
    echo " i most jó";  
  
    /* i feldolgozása */  
  
} while (0);  
?>
```

Ne aggódj, ha ezt nem azonnal vagy egyáltalán értette meg. Lehet szkripteket - sőt hatékony szkripteket - írni ennek a lehetőségnek a használata nélkül is.

for

A *for* cílus a legbonyolultabb ciklus a PHP-ben. Éppen úgy viselkedik, mint a C nyelvbeli párja (→ C-ben értőknek tovább gomb). A *for* ciklus szintaxisa:

```
for (kif1; kif2; kif3)  
    utasítás
```

[A fenti *for* szerkezettel megegyező az alábbi, remélhetőleg már ismerős kifejezés:

```
kif1;  
while (kif2) {  
    utasítás;
```

```
    kif3;
}
```

Remélem így már érthetőbb lesz az alábbi magyarázat].

Az első kifejezés (*kif1*) a ciklus kezdetén egyszer kerül vérehajtásra.

Minden iteráció elején *kif2* kiértékelődik. Ha értéke **TRUE**, akkor a ciklus folytatódik, és az *utasításra* kerül a vezérlés. Ha értéke **FALSE**, akkor a ciklus véget ér.

Minden iteráció végén *kif3* is vérehajtásra kerül.

Bármelyik kifejezést el lehet hagyni. Ha *kif2* üres, az azt jelenti, hogy a ciklus a végtelenségig fut [hacsak nem jön a jó tündér *break* utasítás képében...]. (A PHP implicit **TRUE**-nak feltételezi az üres *kif2*-t, mint a C.) Ez nem annyira haszontalan, mint elsőre amennyire elsőnek tűnik, hiszen gyakran fejezheted be a ciklust egy feltételes kifejezésbe ágyazott [*break*](#) kifejezéssel a *for* feltétel kifejezésének kiértékelése helyett.

Nézd az alábbi példákat, mindegyikük kiírja a számokat 1-től 10-ig:

```
<?php
/* téma */

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* 1. variáció */

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* 2. variáció */

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* 3. variáció - Coda :-) */

for ($i = 1; $i <= 10; print $i, $i++);
?>
```

Természetesen "a téma" a legbarátságosabb (vagy esetleg a 3. variáció). Sok helyen hasznos azonban, hogy üres kifejezés is írható *for* ciklusba...

A PHP a *for* ciklus esetén is megengedi az [alternatív szintaxis](#) használatát:

```
for (kif1; kif2; kif3):
    utasítás
    ...
endfor;
```

foreach

A PHP 4-től a Perlhez és más nyelvekhez hasonlóan létezik az ún. *foreach* szerkezet is. Ez jól használható eszközt ad a tömbökön végzett iterációhoz. Más típusú vagy inicializálatlan változóra nem lehet használni, azokra hibát jelez. Két szintaxisa létezik, a második egy apró, de hasznos kiegészítéssel nyújt többet az elsőhöz képest.

```
foreach ($tömb_kifejezés as $ertek)
    utasítás
foreach ($tömb_kifejezés as $kulcs => $ertek)
    utasítás
```

Az első forma végigmegy a *tömb_kifejezés* szolgáltatta tömbön. minden alkalommal az aktuális elem értéke a *\$ertek* változóba kerül, és a belső tömb mutató növelésre kerül. (A következő alkalommal tehát a soron következő elemet fogja venni).

A második forma ugyanezt végzi el, de az aktuális elem kulcsa a *\$kulcs* változóba kerül.

A PHP 5-től kezdődően lehetséges az [objektumokon való iteráció](#) is.

Megjegyzés: Amikor a *foreach* indul, a belső tömb mutató az első elemre áll. Ez azt jelenti, hogy nem kell meghívni a [reset\(\)](#) függvényt egy *foreach* ciklus előtt.

Megjegyzés: Hacsak nem [referenciaként](#) adjuk meg az átnyárazandó tömböt, a *foreach* függvény a megadott tömb egy másolatával dolgozik és nem magával a tömbbel. Ezért az [each\(\)](#)-el ellentétben az eredeti tömb mutatója nem változik meg, és a tömbön végzett módosítások sem kerülnek be az eredeti tömbbe. Azonban az eredeti tömb belső mutatója növelésre kerül a tömb feldolgozása során. Feltéve ha a *foreach* ciklus végig lefut, a tömb belső mutatója a tömb végén lesz.

A PHP 5-ös verziójától könnyedén megváltoztathatod a tömb elemeit, úgy hogy az *\$ertek* előre egy & jelet írsz. Ez ahelyett hogy átmásolná az értéket, [referenciaként](#) adja át

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$ertek) {
    $ertek = $ertek * 2;
}
// az $arr most array(2, 4, 6, 8)
?>
```

Ez csak akkor lehetséges, ha a végigjárt tömb hivatkozható (azaz változó).

Megjegyzés: A *foreach* nem támogatja a @ hiba-elnyelő operátor használatát!

Talán már tudod, hogy az alábbi példák egyenértékűek:

```
<?php
$arr = array("egy", "kettő", "három");
reset($tömb);
while (list($ertek, $ertek) = each($tömb)) {
    echo "Érték: $ertek<br />\n";
}

foreach ($tömb as $ertek) {
    echo "Érték: $ertek<br />\n";
}
?>
```

Az alábbiak is azonos eredményt szolgáltatnak:

```
<?php
$arr = array("egy", "kettő", "három");
reset($tomb);
while (list($kulcs, $ertek) = each($tomb)) {
    echo "Kulcs: $kulcs, Érték: $ertek<br />\n";
}

foreach ($tomb as $kulcs => $ertek) {
    echo "Kulcs: $kulcs, Érték: $ertek<br />\n";
}
?>
```

Néhány további felhasználási példa:

```
<?php
/* első foreach példa: csak érték */

$tomb = array(1, 2, 3, 17);

foreach ($tomb as $ertek) {
    echo "Az aktuális értéke \${$tomb}-nek: $ertek.\n";
}

/* második foreach példa: érték (a kulcs csak illusztráció) */
$tomb = array(1, 2, 3, 17);

$i = 0; /* csak pedagógiai okokból :) */

foreach ($tomb as $ertek) {
    echo "\${$tomb[$i]} => $ertek.\n";
    $i++;
}

/* harmadik foreach példa: kulcs és érték */

$a = array(
    "egy" => 1,
    "kettő" => 2,
    "három" => 3,
    "tizenhét" => 17
);

foreach ($tomb as $kulcs => $ertek) {
    echo "\${$tomb[$kulcs]} => $ertek.\n";
}

/* negyedik foreach példa: többsdimenziós tömb */
$a = array();
$tomb[0][0] = "a";
$tomb[0][1] = "b";
$tomb[1][0] = "y";
$tomb[1][1] = "z";

foreach ($tomb as $belsotomb) {
    foreach ($belsotomb as $ertek) {
        echo "$ertek\n";
    }
}
```

```
/* ötödik foreach példa: dinamikus tömbök */

foreach (array(1, 2, 3, 4, 5) as $ertek) {
    echo "$ertek\n";
}
?>
```

break

A *break* azonnal kilép az aktuális *for*, *foreach*, *while*, *do-while* ciklusból vagy *switch* szerkezetből.

A *break* elfogad egy elhagyható szám paramétert, amely megadja, hogy hány egymásba ágyazott struktúrából kell egyszerre 'kiugrani'.

```
<?php
$tomb = array('egy', 'kettő', 'három', 'négy', 'stop', 'öt');
while (list ($, $ertek) = each($tomb)) {
    if ($ertek == 'stop') {
        break;      /* írhattál volna ide 'break 1;'-et is */
    }
    echo "$ertek<br />\n";
}

/* Az elhagyható paraméter használata */

$i = 0;
while (++$i) {
    switch ($i) {
        case 5:
            echo "5 esetén<br />\n";
            break 1;  /* csak a switch-ből lép ki */
        case 10:
            echo "10 esetén kilépés<br />\n";
            break 2;  /* a switch és a while befejezése */
        default:
            break;
    }
}
?>
```

continue

A *continue* ciklusok belsejében használható arra, hogy átugorjuk az aktuális iteráció hátralevő részét, és a végrehajtást a feltételkiértékeléssel, majd a következő iterációval folytassuk.

Megjegyzés: Megjegyezzük, hogy a PHP-ben a *switch* utasítást ciklus utasításnak tartjuk, a *continue* definíciója miatt.

A *continue* elfogad egy elhagyható szám paramétert, amely megadja, hogy hány egymásba ágyazott struktúrának a hátralévő részét kell átugrani.

```
<?php
while (list ($kulcs, $ertek) = each($tomb)) {
    if (!($kulcs % 2)) { // a páros indexűek kihagyása
        continue;
    }
}
```

```
    valami_paratlan_dolog($ertek);  
}  
  
$i = 0;  
while ($i++ < 5) {  
    echo "Külső while<br />\n";  
    while (1) {  
        echo "&nbsp;&nbsp;Középső while<br />\n";  
        while (1) {  
            echo "&nbsp;&nbsp;Belső while<br />\n";  
            continue 3;  
        }  
        echo "Ezt soha nem fogja kiírni.<br />\n";  
    }  
    echo "Ezt sem...<br />\n";  
}  
?>
```

A pontosvessző elhagyása a *continue* után zavarhoz vezethet. Íme egy példa arra, hogy milyen helyzet nem kellene előforduljon:

```
<?php  
for ($i = 0; $i < 5; ++$i) {  
    if ($i == 2)  
        continue  
    print "$i\n";  
}  
?>
```

Az elvárt eredmény ez lehetne:

```
0  
1  
3  
4
```

de a szkript ezt fogja kiírni:

```
2
```

mert a **print()** által visszatérített érték *int(1)*, ami úgy tűnhet, mintha a fent tárgyalt opcionális paraméter lenne.

switch

A *switch* kifejezés hasonló egy sereg IF kifejezéshez, ahol a kifejezésekben ugyanaz szerepel. [Pont olyan, mint a C-ben, C-s gyakorlattal rendelkezőknek Tovább gomb]. Gyakori, hogy ugyanazt a változót (vagy kifejezést) kell összehasonlítani több különböző értékkel, és más-más kódot végrehajtani a változó (kifejezés) értékétől függően. Pontosan erre való a *switch*.

Megjegyzés: Néhány más nyelvvel ellentétben, a *continue* utasítás használható switch-ben és ugyanúgy működik mint a *break*. Ha van egy switch-et egy cikluson belül, és a ciklusban szeretnél továbblépni, a *continue 2* parancsot kell kiadnod.

Az alábbi két példa két különböző módon írja ki ugyanazt, az egyik egy sor *if* utasításokat használ, a másik pedig a *switch*-et:

Példa 16-1. A *switch* szerkezet

```
<?php  
if ($i == 0) {  
    echo "i most 0";
```

```
    } elseif ($i == 1) {
        echo "i most 1";
    } elseif ($i == 2) {
        echo "i most 2";
    }

switch ($i) {
case 0:
    echo "i most 0";
    break;
case 1:
    echo "i most 1";
    break;
case 2:
    echo "i most 2";
    break;
}
?>
```

Példa 16-2. A *switch* szerkezetben használható stringek

```
<?php
switch ($i) {
case "alma":
    echo "i most: alma";
    break;
case "bigyo":
    echo "i most: bigyo";
    break;
case "torta":
    echo "i most: torta";
    break;
}
?>
```

A hibák elkerülése végett fontos megérteni, hogy hogyan kerül végrehajtásra a *switch* szerkezet. A *switch* vagyis utasításról utasításra hajtódi végre. Nem hajtódi végre semmilyen utasítás, csak akkor, ha egy olyan *case* kifejezést talál a PHP, amely egyezik a *switch* kifejezés értékével. Ezután a PHP addig folytatja az utasítások végrehajtását, amíg el nem éri a *switch* blokk végét, vagy nem találkozik egy *break* utasítással. FONTOS! Ha nem nincs *break* egy *case*-hez tartozó utasítás(sorozat) végén, akkor a PHP végrehajtja a soron következő *case*-hez tartozó utasításokat is!

Például:

```
<?php
switch ($i) {
    case 0:
        echo "i most 0";
    case 1:
        echo "i most 1";
    case 2:
        echo "i most 2";
}
?>
```

Itt, ha *\$i* értéke 0, akkor a PHP az összes kiíró utasítást végrehajtja! Ha *\$i* értéke 1, akkor a PHP az utolsó két echo-t hajtja végre, és csak ha *\$i* értéke 2, akkor kapod a 'kívánt' eredményt (csak az 'i most 2' íródik ki). Tehát nagyon fontos nem elfelejteni a *break* utasítást (bár bizonyos körülmények között lehet, hogy pont ennek elhagyása a szándékos).

A *switch* kifejezésben a feltétel csak egyszer értékelődik ki és a kapott eredmény lesz összehasonlítva a *case* kifejezések mindegyikével. Ha *elseif* kifejezéset használsz, a kifejezések

újra és újra kiértékelődnek. [és újra és újra be kell gépelni. Ez nem csak fárasztó, de hiba forrása is lehet.] Ha a kifejezés bonyolult, vagy egy ciklus belséjében van, a *switch* a gyorsabb.

Egy eset (case) utasításlistája üres is lehet, így a vezérlés a következő case-címkére adódik.

```
<?php
switch ($i) {
case 0:
case 1:
case 2:
    echo "i 3-nál kisebb, de nem negatív";
    break;
case 3:
    echo "i pont 3";
}
?>
```

Egy különleges eset a *default* [alapértelmezett] címke. Ez a címke bármivel egyezik, amivel a korábbi *case* elemek nem egyeztek. Ennek kell az utolsó elemnek lennie. Például:

```
<?php
switch ($i) {
case 0:
    echo "i most 0";
    break;
case 1:
    echo "i most 1";
    break;
case 2:
    echo "i most 2";
    break;
default:
    echo "i se nem 0, se nem 1, se nem 2";
}
?>
```

A *case* kifejezés tetszőleges kifejezés, aminek egyszerű a típusa, vagyis egész vagy lebegőpontos szám, vagy string. Tömbök és objektumok itt nem használhatók, csak egy-egy elemük ill. változójuk egyszerű típusként.

Az alternatív szintaxis működik a switch-ekkel is. Bővebb információért lásd: [Vezérlési szerkezetek alternatív szintaxisa](#).

```
<?php
switch ($i):
case 0:
    echo "i most 0";
    break;
case 1:
    echo "i most 1";
    break;
case 2:
    echo "i most 2";
    break;
default:
    echo "i se nem 0, se nem 1, se nem 2";
endswitch;
?>
```

declare

A *declare* egy kódblokk számára adott futtatási direktívák beállítását teszi lehetővé. A *declare* szyntaxa hasonló a vezérlési szerkezetekéhez:

```
declare (direktíva)
        utasítás
```

A *direktíva* rész a *declare* blokk működését szabályozza. Jelenleg csak egy direktíva használható, a *ticks*. (Lásd lejjebb a [ticks](#) részleteit)

A *declare* blokk *utasítás* része minden egyszer fut le. Az, hogy miképp, és milyen mellékhatásokkal, a *direktíva* részben megadottaktól függ.

A *declare* konstrukció globális hatáskörben is használható, ilyenkor hatással van minden utána következő kódra.

```
<?php
// ezek ekvivalensek:

// használhatod ezt:
declare(ticks=1) {
    // entire script here
}

// vagy használhatod ezt:
declare(ticks=1);
// entire script here
?>
```

Tick-ek

A tick egy olyan esemény, amely minden N db alacsony szintű utasítás végrehajtásakor bekövetkezik a *declare* blokkban. Az N értéket a *ticks=N* szyntaxossal kell megadni a *declare* blokk *direktíva* részében.

Az egyes tick-ekre bekövetkező esemény(ek) a [register_tick_function\(\)](#) függvényvel állítható(ak) be. Lásd az alábbi példát. Akár több esemény is bekövetkezhet egy tick-re.

Példa 16-3. A PHP kód egy részének időmérése

```
<?php
// Ez a függvény megjegyzi a hívása időpontjait
function idopontok($visszaadni = FALSE)
{
    static $idopontok;

    // Visszaadja a $profile tartalmát, és törli
    if ($visszaadni) {
        $idok = $idopontok;
        unset($idopontok);
        return $idok;
    }

    $idopontok[] = microtime();
}

// A tick kezelő beállítása
register_tick_function("idopontok");
```

```
// Beállítjuk az első időpontot a declare előtt
idopontok();

// A kódblokk futtatása, minden második utasítás egy tick
declare(ticks = 2) {
    for ($x = 1; $x < 50; ++$x) {
        echo similar_text(md5($x), md5($x*$x)), "<br />";
    }
}

// Az időmérő függvény adatainak kiírása
print_r(idopontok(TRUE));
?>
```

A fenti példa a declare blokkban lévő PHP kód sebességét méri, rögzítve minden második alacsonyszintű utasítás végrehajtásának időpontját. Ez az információ alkalmas lehet arra, hogy megtaláld a lassan futó részeket a kódodban. Ezt a hatást másképp is el lehet érni, de tick-eket használva sokkal kényelmesebb és könnyebben megvalósítható megoldást kapsz.

A tick-ek kiválóan alkalmasak hibakeresésre, egyszerű multitasking megvalósítására, háttérben futott I/O-ra, és sok más feladatra.

Lásd még a [register_tick_function\(\)](#) és az [unregister_tick_function\(\)](#) függvényeket!

return

A [return\(\)](#) utasítás függvényen belül használva azonnal befejezi a folyó függvény futását, és a paramétereknél megadott érték szolgáltatja a függvény visszatérési értékét. A [return\(\)](#) az [eval\(\)](#) függvénnel futatott kód vagy a szkript futását is leállítja.

A globális érvényességi körben használva a folyó szkript futását szakítja meg. Ha ez a szkript az [include\(\)](#) vagy a [require\(\)](#) hatására lett futtatva, akkor a vezérlés visszaadódik arra a fájlra, ahol ezek az utasítások szerepelnek, valamint [include\(\)](#) esetén a [return\(\)](#) paramétere lesz az [include\(\)](#) utasítás visszatérési értéke. Ha a [return\(\)](#) a fő szkriptben lett kiadva, akkor befejeződik a szkript futása. Ha ez a [auto_prepend_file](#) vagy [auto_append_file](#) php.ini-ben szereplő fájlok valamelyikében történik akkor, (csak) ezeknek a futása fejeződik be.

További magyarázatért lásd [Visszatérési értékek](#) c. fejezetet!

Megjegyzés: Mivel a [return\(\)](#) nyelvi szerkezet és nem függvény, a paraméterét körülvevő zárójelek csak akkor szükségesek, ha az argumentum kifejezést tartalmaz. Általános szokás, hogy nem használják amikor egy változót térítenek vissza, különösen azért mert a PHP-nek így kevesebb munkát kell végeznie.

Megjegyzés: Return utasításban *soha* ne ír zárójelet a változó köré, ha azt referencia szerint akarod visszaadni, mert úgy nem fog működni. Referencia szerint csak változót adhatsz át, kifejezés eredményét nem. Ha `return ($a);` alakban írod, akkor nem a változót adod vissza, hanem a `($a)` kifejezés értékét (ami természetesen az `$a` változó értéke).

require()

A [require\(\)](#) beilleszti és feldolgozza a megadott fájlt. Ennek részletes mikéntjéről, lásd [include\(\)](#)!

A [require\(\)](#) és az [include\(\)](#) megegyezik egymással a hibakezelését leszámítva. Az [include\(\)](#) nem

fatalis hibát, [figyelmeztetést](#) generál, a [require\(\)](#) viszont [fatalis hibát](#) jelez. Másszóval, ahol az igényelt fájl nemlétekor a futást meg kell szakítani, ajánlott a [require\(\)](#). Az [include\(\)](#) nem így viselkedik, a hibától függetlenül a szkript futtatása folytatódik. Bizonyosodj meg, hogy a [include_path](#) helyesen van beállítva!

Példa 16-4. Egyszerű require() példák

```
<?php  
  
require 'prepend.php';  
  
require $valamifajl;  
  
require ('valamifajl.txt');  
  
?>
```

Lásd az [include\(\)](#) oldalát még több példáért!

Megjegyzés: PHP 4.0.2 előtt, a következők szerint működött. A [require\(\)](#) minden beolvasta a kívánt fájlt, még ha az a [require\(\)](#)-t tartalmazó sorra soha nem is került vezérlés. A feltételes szerkezetek nem befolyásolták a működését. Mégis, ha a [require\(\)](#)-t tartalmazó sorra nem került vezérlés a megadott fájlban lévő kód nem futott le. Ehhez hasonlóan, a ciklusok sem befolyásolták a működését. Habár a fájlban szereplő kód függött az azt körülölelő ciklustól, a [require\(\)](#) maga csak egyszer történt meg.

Megjegyzés: Mivel ez egy nyelvi konstrukció és nem egy függvény, nem hívható meg a [változó változók](#) lehetőség felhasználásának segítségével.

Figyelem

A PHP Windows rendszeren futó verziója a 4.3.0-ásnál régebbi változataiban nem támogatja a távoli állomány elérést e függvény használatakor, még akkor sem, ha az [allow_url_fopen](#) engedélyezett.

Lásd még: [include\(\)](#), [require_once\(\)](#), [include_once\(\)](#), [eval\(\)](#), [file\(\)](#), [readfile\(\)](#) és [virtual\(\)](#) függvényeket valamint az [include_path](#) beállítást!

[include\(\)](#)

Az [include\(\)](#) beilleszti és feldolgozza a megadott fájlt.

Az alábbiak igazak a [require\(\)](#)-ra is. A [require\(\)](#) és az [include\(\)](#) megegyezik egymással a hibakezelését leszámítva. Az [include\(\)](#) nem fatalis hibát, [figyelmeztetést](#) generál, a [require\(\)](#) viszont [fatalis hibát](#) jelez. Magyarán, ahol az igényelt fájl nemlétekor a futást meg kell szakítani, ajánlott a [require\(\)](#). Az [include\(\)](#) nem így viselkedik, a hibától függetlenül a szkript futtatása folytatódik. Bizonyosodj meg, hogy a [include_path](#) helyesen van beállítva! Jegyezd meg, hogy a szintaktikai hiba a [include\(\)](#)-olt fájlból nem okozza a feldolgozás megszakítását a PHP 4.3.5 verziója előtt, ettől a verziótól kezdve viszont igen.

A beillesztendő fájlokat először az aktuális könyvtárhoz viszonyított [include_path](#)-ban keresi, majd az aktuális szkript könyvtárhoz viszonyított [include_path](#)-ban. Például ha az [include_path](#) a .., aktuális könyvtár a /www/, beillesztette az [include/a.php](#)-t amelyben van egy [include "b.php"](#) utasítás, a b.php-t először a /www/-ben keresi, majd a /www/include/-ban. Ha a fájlnév/karakterekkel kezdődik, csak aktuális munkakönyvtárhoz viszonyított [include_path](#)-ban keresi.

A fájl beillesztése során a megadott fájl örökli az [include\(\)](#) helyén érvényes változó hatáskört. Bármely változó, amely azon a ponton elérhető, elérhető a beillesztett fájlban is. Ellenben a beillesztett fájlban szereplő függvény- és osztálydefiníciók, globális láthatósággal rendelkeznek.

Példa 16-5. Egyszerű include() példa

```
valtozok.php
<?php

$szin      = 'zöld';
$gyumolcs = 'alma';

?>

teszt.php
<?php

echo "Egy $szin $gyumolcs"; // Egy
                                include 'valtozok.php';

echo "egy $szin $gyumolcs"; // Egy zöld alma
?>
```

Függvény belsejében a megadott fájlban szereplő kód úgy fog viselkedni, mintha az magában a függvényben szerepelt volna. Ez azt jelenti, hogy a fájl örökli a változók érvényességi körét.

Példa 16-6. Függvényen belüli beillesztés

```
<?php

function ize()
{
    global $szin;

    include 'valtozok.php';

    echo "Egy $szin $gyumolcs";
}

/* valtozok.php az ize() függvény hatóköréébe esik,
 * így a $gyumolcs nem elérhető a főfüggvényen kívül.
 * A $szin igen, mivel globálisnak lett deklarálva. */

ize();                      // Egy zöld alma
echo "Egy $szin $gyumolcs"; // Egy zöld
?>
```

Ha egy fájlt beillesztünk az [include\(\)](#)-dal vagy [require\(\)](#)-ral, akkor a célfájl elején az elemző kilép a PHP módból HTML módba, majd visszaáll PHP módba a fájl végén. Ennek okán bármely beillesztendő fájlban levő PHP kódot közre kell fogni egy [érvényes PHP kezdő- és zárójelöléssel](#).

Ha a "[fopen wrappers](#)" szolgáltatás engedélyezve van (mint például az alapértelmezett konfigurációban), a beillesztendő fájlt megadhatod URL-ként (HTTP-ként vagy más támogatott wrapperként - a protokollok listáját itt találhatod: [M Függelék](#)), ahelyett hogy helyi útvonalaként adnád meg. Ha a célszerver PHP kódként feldolgozza a fájlt, akkor átadhatasz változókat a hívott fájlnak HTTP GET lekérési formában. Ez nem teljesen ugyanaz, mintha a [include\(\)](#)-dal hívott fájl örökölne a helyi változókat, mivel a szkript valójában a távoli szerveren fut le, és a futási eredmény kerül beépítésre a helyi szkriptbe.

Figyelem

A PHP Windows rendszeren futó verziója a 4.3.0-ásnál régebbi változataiban nem támogatja a távoli állomány elérést e függvény használatakor, még akkor sem, ha az [allow_url_fopen](#) engedélyezett.

Példa 16-7. [include\(\)](#) HTTP-n keresztül

```
/* Ezek a példák feltételezik, hogy a szerver be van állítva a .php
 * fájlok feldolgozására és nincs beállítva a .txt fájlok feldolgozására
 * A 'működik' azt jelenti, hogy az $ize és $bigyo változók elérhetőek
 * a hívott fájlban. */

// Nem működik: a file.txt nem kerül feldolgozásra
include ("http://szerver/file.txt?ize=1&bigyo=2");

// Nem működik: egy 'file.php?ize=1&bigyo=2' nevű fájlt keres a helyi gépen
include ("file.php?ize=1&bigyo=2");

// Működik
include ("http://szerver/file.php?ize=1&bigyo=2");

$ize = 1;
$bigyo = 2;
include ("file.txt"); /* Működik */
incluie ("file.php"); /* Működik */
```

A kapcsolódó információkért lásd még: [Távoli fájlok eléréséről szóló részt](#), valamint a [fopen\(\)](#) és a [file\(\)](#) függvényt!

Mivel az [include\(\)](#) és a [require\(\)](#) különleges nyelvi elem, kapcsos zárójelekkel kell közrefogni, ha egy feltételes utasításon belül szerepel.

Példa 16-8. [include\(\)](#) feltételes blokkon belül

```
/* Ez NEM JÓ, és nem a várt eredményt adja */

if ($feltetel)
    include($file);
else
    include($other);

/* Ez a HELYES */

if ($feltetel) {
    include($file);
} else {
    include($other);
}
```

return utasítást lehet elhelyezni egy [include\(\)](#)-olt fájlban annak érdekében, hogy a kiértékelés ott befejeződjön, és visszaadjon egy értéket a hívó szkriptnek. A visszatérési értéket ugyanúgy használhatod, mint egy közönséges függvénynél. Ez azonban nem lehetséges távoli fájlok beillesztésénél, kivéve akkor, ha a távoli fájl kimenete [érvényes kezdő és záró tag-ekkel](#) rendelkezik (mint bármilyen helyi fájl). A szükséges változókat deklarálhatod a tag-ek között, és be lesznek vezetve bármely ponton ahol a fájl beillesztésre került.

Mivel az [include\(\)](#) egy speciális nyelvi konstrukció, nem szükséges bezárójelezni az argumenumát. Vigyázz amikor a visszatérítési értékét hasonlítod.

Példa 16-9. Az include visszatérítési értékének hasonlítása

```
<?php
// nem működik, mivel include('vars.php') == 'OK' -ként értelmezi, azaz include('')
if (include('vars.php') == 'OK') {
```

```
    echo 'OK';
}

// működik
if ((include 'vars.php') == 'OK') {
    echo 'OK';
}
?>
```

Megjegyzés: PHP 3, a return nem jelenhet meg függvény blokkon kívül máshol, amely esetben a függvényből történő visszatérést jelöli.

Példa 16-10. Az [include\(\)](#) és a [return\(\)](#) utasítás

```
return.php
<?php

$var = 'PHP';

return $var;

?>

noreturn.php
<?php

$var = 'PHP';

?>

testreturns.php
<?php

$size = include 'return.php';

echo $size; // kiírja: 'PHP'

$bigyo = include 'noreturn.php';

echo $bigyo; // kiírja: 1

?>
```

\$bigyo értéke 1, mert a beillesztés sikeres volt. Figyeld meg a különbséget a két fenti példa között. Az első a [return\(\)](#) segítségével visszaadott egy értéket, a második nem. Ha nem tudja beilleszteni a fájlt, **FALSE**-t ad vissza és *E_WARNING* szintű hibát ad.

Ha a beillesztett fájlban vannak függvénydefiníciók, azokat használni lehet a fő állományban függetlenül attól, hogy a [return\(\)](#) előtt vagy után vannak. Ha a fájl kétszer kerül beillesztésre, a PHP 5 fatális hibát ad, mivel a függvények már deklarálva voltak. A PHP 4 nem panaszkodik miattuk a [return\(\)](#) után definiált függvények miatt. Ajánlott a [include_once\(\)](#) használata, ahelyett hogy a beillesztett fájlból vizsgáld, hogy már be volt-e illesztve és ez esetben return-al kilépjél.

Egy másik módja a PHP fájlok beemelésének a kimenet elfogása a [kimenet szabályzó függvények](#) és az [include\(\)](#) használatával.

Példa 16-11. PHP file kimenetének stringbe gyűjtése kimenetszabályozás használatával

```
<?php
$string = get_include_contents('somefile.php');

function get_include_contents($filename) {
```

```
if (is_file($filename)) {
    ob_start();
    include $filename;
    $contents = ob_get_contents();
    ob_end_clean();
    return $contents;
}
return false;
}

?>
```

Az automatikus fájlbeillesztéssel kapcsolatban lásd a [auto_prepend_file](#) és a [auto_append_file](#) `php.ini` konfigurációs beállításokat.

Megjegyzés: Mivel ez egy nyelvi konstrukció és nem egy függvény, nem hívható meg a [változó változók](#) lehetőség felhasználásának segítségével.

Lásd még a [require\(\)](#), [require_once\(\)](#), [include_once\(\)](#), [readfile\(\)](#) és [virtual\(\)](#) függvényeket!

[require_once\(\)](#)

Az [require_once\(\)](#) beilleszt és feldolgoz fájlokat a program futása közben. Ez hasonló az [require\(\)](#) működéséhez, azzal a fontos különbséggel, hogy ha a már egyszer beillesztésre került kódot a PHP nem próbálja meg ismét betölteni.

A [require_once\(\)](#) használatos azokban az esetekben, amikor ugyanaz a fájl esetleg többször kerülhet beillesztésre a szkript futása során, de biztosítani kell, hogy ez ténylegesen csak egyszer történjen meg, így megelőzve a függvények újradefiniálását, változók értékének átállítását, stb.

További példákhoz az [require_once\(\)](#) és [include_once\(\)](#) használatához nézd meg a PEAR kódot, ami a legfrissebb PHP disztribúciókban megtalálható.

A visszatérítési értékek ugyanolyanok mint az [include\(\)](#) esetében. Ha a fájl már egyszer be volt illesztve, ez a függvény **TRUE**-t ad vissza.

Megjegyzés: A [require_once\(\)](#) PHP 4.0.1pl2 verzióban került a nyelvbe.

Megjegyzés: Különös figyelemmel kell lenni arra, hogy hogyan viselkedik a [require_once\(\)](#) és a [include_once\(\)](#) olyan operációs rendszereken, ahol az állománynevek nem nagybetűérzékenyek (mint pl. Windows)

Példa 16-12. [require_once\(\)](#) Windows-on nagybetűérzékeny

```
<?php
require_once("a.php"); // a.php kerül beillesztésre
require_once("A.php"); // ismét a.php kerül beillesztésre pl. Windows-on! (csak PHP 4.0.1pl2)
?>
```

Ez a viselkedésmód a PHP 5-ben megváltozott: az útvonalat előbb normalizálja, tehát a C:\PROGRA~1\A.php-t ugyanúgy értelmezi mint a C:\Program Files\A.php-t, és a fájlt csak egyszer require-olja.

Figyelem

A PHP Windows rendszeren futó verziója a 4.3.0-ásnál régebbi változataiban nem támogatja a távoli állomány elérést e függvény használatakor, még akkor sem, ha az [allow_url_fopen](#) engedélyezett.

Lásd még a [include\(\)](#), [require\(\)](#), [include_once\(\)](#), [get_required_files\(\)](#), [get_included_files\(\)](#), [readfile\(\)](#) és [virtual\(\)](#) függvényeket!

[include_once\(\)](#)

Az [include_once\(\)](#) beilleszt és feldolgoz fájlokat a program futása közben. Ez hasonló az [include\(\)](#) működéséhez, azzal a fontos különbséggel, hogy ha a már egyszer beillesztésre került kódot a PHP nem próbálja meg ismét betölteni.

Az [include_once\(\)](#) használatos azokban az esetekben, amikor ugyanaz a fájl esetleg többször kerülhet beillesztésre a szkript futása során, de biztosítani kell, hogy ez ténylegesen csak egyszer történjen meg, így megelőzve a függvények újradefiniálását, változók értékének átállítását, stb.

További példákhoz az [require_once\(\)](#) és [include_once\(\)](#) használatához nézd meg a [PEAR](#) kódöt, ami a legfrissebb PHP disztribúciókban megtalálható.

A visszatérítési értékek ugyanolyanok mint az [include\(\)](#) esetében. Ha a fájl már egyszer be volt illesztve, ez a függvény **TRUE**-t ad vissza.

Megjegyzés: Az [include_once\(\)](#) PHP 4.0.1pl2 verzióban került a nyelvbe.

Megjegyzés: Különös figyelemmel kell lenni arra, hogy hogyan viselkedik a [require_once\(\)](#) és a [include_once\(\)](#) olyan operációs rendszereken, ahol az állománynevek nem nagybetűérzékenyek (mint pl. Windows)

Példa 16-13. [require_once\(\)](#) Windows-on nagybetűérzékeny

```
<?php  
require_once("a.php"); // a.php kerül beillesztésre  
require_once("A.php"); // ismét a.php kerül beillesztésre pl. Windows-on! (csak PHP 4.0.1pl2)
```

Ez a viselkedésmód a PHP 5-ben megváltozott: az útvonalat előbb normalizálja, tehát a C:\PROGRA~1\A.php-t ugyanúgy értelmezi mint a C:\Program Files\A.php-t, és a fájlt csak egyszer include-olja.

Figyelem

A PHP Windows rendszeren futó verziója a 4.3.0-ásnál régebbi változataiban nem támogatja a távoli állomány elérést e függvény használatakor, még akkor sem, ha az [allow_url_fopen](#) engedélyezett.

Lásd még a [include\(\)](#), [require\(\)](#), [require_once\(\)](#), [get_required_files\(\)](#), [get_included_files\(\)](#), [readfile\(\)](#) és [virtual\(\)](#) függvényeket!

17. Fejezet. Függvények

Felhasználó által definiált függvények

Függvényeket a következő szintaxis szerint definiálhatod:

Példa 17-1. Pszeudokód a függvényhasználat bemutatására

```
<?php  
function foo($arg_1, $arg_2, /* ..., */, $arg_n)  
{  
    echo "Példa függvény.\n";  
    return $retval;
```

```
}
```

```
?>
```

Bármely érvényes PHP kód megjelenhet egy függvényen belül, akár még más függvény vagy [osztály](#) definíciók is.

PHP 3-ban a függvényeket definiálni kell, mielőtt hivatkozás történik rájuk (függvényhívás előtt). PHP 4-től nincs ez a megkötés, *kivéve* amikor egy függvény feltételesen van definiálva, mint ahogy az alábbi két példa mutatja.

When a function is defined in a conditional manner such as the two examples shown. Its definition must be processed *prior* to being called.

Példa 17-2. Feltételesen definiált függvények

```
<?php

$izet_definialni = true;

/* Innen nem tudjuk meghívni az ize() függvényt mivel még nem létezik,
   viszont a bigyo() függvényt meghívhatjuk */

bigyo();

if ($izet_definialni) {
    function ize()
    {
        echo "Én nem létezem míg nem a program futása eljut hozzáma.\n";
    }
}

/* Most nyugodtan meghívhatjuk az ize() függvényt,
   abban az esetben, ha az $izet_definialni változó értéke true */

if ($izet_definialni) ize();

function bigyo()
{
    echo "Én létezek már a program futásának megkezdésekor is..\n";
}

?>
```

Példa 17-3. Függvények függvényekben

```
<?php
function ize()
{
    function bigyo()
    {
        echo "Én nem létezek amíg az ize() nincs meghívva.\n";
    }
}

/* Itt nem hívhatjuk meg a bigyo() függvényt
   mivel még nem létezik. */

ize();

/* Itt már meghívhatjuk a bigyo()-t,
   mivel az ize() függvény feldolgozása
   elérhetővé tette. */
```

```
bigyo();  
?>
```

Minden függvény és osztály láthatósága globális: függvényen kívülről is meghívható, akkor is ha függvényen belül definiálták, és fordítva.

A PHP nem támogatja a függvények polimorfizmusát (többalakúságát), a függvényekdefiníciókat nem lehet megszüntetni vagy már definiált függvényeket újradefiniálni.

Megjegyzés: A függvénynevek nem érzékenyek a kis- és nagybetűkre, bár rendszerint jó gyakorlat olyan formában meghívni őket, ahogyan deklaráltuk.

A PHP 3 nem támogatja a változó számú függvényargumentumokat, bár az argumentumok kezdőértéke támogatott. Lásd az [Argumentumok kezdőértéke](#) című részt bővebb információért. A PHP 4 és a későbbi változatok mindenkorral lehetőséget támogatják. Lásd a [Változó számú függvényargumentumok](#) című részt és a [func_num_args\(\)](#), [func_get_arg\(\)](#) és a [func_get_args\(\)](#) függvényeket részletesebb leírásért.

PHP-ben lehetséges rekurzív függvények hívása. Kerüld azonban a 100-200 rekurziós szint elérését, mert ez felemésztheti az egész vermet, így a szkript váratlanul befejezheti a futását.

Példa 17-4. Rekurzív függvények

```
<?php  
function rekurzio($a)  
{  
    if ($a < 20) {  
        echo "$a\n";  
        rekurzio($a + 1);  
    }  
}  
?>
```

Függvényargumentumok

Az információ a függvényekhez az argumentumlistán keresztül jut el, ami vesszővel elválasztott kifejezések listája.

A PHP támogatja az érték szerinti (ez az alapértelmezett) [referenciakénti paraméterátadást](#) is, és az [argumentumok kezdőértékét](#). A változó hosszúságú argumentumlisták csak a PHP 4-ben jelentek meg. Lásd a [változó hosszúságú argumentumlistákat](#) és a [func_num_args\(\)](#), [func_get_arg\(\)](#) és a [func_get_args\(\)](#) függvényeket részletesebb leírásért. PHP 3-ban hasonló hatás érhető el a függvénynek tömb típusú változó paraméterként történő átadásával:

Példa 17-5. Tömb átadása függvények

```
<?php  
function tombot_kezel($input)  
{  
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];  
}  
?>
```

Referencia szerinti argumentumfeltöltés

Alapértelmezésben a függvény paraméterei érték szerint adódnak át (vagyis ha megváltoztatod a változót a függvényen belül, annak a függvényen kívülre nincs hatása). Ha szeretnéd megengedni, hogy a függvény módosítsa az átadott paramétereket, referencia szerint kell átadni azokat.

Ha egy függvényargumentum minden referencia szerint kell átadni, akkor a függvénydefinícióban az argumentum neve előre egy & jelet kell írni.

Példa 17-6. Referencia szerinti paraméterátadás

```
<?php
function fgv_extrakkal(&$string)
{
    $string .= 'és a szükséges plusssz.';
}
$str = 'Ez egy karakterfüzér, ';
fgv_extrakkal($str);
echo $str;      // kiírja, hogy 'Ez egy karakterfüzér, és a szükséges plusssz.'
?>
```

Argumentumok kezdőértékei

Bármely függvény skalár-argumentumainak megadhatasz kezdőértéket a C++ szintaxisnak megfelelően:

Példa 17-7. Alapértelmezett paraméterek használata függvényekben

```
<?php

function kavet_csinal($tipus = "cappuccino")
{
    return "Csinálok egy pohár " . $tipus . "t.\n";
}
echo kavet_csinal();
echo kavet_csinal("espresso");

?>
```

A fenti kód kimenete:

```
Csinálok egy pohár cappucinot.
Csinálok egy pohár espressot.
```

A PHP azt is megengedi, hogy tömböt vagy a speciális NULL típust használd alapértelmezett értékként. Például:

Példa 17-8. Nem-skaláris értékek használata alapértelmezett értékként

```
<?php
function kavet_foz($tipusok = array("cappuccino"), $kaveFozo = NULL)
{
    $eszköz = is_null($kaveFozo) ? "kézzel" : $kaveFozo;
    return "Főzök egy csésze ".join(", ", $tipusok)." $eszköz.\n";
}
echo kavet_foz();
echo kavet_foz(array("cappuccino", "lavazza"), "teafőzővel");
?>
```

A kezdőértéknek konstans kifejezésnek kell lennie, nem lehet változó, objektum vagy függvényhívás.

Figyelj arra, hogy a kezdőértékkel rendelkező argumentumok más argumentumuktól jobbra helyezkedjenek el; különben a dolgok nem úgy mennek majd, ahogy azt várnád [hanem úgy, ahogy írtad :)]. Lásd a következő kódot:

Példa 17-9. Az alapértelmezett függvényargumentumok helytelen használata

```
<?php
function joghurtot_keszit($type = "acidophilus", $flavour)
```

```
{  
    return "Készítek egy köcsög $flavour ízű $type-t.\n";  
}  
  
echo joghurtot_keszit("eper"); // nem úgy működik, mint szeretnéd !?!?  
?>
```

A fenti példa kimenete:

```
Warning: Missing argument 2 in call to joghurtot_keszit() in  
/usr/local/etc/httpd/htdocs/phptest/functest.php on line 41  
Készítek egy köcsög ízű eper-t.
```

Most hasonlítsd össze az alábbival:

Példa 17-10. Az alapértelmezett függvényargumentumok helyes használata

```
<?php  
function joghurtot_keszit($flavour, $type = "acidophilus")  
{  
    return "Készítek egy köcsög $flavour ízű $type-t.\n";  
}  
  
echo joghurtot_keszit("eper"); // ez már jó  
?>
```

A fenti példa kimenete:

```
Készítek egy eper ízű acidophilus-t.
```

Megjegyzés: [Azért vannak kiskapuk... Az argumentumokat adjuk át egy asszociatív tömbben! Az alapértékeket pedig egy másik tömbben. Ezután vegyük végig az alapértékeket tartalmazó tömb elemeit. Ha nem definiáltak a kapott tömbben, akkor írjuk felül az alapértékeket tartalmazó tömb megfelelő elemével. Így bármelyik argumentum elhagyható, sőt még annyit is nyerünk a dolgon, hogy nem kell megjegyeznünk a paraméterek sorrendjét. (forrás: egy bő lére ereszttet amerikai PHP 3 könyv)]

Megjegyzés: A PHP 5-ös változatától kezdődően a referenciaként átadott argumentumokat is el lehet látni alapértékkel.

Változó hosszúságú argumentumlista

A PHP 4 és a későbbi változatok támogatják a változó hosszúságú argumentumlistát a felhasználók által definiált függvényekben. Valóban nagyon egyszerű kezelní ezt a [func_num_args\(\)](#), [func_get_arg\(\)](#) és a [func_get_args\(\)](#) függvényekkel.

Semmi különleges szintaksist nem igényel és az argumentumlista lehet explicit módon adott és viselkedhet úgy is, mint egy normál függvény.

Visszatérési értékek

Az elhagyható return állítást használva adhatnak vissza értéket a függvények. Bármely típus visszaadható, beleértve a listákat és az objektumokat is. A függvény végrehajtása azonnal befejeződik, és a vezérlés visszakerül a függvényhívás utáni pozícióba. További részletes információkért lásd: [return\(\)](#)!

Példa 17-11. A [return\(\)](#) használata

```
<?php
function negyzete($num)
{
    return $num * $num;
}
echo negyzete(4); // kiírja '16'.
?>
```

Megjegyzés: [Ha nincs return, az utolsó kifejezés értékével tér viszszza a függvény]

Több értéket nem tud visszaadni a függvény, de hasonló hatás érhető el ezen többszörös értékek listába szervezésével.

Példa 17-12. Összetett adatok tömbként történő visszatérítése

```
<?php
function kis_szamok()
{
    return array(0, 1, 2);
}
list($nulla, $egy, $ketto) = kis_szamok();
?>
```

Ha a függvénynek referenciával kell visszatérnie, akkor az & referencia operátort kell alkalmaznod a függvény deklarásákor és a a visszatérési érték megadásakor is.

Példa 17-13. Referencia visszatérítése

```
<?php
function &referenciat_ad_vissza()
{
    return &$valtozo;
}

$hivatkozas = &referenciat_ad_vissza();
?>
```

További információkért lásd a [Referenciák](#) fejezetet!

Függvényváltozók

A PHP lehetővé teszi a függvényváltozók használatát. Ha egy változónévöt kerek zárójelek követnek, akkor a PHP megkeresi a változó értékével azonos nevű függvényt, és megpróbálja azt végrehajtani. Ezt többek között visszahívandó (callback) függvények vagy függvénytáblák implementálására használható.

A függvényváltozók nem fognak működni az olyan nyelvi elemekkel, mint például az [echo\(\)](#), [print\(\)](#), [unset\(\)](#), [isset\(\)](#), [empty\(\)](#), [include\(\)](#), [require\(\)](#) és hasonlók. Ahhoz, hogy használhasd ezeket a konstrukciókat függvényváltozóként definiálnod kell saját csomagolófüggvényeket.

Példa 17-14. Függvényváltozó példa

```
<?php
function ize()
{
    echo "Az ize()-ben<br />\n";
}

function bigyo($param = '')
{
    echo "A bigyo()-ban; az argumentum: '$param'.<br />\n";
}
```

```
// Ez az echo-nak a csomagolófüggvénye
function echoit($string)
{
    echo $string;
}

$func = 'ize';
$func();                         // Ez meghívja ize()-t

$func = 'bigyo';
$func('Stex van Boeven');      // Ez meghívja bigyo()-t

$func = 'echoit';
$func('teszt');                 // Ez meghívja az echoit()-ot
?>
```

Egy objektum metódusát is meghívhatod a függvényváltozókat használva.

Példa 17-15. Metódusváltató példa

```
<?php
class Ize
{
    function Valtozo()
    {
        $nev = 'Bigyo';
        $this->$nev(); // Ez meghívja a Bigyo() metódust
    }

    function Bigyo()
    {
        echo "Itt Bigyo!";
    }
}

$size = new Ize();
$fnev = "Valtozo";
$size->$fnev(); // Ez meghívja az $size->Valtozo() -t
?>
```

Lásd még a [call_user_func\(\)](#), [variable variables](#) és [function_exists\(\)](#) függvényeket.

Belső (beépített) függvények

A PHP rengeteg függvényt és konstrukciót biztosít. Ezen kívül vannak függvények, amelyek bizonyos lefordított PHP kiterjesztések igényelnek, különben "undefined function" (definiálatlan függvény) fatális hibát okoznak. Például az [image](#) függvények használatához (mint például a [imagecreatetruecolor\(\)](#)) a PHP GD támogatással kell legyen lefordítva. Vagy a [mysql_connect\(\)](#) használatához a PHP-be bele kell fordítani a [MySQL](#) támogatást. Sok olyan alapvető függvény van, amely minden PHP verzióban megtalálható, mint például a [string](#) és a [változó](#) függvények. A [phpinfo\(\)](#) vagy a [get_loaded_extensions\(\)](#) függvény kiírja mely kiterjesztések vannak betöltve a PHP-be. Sok kiterjesztés van alapértelmezésben engedélyezve. A PHP kézikönyv kiterjesztések szerint van felosztva. Egyéb információért a PHP beállításával kapcsolatban lásd még a [konfiguráció](#), [telepítés](#), és a kiterjesztések különálló fejezeteit.

A [Hogyan értelmezzünk egy függvénydefiníciót \(prototípust\)](#) című fejezetben van tárgyalva hogyan kell értelmezni a függvények kézikönyvben szereplő leírását. Fontos megérteni, hogy egy függvény mit ad vissza vagy hogy egy függvény közvetlenül az átadott értéket módosítja. Például a [str_replace\(\)](#) a módosított karakterláncot adja vissza, miközben a [usort\(\)](#) magán az átadott értéken

dolgozik. A kézikönyvnek minden oldala sajátos információkat is tartalmaz a függvényekről, mint például a paraméterek, viselkedésmód változása, visszatérítési értékek siker és sikertelenség esetén és az információ érvényessége. Ezen fontos (viszont sokszor hajszálnyi) különbségek ismerete döntő jelentőségű lehet helyes PHP kód írásához.

Lásd még: [function_exists\(\)](#), [teljes függvény referencia](#), [get_extension_funcs\(\)](#) és [dl\(\)](#) függvényeket.

18. Fejezet. Osztályok, objektumok (PHP 4-ben)

class

Az osztály (objektumtípus) változók és rajtuk műveletet végző függvények [metódusok] együttese. Osztályt az alábbi szintakszis szerint lehet definiálni:

```
<?php
class Kosar {
    var $dolgok; // A kosárban levő dolgok

    // berak a kosárba $db darabot az $sorsz indexű dologból

    function berak($sorsz, $db) {
        $this->dolgok[$sorsz] += $db;
    }

    // kivesz a kosárba $db darabot az $sorsz indexű dologból

    function kivesz($sorsz, $db) {
        if ($this->dolgok[$sorsz] > $db) {
            $this->dolgok[$sorsz] -= $db;
            return true;
        } elseif ($this->dolgok[$sorsz] == $db) {
            unset($this->dolgok[$sorsz]);
            return true;
        } else {
            return false;
        }
    }
}
?>
```

Ez definiál egy Kosar nevű osztályt, ami a kosárban levő áruk asszociatív tömbjéből áll, és definiál 2 funkciót hozzá, hogy bele lehessen rakni, illetve kivenni a kosárba.

Figyelem

Egy osztálydefiníciót *NEM* szedhetsz szét több fájlra. Szintén *NEM* darabolhatsz egy osztálydefiníciót több PHP blokkra, hacsak nem egy metódus közepén teszed ezt valamelyen oknál fogva. A következő példakód tehát működésképtelen:

```
<?php
class teszt {
<?php
?>
    function teszt() {
        print 'oké';
    }
}
?>
```

Ez viszont rendben van:

```
<?php  
class teszt {  
    function teszt() {  
        ?>  
        <?php  
        print 'oké';  
    }  
}  
?>
```

Az alábbi figyelmeztetések a PHP 4-es verziójára érvényesek.

Figyelem

A *stdClass* név le van foglalva, mivel belsőleg a Zend motor használja. Emiatt nem lehet egy *stdClass* nevű osztályod PHP-ben.

A *_sleep* és *_wakeup* nevek speciálisak a PHP osztályokban. Nem szabad ezeket a neveket használni a metódusokhoz, ha csak nem a hozzájuk tartozó speciális funkcionálitást szeretnéd felülírni. Lásd az alábbi részletes leírást.

A PHP számára lefoglalt az összes *_* karakterekkel kezdődő metódusnév, mint speciális név. Nem szabad *_* karakterekkel kezdődő metódusnevet adni, ha csak nem a dokumentált funkciókat szeretnéd használni.

PHP 4-ben csak állandó értékű *var* inicializáló értékek megengedettek. Ha nem állandó értéket szeretnél beállítani, írnod kell egy inicializáló metódust, ami automatikusan meghívódik, amikor egy objektumpéldány létrejön az osztályból. Az ilyen metódusokat hívjuk konstruktornak (lásd lentebb).

```
class Kosar {  
    /* Egyik alábbi értékkadás sem működik PHP 4-ben */  
    var $mai_datum = date("Y. m. d.");  
    var $nev = $csaladi_nev;  
    var $tulajdonos = 'Ferenc' . 'János';  
    /* Ámde tömböt konstans elemekkel megadhatunk */  
    var $termekek = array("Videó", "TV");  
}  
  
/* Így kell a fenti beállításokat elérni */  
class Kosar {  
    var $mai_datum;  
    var $nev;  
    var $tulajdonos;  
    var $termekek = array("Videó", "TV");  
  
    function Kosar() {  
        $this->mai_nap = date("Y. m. d.");  
        $this->nev = $GLOBALS['csaladi_nev'];  
        /* stb. . . */  
    }  
}  
?>
```

Az osztályok típusok, vagyis az aktuális változók tervrajzai. A kívánt típusú változót a *new* operátorral hozhatod létre.

```
<?php
```

```
$kosar = new Kosar;
$kosar->berak("10", 1);

$masik_kosar = new Kosar;
$masik_kosar->berak("0815", 3);
?>
```

Ez létrehozza a Kosar objektumosztály *\$kosar* és *\$masik_kosar* nevű objektumpéldányait. A *\$kosar* objektum berak() függvényét meghívtuk, hogy a 10-es számú árucikkből rakjon 1 darabot a kosárba. Három darab 0815 számú ternék került a *\$masik_kosar* nevű kosárba.

Mind a *\$kosar*, mind a *\$masik_kosar* objektumoknak megvannak a berak() és kivesz() metódusai, és tulajdonságai. Ezek azonban egymástól független metódusok és tulajdonságok. Az objektumokról hasonlóan gondolkozhatsz, mint a könyvtárakról az állományrendszerben. Lehetséges, hogy két különböző OLVASSEL.TXT állományod van, ha ezek két különböző könyvtárban vannak. Úgy mint a könytárnál, ahol meg kell adnod a teljes elérési utat, hogy egy állományra szeretnél hivatkozni a gyökérkönyvtárban, a teljes metódusnevét meg kell adnod, hogy meg tudd azt hívni. A PHP nyelvben a gyökérkönyvtár analógiája a globális környezet, és az elérési út elválasztója a *->*. Ezért a *\$kosar->dolgok* név és a *\$masik_kosar->dolgok* név két különböző változót ad meg. Figyeld meg, hogy a változót *\$kosar->dolgok* néven kell elérni, és nem *\$kosar->\$dolgok* néven, azaz a PHP változók neveibe csak egy dollárjelet kell tenned.

Egy osztály definiálásakor nem tudhatod, milyen néven lesz majd elérhető az objektumod a PHP programban: a Kosar osztály készítése idején nem volt ismert, hogy később *\$kosar* vagy *\$masik_kosar* néven nevezzük-e majd az objektumpéldányt. Ezért nem használhatod a Kosar osztályban a *\$kosar->dolgok* hivatkozást. De hogy el tudjad érni az osztály saját metódusait és tulajdonságait az objektumpéldány nevétől függetlenül, használhatod a *\$this* kvázi-változót, amit 'a sajátom' vagy 'az aktuális objektumpéldány' értelemben alkalmazhatsz. Ezért a '*\$this->dolgok[\$sorsz] += \$db*' úgy olvasható, hogy 'adj *\$db* darab *\$sorsz* sorszámu terméket a saját dolgok tömbömhöz', vagy 'adj *\$db* darab *\$sorsz* sorszámu terméket az aktuális objektumpéldány dolgok tömbjéhez'.

Megjegyzés: A *\$this* néven futó kvázi-változó statikusan hívott osztálymetódusok esetében nincs jelen. Ez alól kivétel az az eset, amikor statikusan hívjuk meg a metódust egy példányosított objektumon belülről. Ez esetben a *\$this* a meghívó objektumra fog hivatkozni. Ezt mutatja be szemléletesen az alábbi példa:

```
<?php
class A {
    function ize() {
        if (isset($this)) {
            echo '$this definiálva van (' ;
            echo get_class($this) ;
            echo ")\n";
        } else {
            echo "\$this nincs definiálva.\n";
        }
    }
}

class B {
    function bigyo() {
        A::ize();
    }
}
```

```
$a = new A();
$a->ize();
A::ize();
$a = new B();
$b->bigyo();
B::bigyo();
?>
```

A fenti példa a következő kimenetet adja:

```
$this definiálva van (a)
$this nincs definiálva.
$this definiálva van (b)
$this nincs definiálva.
```

Megjegyzés: Van pár hasznos függvény az osztályok és objektumok kezeléséhez. Lásd az [Osztály és objektum függvények](#) című részt.

extends

Gyakori, hogy szeretnél olyan osztályokat kialakítani, amelyek egy már meglévő osztályhoz hasonló tulajdonságokkal és metódusokkal rendelkeznek. Tulajdonképpen jó gyakorlat egy általános osztályt definiálni, amit minden projektedben használhatsz, és ezt az osztályt alakítani az egyes projektek igényeinek megfelelően. Ennek a megvalósítása érdekében az osztályok lehetnek más osztályok kiterjesztései. A kiterjesztett, vagy származtatott osztály minden tulajdonággal és metódussal rendelkezik, ami a kiindulási osztályban megvolt (ezt nevezzük öröklésnek, bár senki sem hal meg a folyamat során). Amit hozzáadsz a kiindulási osztályhoz, azt nevezzük kiterjesztésnek. Nem lehetséges megcsónkítani egy osztályt, azaz megszüntetni egy metódust, vagy tulajdonságot. Egy leszármazott osztály minden pontosan egy alaposztálytól függ, azaz egyidejűleg többszörös leszármaztatás nem támogatott. A kiterjesztés kulcsszava az 'extends'.

```
<?php
class Birtokolt_Kosar extends Kosar {
    var $tulaj;
    function tulajdonos_beallitasa($nev) {
        $this->tulaj = $nev;
    }
}
?>
```

Ez definiál egy Birtokolt_Kosar nevű osztályt, ami a Kosar összes változójával és metódusával rendelkezik, és van egy saját változója, a *\$tulaj*, no meg egy saját metódusa, a tulajdonos_beallitasa(). Az így gázdával ellátott kosarat a hagyományos módon hozhatod létre, és a kosár tulajdonosát is be tudod állítani, le tudod kérdezni [ebben az esetben favágó módszerrel]. A gázdas kosarakon továbbra is lehet használni a Kosar függvényeit:

```
<?php
// Gázdas kosár létrehozása
$gkosar = new Birtokolt_Kosar;
// a tulaj beállítása
$gkosar->tulajdonos_beallitasa("Namilesz Teosztasz");
// a tulajdonos neve
print $gkosar->tulaj;
// (Kosar-ból öröklött funkcionális)
$gkosar->berak("10", 1);
?>
```

Ezt "szülő-gyermek" kapcsolatnak is hívják. Egy osztály akkor válik szülőosztállyá, ha más osztály

létrehozásakor ezt *veszik alapul* az *extends* kulcsszó használatával. Ezzel a kiterjesztéssel definiált osztályt hívjuk gyermek-osztálynak -- ebben a kapcsolatban. Ugyanis ezek után ez a gyermekosztály is lehet más osztályok szülőosztálya, ha belőle is származtatnak egy osztályt.

Megjegyzés: Az osztálydefinícióknak a további felhasználásuk előtt rendelkezésre kell állniuk! Ha a *Gazdas_Kosar*-ra van szükség a *Kosar* kiterjesztéseként, akkor előbb a *Kosar* osztályt kell definálni. Ha egy újabb osztályt származtatunk *Sarga_Gazdas_Kosar* néven a *Gazdas_Kosar* osztályból, akkor a *Gazdas_Kosar* osztálydefiníció előbb kell szerepelnie. Röviden, az osztálydefiníciók sorrendje nagyon fontos. Az osztálydefiníciókat hierarchia legfelső szintjén álló osztállyal kell kezdeni, azzal, amelyik minden más osztálynak az ősének számít (szülő vagy nagy-, déd-, ük-, szépszülő).

Konstruktur

A konstruktorkok az osztályok olyan metódusai, amelyek automatikusan meghívásra kerülnek egy új objektumpéldány *new* kulcsszóval történő létrehozása során. Egy függvény akkor lesz konstruktur, ha a neve megegyezik az hordozó osztály nevével. Ha az osztálynak nincs konstruktora, a szülő osztály konstruktora lesz meghívva, már ha létezik.

```
<?php
class Auto_Kosar extends Kosar {
    function Auto_Kosar () {
        $this->berak("10", 1);
    }
}
?>
```

Ez egy olyan *Auto_Kosar* nevű osztályt [objektumtípus] hoz létre, mint a *Kosar*, csak rendelkezik egy konstruktorkkal, amely inicializálja a kosarat 1 darab "10"-es áruval, valahányszor a *new* operátorral hozzuk létre az objektumot. [de csak akkor!!!] A konstruktorknak is lehet átadni paramétereket, és ezek lehetnek elhagyhatók is, amely még hasznosabbá teszi őket. Ha paraméterek nélkül is használható osztályt szeretnél, állíts be minden paraméternek alapértéket.

```
<?php
class Konstruktoros_Kosar extends Kosar {
    function Konstruktoros_Kosar ($sorsz = "10", $db = 1) {
        $this->berak($sorsz, $db);
    }
}

// Mindig ugyanazt az uncsi dolgot veszi...
$kiindulo_kosar = new Konstruktoros_Kosar;

// Igazi vásárlás
$masik_kosar = new Konstruktoros_kosar ("20", 17);
?>
```

Az @ operátor használatával *elnémíthatók* a konstruktorkban fellépő hibák, használata ilyen esetben: @*new*.

```
<?php
class A {
    function A() {
        echo "Én vagyok az A osztály konstruktora.<br />\n";
    }
}
```

```
function B() {
    echo "Én egy B nevű metódus vagyok az A osztályban.<br />\n";
    echo "Nem vagyok A konstruktora.<br />\n";
}
}

class B extends A {

// Ez meghívja B()-t, mint konstruktort
$b = new B;
```

Az A osztály B() metódusa hirtelen konstruktorrá válik a B osztályban, habár ez soha sem volt cél. A PHP 4 nem foglalkozik azzal, hogy a metódus a B osztályban van-e definiálva, vagy öröklés útján áll rendelkezésre.

Figyelem

A PHP 4 nem hívja meg a szülő osztály konstruktörét automatikusan egy leszármazott osztály definiált konstruktorból. A te feladatod, hogy meghívd a szülő konstruktört, ha szükséges.

A destruktur olyan metódus, ami automatikusan meghívódik, amikor egy objektum megszűnik, akár az [unset\(\)](#) meghívásával, akár a környezete megszűnése miatt. PHP-ben nincsenek destruktorkor. Ennek pótálásaképp a legtöbb esetben jól alkalmazható a [register_shutdown_function\(\)](#) függvény.

Hatókör megadó operátor (::)

Figyelem

Az alábbiak csak PHP 4-től érvényesek.

Időnként hasznos az ősosztályok metódusaira vagy tulajdonságaira hivatkozni, vagy olyan osztálymetódusokat meghívni, amelyek nem példányosított objektumokhoz tartoznak. A :: operátor erre használható.

```
<?php
class A {
    function pelda() {
        echo "Én az eredeti A::pelda() metódus vagyok.<br />\n";
    }
}

class B extends A {
    function pelda() {
        echo "Én a felüldefiniáló B::pelda() metódus vagyok.<br />\n";
        A::example();
    }
}

// nincs semmilyen objektum az A osztályból
// ez azonban ki fogja írni:
// Én az eredeti A::pelda() metódus vagyok.<br />
A::pelda();

// B egy objektuát hozzuk létre
$b = new B;

// ez ki fogja írni:
// Én a felüldefiniáló B::pelda() metódus vagyok.<br />
// Én az eredeti A::pelda() metódus vagyok.<br />
```

```
$b->pelda();  
?>
```

A fenti példa meghívja az A osztály pelda() metódusát, habár nincs konkrét példányunk az A osztályból, tehát ezt nem írhatnánk le az *\$a->pelda()*-hoz hasonlóan. Ehelyett a pelda() egy 'osztálymetódusként' viselkedik, azaz az osztály egy függvényeként, és nem egy példány metódusaként.

Osztálymetódusok léteznek, de osztálytulajdonságok (változók) nem. Mivel a hívás pillanatában semmilyen objektum nem létezik, egy osztálymetódus nem használhat objektum változókat, és egyáltalán nem használhatja a *\$this* speciális referenciát. Egy objektummetódus azonban természetesen dolgozhat globális változókkal és lokális változókkal is.

A fenti példa a B osztályban felüldefiniálja a pelda() metódust. Az A osztálytól örökölt eredeti definíció eltűnik, és többé nem érhető el, ha csak nem az A osztályban megvalósított pelda() függvényre hivatkozol közvetlenül, a :: operátor segítségével. Ennek eléréséhez A::pelda()-t kell használni (ebben az esetben írhatnál parent::pelda()-t is, ahogy az a következő szakaszban olvasható).

Ebben a környezetben van aktuálisan használt objektum, és ennek lehetnek objektum változói (tulajdonságai). Ekképpen ha egy objektummetóduson belül használod ezt az operátort, akkor alkalmazhatod a *\$this*-t, és felhasználhatod az objektum tulajdonságait.

parent

Gyakran van szükség arra, hogy a szülő tulajdonságaira vagy metódusaira hivatkozzunk leszármazott osztályokban. Ez különösen igaz, ha a leszármazott osztály egy finomítása, vagy specializálása az alaposztálynak.

Ahelyett, hogy a szülő osztály nevét megadd minden ilyen meghíváskor (mint a hogy a :: operátor példája mutatta), használhatod a *parent* speciális nevet, ami tulajdonképpen a szülő osztály nevét jelenti, amit az *extends* kulcsszónál megadtál. Ennek a speciális névnek a használatával elkerülőd a szülő osztály nevének ismétlődését. Ha a megvalósítás során a leszármazási fát meg kell változtatni, csak egy helyen, az *extends* kulcsszónál kell átírnod a nevet.

```
<?php  
class A {  
    function pelda() {  
        echo "Én A::pelda() vagyok egyszerű funkcióval.<br />\n";  
    }  
}  
  
class B extends A {  
    function pelda() {  
        echo "Én B::pelda() vagyok több funkcióval.<br />\n";  
        parent::pelda();  
    }  
}  
  
$b = new B;  
  
// Ez a B::pelda() metódust hívja, ami az A::pelda()-t hívja  
$b->pelda();  
?>
```

Objektumok szerIALIZációja, objektumok session-ökben

Megjegyzés: A PHP 3-ban az objektumok elveszítik az osztály-hozzárendelésüket a szerializációs, és deserializációs folyamat során. Az eredmény objektum típusú, de nem tartozik semelyik osztályhoz, és nincs egy metódusa sem, tehát elégé használhatatlan (csupán egy tömb, furcsa szintakszissal).

Figyelem

A következő információk csak a PHP 4-es változatára érvényesek.

A [serialize\(\)](#) egy karaktersorozatot ad vissza, ami az átadott érték byte-sorozatban megadott megfelelője. Az [unserialize\(\)](#) visszaalakít egy ilyen karaktersorozatot az eredeti értékké. A szerializációs folyamat során egy objektum átadásával elmenthetjük az objektum minden tulajdonságát (változóját). A függvények nem kerülnek elmentésre, csak az osztály neve.

Ahhoz, hogy az [unserialize\(\)](#) segítségével vissza lehessen állítani egy objektumot, az objektum osztályának (típusának) már definiálva kell lennie. Ez a következőket jelenti egy példán keresztül megvilágítva: Ha az elso.php oldalon az A osztályú \$a objektumot szerializálod, akkor kapsz egy olyan karaktersorozatot, amely az A osztályra hivatkozik, és tartalmazza az összes \$a-ban lévő változó (tulajdonság) értékét. Ha ezt a karaktersorozatot a masodik.php oldalon objektummá szeretnéd alakítani, újra létrehozva az A osztályú \$a nevű objektumot, akkor az A osztály definíciójának rendelkezésre kell állnia a masodik.php oldalon is. Ez úgy érhető el, hogy az A osztály definícióját egy külső állományban tárolod, és ezt alkalmazod mind az elso.php, mind a masodik.php oldalon.

```
<?php
// aosztaly.inc:

class A {
    var $egy = 1;

    function egyet_mutat() {
        echo $this->egy;
    }
}

// elso.php:
include("aosztaly.inc");

$a = new A;
$s = serialize($a);
// tároljuk az $s-t valahol, ahol masodik.php megtalálja
$fp = fopen("tarolas", "w");
fwrite($fp, $s);
fclose($fp);

// masodik.php:
// ez szükséges, hogy a deserializáció rendben menjen
include("aosztaly.inc");

$s = implode("", @file("tarolas"));
$a = unserialize($s);

// most már használható az egyet_mutat() metódus
$a->egyet_mutat();
?>
```

Ha session-öket alkalmazol, és a [session_register\(\)](#) függvényteljes regisztrálsz objektumokat, ezek az

objektumok automatikusan szerizálódnak minden PHP program futása után, és deszerizálódnak minden további programban. Ez egyszerűen azt jelenti, hogy ezek az objektumok akármelyik oldalon feltűnhetnek, miután a session részévé váltak.

Erősen javasolt, hogy minden regisztrált objektum osztály definícióját töltsd minden oldalon, még akkor is, ha éppen nem használod azokat. Ha ezt nem teszed meg, és egy objektum úgy deszerizálódik, hogy nem áll rendelkezésre az osztály definíciója, el fogja veszteni az osztály hozzárendelését, és az *stdClass* osztály egy példánya lesz, metódusok nélkül, így használhatatlanná válik.

Ezért ha a fenti példában az `$a` a session részévé válik a `session_register("a")` meghívásával, akkor be kell töltened az *aosztaly.inc* külső állományt minden oldalon, nem csak az *elso.php* és *masodik.php* programokban.

A speciális `__sleep` és `__wakeup` metódusok

A `serialize()` ellenőrzi, hogy van-e az osztályodnak `__sleep` nevű metódusa. Ha van, ez lefut a szerializáció előtt. Ez megtisztíthatja az objektumot, és végül egy tömbbel tér vissza, amely tartalmazza az adott objektum ténylegesen szerializálandó tulajdonságainak neveit.

A `__sleep` célja, hogy bezárjon minden adatbázis kapcsolatot, a várakozó adatokat lementse, és hasonló 'tisztító' jellegű tevékenységeket végezzen. Hasznos lehet akkor is, ha nagyon nagy objektumaid vannak, amelyeket külön szeretnél lementeni.

Ezzel szemben az `unserialize()` a speciális `__wakeup` nevű függvényt használja. Ha ez létezik, ez a függvény alkalmazható arra, hogy visszaállítsa az objektum erőforrásait.

A `__wakeup` célja lehet például, hogy visszaállítson egy adatbázis kapcsolatot, ami a szerializáció során elveszett, és hasonló beállítási feladatokat végezzen.

Referenciák a konstruktorkban

Referenciák képzése konstruktorokban problémás helyzetekhez vezethet. Ez a leírás segít a bajok elkerülésében.

```
<?php
class Ize {
    function Ize($nev) {
        // egy referencia létrehozása a globális $globalref változóban
        global $globalref;
        $globalref[] = &$this;
        // a név beállítása a kapott értékre
        $this->nevBeallitas($nev);
        // és kiírás
        $this->nevKiiras();
    }

    function nevKiiras() {
        echo "<br />", $this->nev;
    }

    function nevBeallitas($nev) {
        $this->nev = $nev;
    }
}
?>
```

Nézzük, hogy van-e különbség az `$obj1` és az `$obj2` objektum között. Az előbbi a = másoló operátorral készült, az utóbbi a =& referencia operátorral készült.

```
<?php
$obj1 = new Ize('konstruktorban beállított');
$obj1->nevKiiras();
$globalref[0]->nevKiiras();

/* kimenete:
konstruktorban beállított
konstruktorban beállított
konstruktorban beállított */

$obj2 = & new Ize('konstruktorban beállított');
$obj2->nevKiiras();
$globalref[1]->nevKiiras();

/* kimenete:
konstruktorban beállított
konstruktorban beállított
konstruktorban beállított */
?>
```

Szemmel láthatóan nincs semmi különbség, de valójában egy nagyon fontos különbség van a két forma között: az `$obj1` és `$globalref[0]` _NEM_ referenciák, NEM ugyanaz a két változó. Ez azért történhet így, mert a "new" alapvetően nem referenciával tér vissza, hanem egy másolatot ad.

Megjegyzés: Nincsenek teljesítménybeli problémák a másolatok visszaadásakor, mivel a PHP 4 és újabb verziók referencia számlálást alkalmaznak. Legtöbbször ellenben jobb másolatokkal dolgozni referenciák helyett, mivel a referenciák képzése eltart egy kis ideig, de a másolatok képzése gyakorlatilag nem igényel időt. Ha egyik sem egy nagy tömb, vagy objektum, és a változásokat nem szeretnéd mindegyik példányban egyszerre látni, akkor másolatok használatával jobban jársz.

Hogy bebizonyítsuk, amit fent írtunk, lásd az alábbi kódot:

```
<?php
// Most megváltoztatjuk a nevet. Mit vársz?
// Számíthatsz arra, hogy mind $obj1 és $globalref[0] megváltozik...
$obj1->nevBeallitas('kívülről beállítva');

// mint korábban írtuk, nem ez a helyzet
$obj1->nevKiiras();
$globalref[0]->nevKiiras();

/* kimenet:
konstruktorban beállított
kívülről beállítva */

// lássuk mi a különbség az $obj2 és $globalref[1] esetén
$obj2->nevBeallitas('kívülről beállítva');

// szerencsére ezek nem csak egyenlőek, hanem éppen ugyan az
// a két változó, így $obj2->nev és $globalref[1]->nev ugyanaz
$obj2->nevKiiras();
$globalref[1]->nevKiiras();

/* kimenete:
kívülről beállítva
kívülről beállítva */
```

```
?>
```

Végül még egy utolsó példa, próbáld meg megérteni.

```
<?php
class A {
    function A($i) {
        $this->ertek = $i;
        // próbáld meg kitalálni, miért nem kell itt referencia
        $this->b = new B($this);
    }

    function refLetrehozas() {
        $this->c = new B($this);
    }

    function ertekKiiras() {
        echo "<br />",get_class($this),' osztály: ', $this->value;
    }
}

class B {
    function B(&$a) {
        $this->a = &$a;
    }

    function ertekKiiras() {
        echo "<br />",get_class($this),' osztály: ', $this->a->value;
    }
}

// próbáld meg megérteni, hogy egy egyszerű másolás
// miért okoz nem várt eredményeket a *-al jelölt sorban
$a =& new A(10);
$a->refLetrehozas();

$a->ertekKiiras();
$a->b->ertekKiiras();
$a->c->ertekKiiras();

$a->ertek = 11;

$a->ertekKiiras();
$a->b->ertekKiiras(); // *
$a->c->ertekKiiras();
?>
```

A fenti példa a következő kimenetet adja:

```
A osztály: 10
B osztály: 10
B osztály: 10
A osztály: 11
B osztály: 11
B osztály: 11
```

Objektumok összehasonlítása

PHP 4-ben az objektumok igen egyszerű módon kerülnek összevetésre: Két objektumpéldány megegyezik, ha azonos attribútumaik, értékeik vannak, és ugyanazon osztály megtestesülései.

Hasonló szabályok érvényesek két objektum azonosságának vizsgáltaára az egyezőség operátor (=) használatakor.

Ha a lenti példakódot lefuttatjuk:

Példa 18-1. Példa objektum-összehasonlításra PHP 4-ben

```
<?php
function bool2str($bool) {
    if ($bool === false) {
        return 'HAMIS';
    } else {
        return 'IGAZ';
    }
}

function compareObjects(&$o1, &$o2) {
    echo '$o1 == $o2 : '.bool2str($o1 == $o2)."\n";
    echo '$o1 != $o2 : '.bool2str($o1 != $o2)."\n";
    echo '$o1 === $o2 : '.bool2str($o1 === $o2)."\n";
    echo '$o1 !== $o2 : '.bool2str($o1 !== $o2)."\n";
}

class Flag {
    var $flag;

    function Flag($flag=true) {
        $this->flag = $flag;
    }
}

class SwitchableFlag extends Flag {

    function turnOn() {
        $this->flag = true;
    }

    function turnOff() {
        $this->flag = false;
    }
}

$o = new Flag();
$p = new Flag(false);
$q = new Flag();

$r = new SwitchableFlag();

echo "Azonos paraméterekkel létrehozott példányok összehasonlítása\n";
compareObjects($o, $q);

echo "\nEltérő paraméterekkel létrehozott példányok összehasonlítása\n";
compareObjects($o, $p);

echo "\nEgy szülő és annak gyerek osztályából képzett példányok összehasonlítása\n";
compareObjects($o, $r);
?>
```

A fenti példa a következő kimenetet adja:

```
Azonos paraméterekkel létrehozott példányok összehasonlítása
$o == $q : IGAZ
$o != $q : HAMIS
$o === $q : IGAZ
```

```
o1 !== o2 : HAMIS
```

Eltérő paraméterekkel létrehozott példányok összehasonlítása

```
o1 == o2 : HAMIS
```

```
o1 != o2 : IGAZ
```

```
o1 === o2 : HAMIS
```

```
o1 !=== o2 : IGAZ
```

Egy szülő és annak gyerek osztályából képzett példények összehasonlítása

```
o1 == o2 : HAMIS
```

```
o1 != o2 : IGAZ
```

```
o1 === o2 : HAMIS
```

```
o1 !=== o2 : IGAZ
```

Ez tehát az az elvárt működés, amit a fentebb megfogalmazott szabályok alapján tapasztalunk kell. Csak azon példányok fognak megegyezőnek bizonyulni, amelyeknek azonos értékeik és attribútumai vannak és egyazon osztályból lettek példányosítva.

Akkor is ezek a szabályok vannak érvényben, ha több, egységebe foglalt objektum egyben való vizsgálatáról van szó. A következő példában egy olyan hordozó osztályt használunk, amely egy asszociatív tömbben tárol a fenti példából már ismert **Flag** típusú objektumokat.

Példa 18-2. Vegyített objektumok összehasonlítása PHP 4-ben

```
<?php
class FlagSet {
    var $set;

    function FlagSet($flagArr = array()) {
        $this->set = $flagArr;
    }

    function addFlag($name, $flag) {
        $this->set[$name] = $flag;
    }

    function removeFlag($name) {
        if (array_key_exists($name, $this->set)) {
            unset($this->set[$name]);
        }
    }
}

$u = new FlagSet();
$u->addFlag('flag1', $o);
$u->addFlag('flag2', $p);
$v = new FlagSet(array('flag1'=>$q, 'flag2'=>$p));
$w = new FlagSet(array('flag1'=>$q));

echo "\nVegyes objektumok: u(o,p) és v(q,p)\n";
compareObjects($u, $v);

echo "\nu(o,p) and w(q)\n";
compareObjects($u, $w);
?>
```

A fenti példa a következő kimenetet adja:

```
Vegyes objektumok: u(o,p) és v(q,p)
```

```
o1 == o2 : IGAZ
```

```
o1 != o2 : HAMIS
```

```
o1 === o2 : IGAZ
```

```
o1 !=== o2 : HAMIS
```

```
u(o,p) and w(q)
o1 == o2 : HAMIS
o1 != o2 : IGAZ
o1 === o2 : HAMIS
o1 !== o2 : IGAZ
```

19. Fejezet. Osztályok és objektumok (PHP 5)

Bevezetés

A PHP 5-ben új objektummodell van. A PHP objektumkezelését teljesen újraírtuk a jobb teljesítmény, és a több lehetőség figyelembevételével.

Tipp: You may also want to take a look at the [R Függelék](#).

Az alapok

class

Minden osztálydefiníció a class kulcsszóval kezdődik, amit az osztály neve követ, ami bármi lehet, kivéve [fenntartott](#) szó a PHP-ban. Ez egy pár kapcsos zárójel ({ és }) követ, ami tartalmazza az adattagok és metódusok definícióját. Az álváltozó, *\$this* hozzáférhető amikor a metódust objektumon belül hívtuk meg. *\$this* egy referencia a hívó objektumra (rendszerint az az objektum, amihez a metódus tartozik, de lehet más objektum is, ha a metódust [statikusan](#) hívtuk meg vagy másodlagos objektumból). Ezt mutatják be a következő példák:

Példa 19-1. A *\$this* változó objektum-orientált nyelvben

```
<?php
class A
{
    function foo()
    {
        if (isset($this)) {
            echo '$this definiálva van (' ;
            echo get_class($this);
            echo ")\n";
        } else {
            echo "\$this nincs definiálva.\n";
        }
    }
}

class B
{
    function bar()
    {
        A::foo();
    }
}

$a = new A();
$a->foo();
A::foo();
$b = new B();
$b->bar();
```

```
B:::bar();  
?>
```

A fenti példa a következő kimenetet adja:

```
$this definiálva van (a)  
$this nincs definiálva.  
$this definiálva van (b)  
$this nincs definiálva.
```

Példa 19-2. Egyszerű Osztálydefiníció

```
<?php  
class SimpleClass  
{  
    // adattag deklaráció  
    public $var = 'alapértelmezett érték';  
  
    // metódus deklaráció  
    public function displayVar() {  
        echo $this->var;  
    }  
}  
?>
```

Az alapértelmezett értéknek állandós (konstans) kifejezésnek kell lennie, nem (például) egy változó, osztály adattag vagy függvényhívás.

Példa 19-3. Osztály adattagok alapértelmezett értékei

```
<?php  
class SimpleClass  
{  
    // érvénytelen adattag-deklarációk:  
    public $var1 = 'hello '.'world';  
    public $var2 = <<<EOD  
hello world  
EOD;  
    public $var3 = 1+2;  
    public $var4 = self:::myStaticMethod();  
    public $var5 = $myVar;  
  
    // érvényes adattag-deklarációk:  
    public $var6 = myConstant;  
    public $var7 = self:::classConstant;  
    public $var8 = array(true, false);  
  
}
```

Megjegyzés: Itt található néhány hasznos függvény osztályok és objektumok kezeléséhez. Ehhez nézdd meg a [Osztály/Objektum Függvények](#) részt.

new

Egy osztálypéldány létrehozásához új objektumot kell létrehozni és változóba tárolni. Az objektum minden bekerül a változóba létrehozáskor, kivéve ha az új objektumnak van [konstruktora](#), és ez [kivételt \(exception\)](#) dob vagy hibát okoz. Az osztályokat használat előtt kellene definiálni (és néhány esetben ez egy követelmény).

Példa 19-4. Osztálypéldány létrehozása

```
<?php
$instance = new SimpleClass();
?>
```

Amikor egy előre létrehozott objektumpéldányt tárolunk új változóba, az új változó hozzáfér ugyanahhoz a példányhoz, amit tároltunk. Így viselkedik akkor is, amikor az objetumpéldányt függvénynek adjuk át. Egy, már meglévő objektumnak a másolatát klónozással lehet elkészíteni.

Példa 19-5. Objektumok értékadása

```
<?php
$assigned    = $instance;
$reference   =& $instance;

$instance->var = '$assigned-nek ez az értéke lesz';

$instance = null; // $instance és $reference értéke null lesz

var_dump($instance);
var_dump($reference);
var_dump($assigned);
?>
```

A fenti példa a következő kimenetet adja:

```
NULL
NULL
object(SimpleClass)#1 (1) {
    ["var"]=>
        string(30) "$assigned-nek ez az értéke lesz"
}
```

extends

Egy osztály örökölhet metódusokat és adattagokat már osztályuktól az extends kulcsszót használva az osztálydeklarációban. Nem lehetséges öröklös egyszerre több osztálytól, egy osztály egyszerre csak egy alap osztálytól örökölhet.

Az örökölt metódusok és adattagok felülírhatóak, ha csak nem azok a szülő osztályban a final kulcsszóval vannak definiálva, és ugyanazzal a névvel van definiálva a szülő osztályban. Lehetséges a hozzáférés a felülírt metódusokhoz és adattagokhoz ezeket renenciaként használva a parent::-tal.

Példa 19-6. Egyszerű osztályöröklődés

```
<?php
class ExtendClass extends SimpleClass
{
    // Szülő osztály metódusának újradefiniálása
    function displayVar()
    {
        echo "Öröklő osztály\n";
        parent::displayVar();
    }
}

$extended = new ExtendClass();
$extended->displayVar();
?>
```

A fenti példa a következő kimenetet adja:

```
Öröklő osztály
alapértelmezett érték
```

Automatikusan betöltődő objektumok

Több objektum-orientált programot készítő fejlesztő létrehoz egy külön PHP forrásfájlt osztálydefinícióinként. Egy a legnagyobb bosszankodások közül az, hogy hosszú listát kell írni a fájlok beágyazása miatt minden program elején (egy beágyazás minden egyes osztályra).

A PHP 5-ben, ez többé nem szükséges. Te definiálhatsz egy `__autoload` nevű függvényt ami automatikusan meghívódik minden esetben, amikor egy olyan próbálsz meg osztályt használni, ami még nincs definiálva. Ezt a függvényt meghívva a programmotor egy utolsó esélyt ad az osztály betöltésére, mielőtt a PHP hibát eredményez.

Megjegyzés: Az `__autoload` függvényben dobott kivételeket nem lehet `catch` blokkal elkapni, és mert fatalis hibát eredményez.

Példa 19-7. Autoload példa

Ez a példa megpróbálja betölteni a `MyClass1` és a `MyClass2` osztályokat a `MyClass1.php`, illetve a `MyClass2.php` fájlokóból.

```
<?php
function __autoload($class_name) {
    require_once $class_name . '.php';
}

$obj = new MyClass1();
$obj2 = new MyClass2();
?>
```

Konstruktörök és destruktörök

Konstruktur

`void __construct ([mixed args [, ...]])`

A PHP 5 lehetővé teszi a fejlesztők számára, hogy konstruktur metódust deklaráljanak osztályoknak. Azok az osztályok, melyeknek van konstruktur metódusuk, meghívják ezt a metódust minden egyes újonnan létrehozott objektumon, ezért ez alkalmas bármilyen kezdeti beállításhoz ami az objektum számára szükséges a használata előtt.

Megjegyzés: A szülő konstruktur nem hívódik meg, ha az utód osztály definiál egy konstruktort. A szülő konstruktur futtatásához meg kell hívni a `parent::__construct()` metódust az utód konstrukturban.

Példa 19-8. Új egységes konstruktörök használata

```
<?php
class BaseClass {
    function __construct() {
        print "BaseClass konstruktor\n";
    }
}

class SubClass extends BaseClass {
    function __construct() {
        parent::__construct();
        print "SubClass konstruktor\n";
}
```

```
    }
}

$obj = new BaseClass();
$obj = new SubClass();
?>
```

A visszafelé kompatibilitás érdekében ha a PHP 5 nem talál **construct()** függvényt az addot osztályban, megkeresi a régi stílusú konstruktur függvényt, aminek a neve megegyezik az osztályéval. Hathatósan ez azt jelenti, hogy csak akkor van kompatibilitási probléma, ha az osztálynak van egy metódusa **construct()** néven és másmilyen a szemantikája (más lenne az eredeti célja, nem az, hogy konstruktur legyen).

Destruktor

void destruct(void)

A PHP 5 által bevezetett destruktur hasonló a többi objektum-orientált nyelkekéhez, mint a C++. A destruktur metódus akkor hívódik meg, amint az összes referencia az egyéni objektumhoz vagy amikor az objektum nyíltan megsemmisül.

Példa 19-9. Destruktor példa

```
<?php
class MyDestructableClass {
    function __construct() {
        print "Konstruktor\n";
        $this->name = "MyDestructableClass";
    }

    function __destruct() {
        print $this->name . " megsemmisítése\n";
    }
}

$obj = new MyDestructableClass();
?>
```

A konstruktorkhoz hasonlóan, a szülő destruktort nem hívja meg a motor közvetlenül. A szülő destruktur futtatásához nyíltan meg kell hívnod a **parent::__destruct()** metódust a destrukturban.

Megjegyzés: A destruktur a program leállítása során hívódik meg, tehát a fejlécek már minden el vannak küldve.

Megjegyzés: Kivétel dobása a destrukturból fatális hibát eredményez.

Láthatóság

Egy tulajdonság vagy metódus láthatóságát közvetlenül a deklaráció előtt kell megadni a public, protected vagy privát kulcsszavakkal. A publikusként (public) deklárált elemek bárhonnán elérhetőek. A védetteket (protected) csak az öröklő és szülő osztályokból lehet elérni (és abból az osztályból, amelyik definiálja az elemet). A privát (private) láthatóságúakat csak a definiáló osztály éri el.

Adattagok láthatósága

Az osztály adattagokat publikusként, privátként vagy védettként kell definiálni.

Példa 19-10. Adattag deklaráció

```
<?php
/**
 * MyClass definiálása
 */
class MyClass
{
    public $public = 'Publikus';
    protected $protected = 'Védett';
    private $private = 'Privát';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Működik
echo $obj->protected; // Fatális hiba
echo $obj->private; // Fatális hiba
$obj->printHello(); // Kiírja hogy Publikus, Védett és Privát

/**
 * MyClass2 definiálása
 */
class MyClass2 extends MyClass
{
    // Újradeklarálhatjuk a publikus és a védett adattagot, de a privátot nem
    protected $protected = 'Védett2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj->public; // Működik
echo $obj2->private; // Meghatározatlan
echo $obj2->protected; // Fatális hiba
$obj2->printHello(); // Kiírja hogy Publikus és Védett2, de nem írja ki hogy Privát
?>
```

Megjegyzés: A PHP 4 változó deklarálási metódusa a *var* kulcsszóval még támogatott kompatibilitási okok miatt (a public kulcsszó szinonímájaként). Az 5.1.3 verzió előtti PHP 5 motorok **E_STRICT** típusú hibát generáltak.

Metódus láthatóság

Az osztály metódosuait publikusként, privátként vagy védettként kell definiálni. A láthatóság nélküli metódusok pulikusak lesznek.

Példa 19-11. Method Declaration

```
<?php
/**
 * MyClass definiálása
 */
class MyClass
{
    // A konstruktoroknak pulikusnak kell lenniük
    public function __construct() { }

    // Publikus metódus deklarálása
    public function MyPublic() { }

    // Védett metódus deklarálása
    protected function MyProtected() { }

    // Privát metódus deklarálása
    private function MyPrivate() { }

    // Ez is publikus
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Működik
$myclass->MyProtected(); // Fatális hiba
$myclass->MyPrivate(); // Fatális hiba
$myclass->Foo(); // A publikus, a védett és a privát is működik

/**
 * MyClass2 definiálása
 */
class MyClass2 extends MyClass
{
    // Ez is publikus
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Fatális hiba
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Működik
$myclass2->Foo2(); // A publikus és a védett működik, a privát nem
?>
```

Scope Resolution Operator (::)

The Scope Resolution Operator (also called Paamayim Nekudotayim) or in simpler terms, the double colon, is a token that allows access to [static](#), [constant](#), and overridden members or methods of a class.

When referencing these items from outside the class definition, use the name of the class.

Paamayim Nekudotayim would, at first, seem like a strange choice for naming a double-colon. However, while writing the Zend Engine 0.5 (which powers PHP 3), that's what the Zend team decided to call it. It actually does mean double-colon - in Hebrew!

Példa 19-12. :: from outside the class definition

```
<?php
class MyClass {
    const CONST_VALUE = 'A constant value';
}

echo MyClass::CONST_VALUE;
?>
```

Two special keywords *self* and *parent* are used to access members or methods from inside the class definition.

Példa 19-13. :: from inside the class definition

```
<?php
class OtherClass extends MyClass
{
    public static $my_static = 'static var';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}

OtherClass::doubleColon();
?>
```

When an extending class overrides the parents definition of a method, PHP will not call the parent's method. It's up to the extended class on whether or not the parent's method is called. This also applies to [Constructors and Destructors](#), [Overloading](#), and [Magic](#) method definitions.

Példa 19-14. Calling a parent's method

```
<?php
class MyClass
{
    protected function myFunc() {
        echo "MyClass::myFunc()\n";
    }
}

class OtherClass extends MyClass
{
    // Override parent's definition
    public function myFunc()
    {
        // But still call the parent function
        parent::myFunc();
        echo "OtherClass::myFunc()\n";
    }
}
```

```
}
```

```
$class = new OtherClass();
```

```
$class->myFunc();
```

```
?>
```

A static kulcsszó

A statikus (static) osztály adattagok vagy metódusok elérhetőek bármilyen osztálypéldány segítsége nélkül. Egy statikusként deklarált adattag nem érhető el egy példányosított objektumból (bár egy statikus metódust el lehet érni).

A statikus deklarációk a láthatóság deklarációja után kell lenni. A PHP 4-gyel való kompatibilitás miatt, ha nincs [láthatóság](#) deklaráció, az adattag vagy metódus publikus láthatósággal fog rendelkezni.

Mivel a statikus metódusok meghívhatóak osztálypéldány nélkül, a `$this` által-változó nem érhető el a statikus metóduson belül.

Valójában a statikus metódushívások a fordítás idejében meghatározódnak. Amikor egyértelmű osztálynevet használsz egy, már teljesen azonosított metódus használatánál, nincs semmilyen öröklődési szabály. Ha a hívást *self*-fel végzed, a *self*-et a jelenlegi osztályként fordítja le a motor, ez az az oszálly, amelyikhez a kód tartozik. Itt sincs semmilyen öröklődési szabály.

Statikus tulajdonságok nem érhetők el az objektumon keresztül a `->` operátort használva.

Nem statikus metódusok statikus meghívása esetén a motor E_STRICT szintű hibát generál.

Példa 19-15. Statikus adattag példa

```
<?php
class Foo
{
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}

class Bar extends Foo
{
    public function fooStatic() {
        return parent::$my_static;
    }
}

print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";      // Meghatározatlan addattag: my_static

// $foo::my_static nem lehetséges.

print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

Példa 19-16. Statikus metódus példa

```
<?php
class Foo {
    public static function aStaticMethod() {
        // ...
    }
}

Foo:::aStaticMethod();
?>
```

Osztály konstansok

Lehetséges konstans értékek definiálása amelyeknek értéke állandó, és megváltozhatatlan. A konstansok használata annyiban tér el a rendes változóktól, hogy nem kell a \$ jelet használni deklarálásukhoz vagy elérésükhez. A [statikus](#) adattagokhoz hasonlóan, a konstant értékekhez sem lehet közvetlenül hozzáférni az objektumpéldányon keresztül (*\$object::constant* használata).

Az értéknek konstant kifejezésnek kell lennie, nem (például) egy változó, egy osztály adattag, egy matematikai művelet eredménye vagy függvényhívás.

Példa 19-17. Konstans definiálása és használata

```
<?php
class MyClass
{
    const constant = 'konstans érték';

    function showConstant() {
        echo self::constant . "\n";
    }
}

echo MyClass::constant . "\n";

$class = new MyClass();
$class->showConstant();
// echo $class::constant; nem engedélyezett
?>
```

Elvont osztályok (*abstract*)

A PHP 5 lehetővé teszi az elvont osztályok és metódusok használatát. Nem lehet elvont osztálynak objektumtumpéldányát készíteni. minden osztálynak elvontnak kell lennie, ami legalább egy elvont metódust tartalmaz. Az elvont metódusoknak egyszerű metódusként kell deklarálni, de nem tartalmazhatnak belső kódot.

Amikor egy osztály elvont osztálytól örököl, minden elvont metódust deklarálni kell az osztályban; pótlólag ezeknek a metódusoknak ugyanolyan (vagy gyengébb) [láthatósággal \(visibility\)](#) kell rendelkeznie. Például, ha egy elvont metódus "protected", a függvénynek vagy "protected"-nek, vagy "public"-nak kell lennie.

Példa 19-18. Elvont osztály példa

```
<?php
abstract class AbstractClass
{
    // Kényszerítjük az utód osztályt a deklarálásra
```

```
abstract protected function getValue();
abstract protected function prefixValue($prefix);

// Közös metódus
public function printOut() {
    print $this->getValue() . "\n";
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') ."\n";

$class2 = new ConcreteClass2;
$class2->printOut();
echo $class2->prefixValue('FOO_') ."\n";
?>
```

A fenti példa a következő kimenetet adja:

```
ConcreteClass1
FOO_ConcreteClass1
ConcreteClass2
FOO_ConcreteClass2
```

Öreg kódnak, melynek nincs felhasználó által definált 'abstarct' nevű osztálya vagy függvénye módosítások nélkül futnia kell.

Objektum interfések

Az objektum interfések lehetővé teszik, hogy létrehozz egy kódot mely megszabja, milyen metódusokat kötelező implementálni anélkül, hogy definiáljuk ezeknek a metódusoknak a kezelését.

Az interfésekkel az interface kulcsszóval lehet definiálni, ugyanúgy mint az általános osztályokat, de a metódusok tartalma nélkül.

Minden interfészben definiált metódusnak publikusnak kell lennie, ez a természetes egy interfészben.

implements

Egy interfész megvalósításához az *implements* kezelőt kell használni. minden interfészben szereplő metódust meg kell valósítani az osztályba, különben a program fatális hibát eredményez. Az osztályok több, mint egy interfészt is megvalósíthatnak, ha ezeket vesszővel választjuk el.

Megjegyzés: Egy osztály nem valósíthat meg két interfészt amelyek azonos függvényneveket tárolnak, ugyanis ez kétértelműséget okozhat.

Példák

Példa 19-19. Interface example

```
<?php
// Az "iTemplate" interfész definiálása
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Interfész megvalósítása
// Ez működni fog
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}

// Ez nem működik
// Fatal error: Class BadTemplate contains 1 abstract methods
// and must therefore be declared abstract (iTemplate::getHtml)
// Fatális hiba: A BadTemplate talmazza 1 elvont metódust és ezért ezt definálni kell
class BadTemplate implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
}

?>
```

Lásd még az [instanceof](#) operátort.

Overloading

Both method calls and member accesses can be overloaded via the `__call`, `__get` and `__set` methods. These methods will only be triggered when your object or inherited object doesn't contain the member or method you're trying to access. All overloading methods must not be defined as `static`. In PHP 5.0.x, all overloading methods must be defined as `public`.

Since PHP 5.1.0 it is also possible to overload the `isSet()` and `unset()` functions via the `__isSet` and `__unset` methods respectively.

Member overloading

`void __set (string name, mixed value)`

`mixed __get (string name)`

`bool __isSet (string name)`

`void __unset (string name)`

Class members can be overloaded to run custom code defined in your class by defining these specially named methods. The `$name` parameter used is the name of the variable that should be set or retrieved. The `__set()` method's `$value` parameter specifies the value that the object should set the `$name`.

Példa 19-20. overloading with `__get`, `__set`, `__isSet` and `__unset` example

```
<?php
class Setter
{
    public $n;
    private $x = array("a" => 1, "b" => 2, "c" => 3);

    private function __get($nm)
    {
        echo "Getting [$nm]\n";

        if (isset($this->x[$nm])) {
            $r = $this->x[$nm];
            print "Returning: $r\n";
            return $r;
        } else {
            echo "Nothing!\n";
        }
    }

    private function __set($nm, $val)
    {
        echo "Setting [$nm] to $val\n";

        if (isset($this->x[$nm])) {
            $this->x[$nm] = $val;
            echo "OK!\n";
        } else {
            echo "Not OK!\n";
        }
    }
}
```

```
private function __isset($nm)
{
    echo "Checking if $nm is set\n";
    return isset($this->x[$nm]);
}

private function __unset($nm)
{
    echo "Unsetting $nm\n";
    unset($this->x[$nm]);
}
}

$foo = new Setter();
$foo->n = 1;
$foo->a = 100;
$foo->a++;
$foo->z++;

var_dump(isset($foo->a)); //true
unset($foo->a);
var_dump(isset($foo->a)); //false

// this doesn't pass through the __isset() method
// because 'n' is a public property
var_dump(isset($foo->n));

var_dump($foo);
?>
```

A fenti példa a következő kimenetet adja:

```
Setting [a] to 100
OK!
Getting [a]
Returning: 100
Setting [a] to 101
OK!
Getting [z]
Nothing!
Setting [z] to 1
Not OK!

Checking if a is set
bool(true)
Unsetting a
Checking if a is set
bool(false)
bool(true)

object(Setter)#1 (2) {
    ["n"]=>
        int(1)
    ["x:private"]=>
        array(2) {
            ["b"]=>
                int(2)
            ["c"]=>
                int(3)
        }
}
```

```
    }  
}
```

Method overloading

mixed **__call** (string name, array arguments)

Class methods can be overloaded to run custom code defined in your class by defining this specially named method. The *\$name* parameter used is the name as the function name that was requested to be used. The arguments that were passed in the function will be defined as an array in the *\$arguments* parameter. The value returned from the **__call()** method will be returned to the caller of the method.

Példa 19-21. overloading with __call example

```
<?php  
class Caller  
{  
    private $x = array(1, 2, 3);  
  
    private function __call($m, $a)  
    {  
        print "Method $m called:\n";  
        var_dump($a);  
        return $this->x;  
    }  
}  
  
$foo = new Caller();  
$a = $foo->test(1, "2", 3.4, true);  
var_dump($a);  
?>
```

A fenti példa a következő kimenetet adja:

```
Method test called:  
array(4) {  
    [0]=>  
    int(1)  
    [1]=>  
    string(1) "2"  
    [2]=>  
    float(3.4)  
    [3]=>  
    bool(true)  
}  
array(3) {  
    [0]=>  
    int(1)  
    [1]=>  
    int(2)  
    [2]=>  
    int(3)  
}
```

Object Iteration

PHP 5 provides a way for objects to be defined so it is possible to iterate through a list of items, with, for example a [foreach](#) statement. By default, all [visible](#) properties will be used for the

iteration.

Példa 19-22. Simple Object Iteration

```
<?php
class MyClass
{
    public $var1 = 'value 1';
    public $var2 = 'value 2';
    public $var3 = 'value 3';

    protected $protected = 'protected var';
    private   $private   = 'private var';

    function iterateVisible() {
        echo "MyClass::iterateVisible:\n";
        foreach($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$class = new MyClass();

foreach($class as $key => $value) {
    print "$key => $value\n";
}
echo "\n";

$class->iterateVisible();

?>
```

A fenti példa a következő kimenetet adja:

```
var1 => value 1
var2 => value 2
var3 => value 3

MyClass::iterateVisible:
var1 => value 1
var2 => value 2
var3 => value 3
protected => protected var
private => private var
```

As the output shows, the [foreach](#) iterated through all [visible](#) variables that can be accessed. To take it a step further you can *implement* one of PHP 5's internal [interface](#) named *Iterator*. This allows the object to decide what and how the object will be iterated.

Példa 19-23. Object Iteration implementing Iterator

```
<?php
class MyIterator implements Iterator
{
    private $var = array();

    public function __construct($array)
    {
        if (is_array($array)) {
            $this->var = $array;
        }
    }
}
```

```
public function rewind() {
    echo "rewinding\n";
    reset($this->var);
}

public function current() {
    $var = current($this->var);
    echo "current: $var\n";
    return $var;
}

public function key() {
    $var = key($this->var);
    echo "key: $var\n";
    return $var;
}

public function next() {
    $var = next($this->var);
    echo "next: $var\n";
    return $var;
}

public function valid() {
    $var = $this->current() !== false;
    echo "valid: {$var}\n";
    return $var;
}

$values = array(1,2,3);
$it = new MyIterator($values);

foreach ($it as $a => $b) {
    print "$a: $b\n";
}
?>
```

A fenti példa a következő kimenetet adja:

```
rewinding
current: 1
valid: 1
current: 1
key: 0
0: 1
next: 2
current: 2
valid: 1
current: 2
key: 1
1: 2
next: 3
current: 3
valid: 1
current: 3
key: 2
2: 3
next:
current:
valid:
```

You can also define your class so that it doesn't have to define all the *Iterator* functions by simply implementing the PHP 5 *IteratorAggregate* interface.

Példa 19-24. Object Iteration implementing IteratorAggregate

```
<?php
class MyCollection implements IteratorAggregate
{
    private $items = array();
    private $count = 0;

    // Required definition of interface IteratorAggregate
    public function getIterator() {
        return new MyIterator($this->items);
    }

    public function add($value) {
        $this->items[$this->count++] = $value;
    }
}

$coll = new MyCollection();
$coll->add('value 1');
$coll->add('value 2');
$coll->add('value 3');

foreach ($coll as $key => $val) {
    echo "key/value: [$key -> $val]\n\n";
}
?>
```

A fenti példa a következő kimenetet adja:

```
rewinding
current: value 1
valid: 1
current: value 1
key: 0
key/value: [0 -> value 1]

next: value 2
current: value 2
valid: 1
current: value 2
key: 1
key/value: [1 -> value 2]

next: value 3
current: value 3
valid: 1
current: value 3
key: 2
key/value: [2 -> value 3]

next:
current:
valid:
```

Megjegyzés: For more examples of iterators, see the [SPL Extension](#).

Patterns

Patterns are ways to describe best practices and good designs. They show a flexible solution to common programming problems.

Factory

The Factory pattern allows for the instantiation of objects at runtime. It is called a Factory Pattern since it is responsible for "manufacturing" an object. A Parameterized Factory receives the name of the class to instantiate as argument.

Példa 19-25. Parameterized Factory Method

```
<?php
class Example
{
    // The parameterized factory method
    public static function factory($type)
    {
        if (include_once 'Drivers/' . $type . '.php') {
            $classname = 'Driver_' . $type;
            return new $classname;
        } else {
            throw new Exception ('Driver not found');
        }
    }
?>
```

Defining this method in a class allows drivers to be loaded on the fly. If the *Example* class was a database abstraction class, loading a *MySQL* and *SQLite* driver could be done as follows:

```
<?php
// Load a MySQL Driver
$mysql = Example::factory('MySQL');

// Load a SQLite Driver
$sqlite = Example::factory('SQLite');
?>
```

Singleton

The Singleton pattern applies to situations in which there needs to be a single instance of a class. The most common example of this is a database connection. Implementing this pattern allows a programmer to make this single instance easily accessible by many other objects.

Példa 19-26. Singleton Function

```
<?php
class Example
{
    // Hold an instance of the class
    private static $instance;

    // A private constructor; prevents direct creation of object
    private function __construct()
    {
        echo 'I am constructed';
    }
}
```

```
// The singleton method
public static function singleton()
{
    if (!isset(self::$instance)) {
        $c = __CLASS__;
        self::$instance = new $c;
    }

    return self::$instance;
}

// Example method
public function bark()
{
    echo 'Woof!';
}

// Prevent users to clone the instance
public function __clone()
{
    trigger_error('Clone is not allowed.', E_USER_ERROR);
}

}

?>
```

This allows a single instance of the *Example* class to be retrieved.

```
<?php
// This would fail because the constructor is private
$test = new Example;

// This will always retrieve a single instance of the class
$test = Example::singleton();
$test->bark();

// This will issue an E_USER_ERROR.
$test_clone = clone($test);

?>
```

Magic Methods

The function names `__construct`, `__destruct` (see [Constructors and Destructors](#)), `__call`, `__get`, `__set`, `__isset`, `__unset` (see [Overloading](#)), `__sleep`, `__wakeup`, `__toString`, `__set_state`, `__clone` and `__autoload` are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them.

Figyelem

PHP reserves all function names starting with `__` as magical. It is recommended that you do not use function names with `__` in PHP unless you want some documented magic functionality.

`__sleep` and `__wakeup`

[`serialize\(\)`](#) checks if your class has a function with the magic name `__sleep`. If so, that function is executed prior to any serialization. It can clean up the object and is supposed to return an array with

the names of all variables of that object that should be serialized.

The intended use of `__sleep` is to close any database connections that the object may have, commit pending data or perform similar cleanup tasks. Also, the function is useful if you have very large objects which do not need to be saved completely.

Conversely, `unserialize()` checks for the presence of a function with the magic name `__wakeup`. If present, this function can reconstruct any resources that the object may have.

The intended use of `__wakeup` is to reestablish any database connections that may have been lost during serialization and perform other reinitialization tasks.

Példa 19-27. Sleep and wakeup

```
<?php
class Connection {
    protected $link;
    private $server, $username, $password, $db;

    public function __construct($server, $username, $password, $db)
    {
        $this->server = $server;
        $this->username = $username;
        $this->password = $password;
        $this->db = $db;
        $this->connect();
    }

    private function connect()
    {
        $this->link = mysql_connect($this->server, $this->username, $this->password);
        mysql_select_db($this->db, $this->link);
    }

    public function __sleep()
    {
        mysql_close($this->link);
    }

    public function __wakeup()
    {
        $this->connect();
    }
}
?>
```

`__toString`

The `__toString` method allows a class to decide how it will react when it is converted to a string.

Példa 19-28. Simple example

```
<?php
// Declare a simple class
class TestClass
{
    public $foo;

    public function __construct($foo) {
        $this->foo = $foo;
    }
}
```

```
    public function __toString() {
        return $this->foo;
    }

$class = new TestClass('Hello');
echo $class;
?>
```

A fenti példa a következő kimenetet adja:

```
Hello
```

It is worth noting that before PHP 5.2.0 the `__toString` method was only called when it was directly combined with [echo\(\)](#) or [print\(\)](#).

__set_state

This [static](#) method is called for classes exported by [var_export\(\)](#) since PHP 5.1.0.

The only parameter of this method is an array containing exported properties in the form `array('property' => value, ...)`.

A final kulcsszó

A PHP 5 bevezeti a final kulcsszót, ami megakadályozza ezzel a kulcsszóval definiált metódusok felülírását az öröklő osztályokban. Ha az oszálly a final kulcsszóval van definiálva, nem lehet már egy osztálynak se szülő osztálya.

Példa 19-29. Zárt metódus példa

```
<?php
class BaseClass {
    public function test() {
        echo "BaseClass::test() meghívva\n";
    }

    final public function moreTesting() {
        echo "BaseClass::moreTesting() meghívva\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() meghívva\n";
    }
}
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
// Fatális hiba: Nem lehet felülírni a zárt metódost: BaseClass::moreTesting()
?>
```

Példa 19-30. Zárt osztály példa

```
<?php
final class BaseClass {
    public function test() {
        echo "BaseClass::test() meghívva\n";
    }

    // Itt nem gond, ha a függvényt zártként határozod meg, vagy nem
```

```
final public function moreTesting() {
    echo "BaseClass::moreTesting() meghívva\n";
}

class ChildClass extends BaseClass {
}
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)
// Fatális hiba: A ChildClass osztály nem örökölhet zárt osztálytól (BaseClass)
?>
```

Objektum klónozás

Egy objektum másolatának az elkészítéséhez az összes tulajdonság teljes másolásával nem minden eredményezi a várt viselkedést. Egy jó példa, amikor a konstruktorok másolata szükséges, ha neked van egy objektummod ami GTK ablakot képvisel, és te létre akarsz hozni egy új ablakot ugyanazokkal a tulajdonságokkal hogy az új objektum tárolja a forrását az új ablaknak. Másik példa ha a te objektumod referenciát tárol egy másik objektumhoz, és ha te létre akarsz hozni egy másolat objektumot, amiben ennek a másik objektumnak is külön másolata lesz.

Egy objektum másolatát a `clone` kulcsszó használatával lehet létrehozni (ami meghívja az objektum `__clone()` metódusát, ha lehetséges). Az objektum `__clone()` metódusa nem hívható meg közvetlenül.

```
$copy_of_object = clone $object;
```

Amikor egy objektum klónozott, a PHP 5 létrehoz egy felületes másolatot az objektum tulajdonságairól. Valamennyi tulajdonság, ami referencia más változókhöz, referencia fog maradni. Ha a `__clone()` metódus definiált, az újonnan elkészített objektum `__clone()` metódusa fog meghívódni, hogy engedélyezze bármely szükség esetén a tulajdonságok cseréjét.

Példa 19-31. Objektum klónozása

```
<?php
class SubObject
{
    static $instances = 0;
    public $instance;

    public function __construct() {
        $this->instance = ++self::$instances;
    }

    public function __clone() {
        $this->instance = ++self::$instances;
    }
}

class MyCloneable
{
    public $object1;
    public $object2;

    function __clone()
    {
        // Kényszerítjük a this->object másolatát, különben
        // ugyanarra az objektumra mutatna.
        $this->object1 = clone($this->object1);
    }
}
```

```
}
```

```
$obj = new MyCloneable();
```

```
$obj->object1 = new SubObject();
```

```
$obj->object2 = new SubObject();
```

```
$obj2 = clone $obj;
```

```
print("Eredeti Objektum:\n");
```

```
print_r($obj);
```

```
print("Klónozott Objektum:\n");
```

```
print_r($obj2);
```

```
?>
```

A fenti példa a következő kimenetet adja:

```
Eredeti Objektum:
```

```
MyCloneable Object
```

```
(
```

```
    [object1] => SubObject Object
```

```
        (
```

```
            [instance] => 1
```

```
        )
```

```
    [object2] => SubObject Object
```

```
        (
```

```
            [instance] => 2
```

```
        )
```

```
)
```

```
Klónozott Objektum:
```

```
MyCloneable Object
```

```
(
```

```
    [object1] => SubObject Object
```

```
        (
```

```
            [instance] => 3
```

```
        )
```

```
    [object2] => SubObject Object
```

```
        (
```

```
            [instance] => 2
```

```
        )
```

```
)
```

Comparing objects

In PHP 5, object comparison is more complicated than in PHP 4 and more in accordance to what one will expect from an Object Oriented Language (not that PHP 5 is such a language).

When using the comparison operator (==), object variables are compared in a simple manner, namely: Two object instances are equal if they have the same attributes and values, and are instances of the same class.

On the other hand, when using the identity operator (===), object variables are identical if and only if they refer to the same instance of the same class.

An example will clarify these rules.

Példa 19-32. Example of object comparison in PHP 5

```
<?php
function bool2str($bool)
{
    if ($bool === false) {
        return 'FALSE';
    } else {
        return 'TRUE';
    }
}

function compareObjects(&$o1, &$o2)
{
    echo 'o1 == o2 : ' . bool2str($o1 == $o2) . "\n";
    echo 'o1 != o2 : ' . bool2str($o1 != $o2) . "\n";
    echo 'o1 === o2 : ' . bool2str($o1 === $o2) . "\n";
    echo 'o1 !== o2 : ' . bool2str($o1 !== $o2) . "\n";
}

class Flag
{
    public $flag;

    function Flag($flag = true) {
        $this->flag = $flag;
    }
}

class OtherFlag
{
    public $flag;

    function OtherFlag($flag = true) {
        $this->flag = $flag;
    }
}

$o = new Flag();
$p = new Flag();
$q = $o;
$r = new OtherFlag();

echo "Two instances of the same class\n";
compareObjects($o, $p);

echo "\nTwo references to the same instance\n";
compareObjects($o, $q);

echo "\nInstances of two different classes\n";
compareObjects($o, $r);
?>
```

A fenti példa a következő kimenetet adja:

```
Two instances of the same class
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

```
Two references to the same instance
```

```
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE

Instances of two different classes
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Reflection

Introduction

PHP 5 comes with a complete reflection API that adds the ability to reverse-engineer classes, interfaces, functions and methods as well as extensions. Additionally, the reflection API also offers ways of retrieving doc comments for functions, classes and methods.

The reflection API is an object-oriented extension to the Zend Engine, consisting of the following classes:

```
<?php
class Reflection { }
interface Reflector { }
class ReflectionException extends Exception { }
class ReflectionFunction implements Reflector { }
class ReflectionParameter implements Reflector { }
class ReflectionMethod extends ReflectionFunction { }
class ReflectionClass implements Reflector { }
class ReflectionObject extends ReflectionClass { }
class ReflectionProperty implements Reflector { }
class ReflectionExtension implements Reflector { }
?>
```

Megjegyzés: For details on these classes, have a look at the next chapters.

If we were to execute the code in the example below:

Példa 19-33. Basic usage of the reflection API

```
<?php
Reflection::export(new ReflectionClass('Exception'));
?>
```

A fenti példa a következő kimenetet adja:

```
Class [ <internal> class Exception ] {

    - Constants [0] {
    }

    - Static properties [0] {
    }

    - Static methods [0] {
    }

    - Properties [6] {
        Property [ <default> protected $message ]
        Property [ <default> private $string ]
    }
```

```
Property [ <default> protected $code ]
Property [ <default> protected $file ]
Property [ <default> protected $line ]
Property [ <default> private $trace ]
}

- Methods [9] {
Method [ <internal> final private method __clone ] {
}
Method [ <internal> <ctor> public method __construct ] {
    - Parameters [2] {
        Parameter #0 [ <required> $message ]
        Parameter #1 [ <required> $code ]
    }
}

Method [ <internal> final public method getMessage ] {
}

Method [ <internal> final public method getCode ] {
}

Method [ <internal> final public method getFile ] {
}

Method [ <internal> final public method getLine ] {
}

Method [ <internal> final public method getTrace ] {
}

Method [ <internal> final public method getTraceAsString ] {
}

Method [ <internal> public method __toString ] {
}
}
}
```

ReflectionException

ReflectionException extends the standard [Exception](#) and is thrown by Reflection API. No specific methods or properties are introduced.

ReflectionFunction

The **ReflectionFunction** class lets you reverse-engineer functions.

```
<?php
class ReflectionFunction implements Reflector
{
    final private __clone()
    public object __construct(string name)
    public string __toString()
    public static string export(string name, bool return)
```

```
    public string getName()
    public bool isInternal()
    public bool isUserDefined()
    public string getFileName()
    public int getStartLine()
    public int getEndLine()
    public string getDocComment()
    public array getStaticVariables()
    public mixed invoke(mixed args)
    public mixed invokeArgs(array args)
    public bool returnsReference()
    public ReflectionParameter[] getParameters()
    public int getNumberOfParameters()
    public int getNumberOfRequiredParameters()
}
?>
```

Megjegyzés: `getNumberOfParameters()` and `getNumberOfRequiredParameters()` were added in PHP 5.0.3, while `invokeArgs()` was added in PHP 5.1.0.

To introspect a function, you will first have to create an instance of the **ReflectionFunction** class. You can then call any of the above methods on this instance.

Példa 19-34. Using the **ReflectionFunction** class

```
<?php
/**
 * A simple counter
 *
 * @return      int
 */
function counter()
{
    static $c = 0;
    return $c++;
}

// Create an instance of the Reflection_Function class
$func = new ReflectionFunction('counter');

// Print out basic information
printf(
    "====> The %s function '%s'\n".
    "      declared in %s\n".
    "      lines %d to %d\n",
    $func->isInternal() ? 'internal' : 'user-defined',
    $func->getName(),
    $func->getFileName(),
    $func->getStartLine(),
    $func->getEndline()
);

// Print documentation comment
printf("----> Documentation:\n %s\n", var_export($func->getDocComment(), 1));

// Print static variables if existant
if ($statics = $func->getStaticVariables())
{
    printf("----> Static variables: %s\n", var_export($statics, 1));
}
```

```
// Invoke the function
printf("----> Invokation results in: ");
var_dump($func->invoke());

// you may prefer to use the export() method
echo "\nReflectionFunction::export() results:\n";
echo ReflectionFunction::export('counter');
?>
```

Megjegyzés: The method **invoke()** accepts a variable number of arguments which are passed to the function just as in [call_user_func\(\)](#).

ReflectionParameter

The **ReflectionParameter** class retrieves information about a function's or method's parameters.

```
<?php
class ReflectionParameter implements Reflector
{
    final private __clone()
    public object __construct(string name)
    public string __toString()
    public static string export(mixed function, mixed parameter, bool return)
    public string getName()
    public bool isPassedByReference()
    public ReflectionFunction getDeclaringFunction()
    public ReflectionClass getDeclaringClass()
    public ReflectionClass getClass()
    public bool isArray()
    public bool allowsNull()
    public bool isPassedByReference()
    public bool getPosition()
    public bool isOptional()
    public bool isDefaultValueAvailable()
    public mixed getDefaultValue()
}
?>
```

Megjegyzés: **getDefaultValue()**, **isDefaultValueAvailable()** and **isOptional()** were added in PHP 5.0.3, while **isArray()** was added in PHP 5.1.0. **getDeclaringFunction()** and **getPosition()** were added in PHP 5.1.3.

To introspect function parameters, you will first have to create an instance of the **ReflectionFunction** or **ReflectionMethod** classes and then use their **getParameters()** method to retrieve an array of parameters.

Példa 19-35. Using the ReflectionParameter class

```
<?php
function foo($a, $b, $c) { }
function bar(Exception $a, &$b, $c) { }
function baz(ReflectionFunction $a, $b = 1, $c = null) { }
function abc() { }

// Create an instance of Reflection_Function with the
// parameter given from the command line.
$reflect = new ReflectionFunction($argv[1]);
```

```
echo $reflect;

foreach ($reflect->getParameters() as $i => $param) {
    printf(
        "-- Parameter #%-d: %s {\n".
        "    Class: %s\n".
        "    Allows NULL: %s\n".
        "    Passed to by reference: %s\n".
        "    Is optional?: %s\n".
        "}\n",
        $i,
        $param->getName(),
        var_export($param->getClass(), 1),
        var_export($param->allowsNull(), 1),
        var_export($param->isPassedByReference(), 1),
        $param->isOptional() ? 'yes' : 'no'
    );
}
?>
```

ReflectionClass

The **ReflectionClass** class lets you reverse-engineer classes.

```
<?php
class ReflectionClass implements Reflector
{
    final private __clone()
    public object __construct(string name)
    public string __toString()
    public static string export(mixed class, bool return)
    public string getName()
    public bool isInternal()
    public bool isUserDefined()
    public bool isInstantiable()
    public bool hasConstant(string name)
    public bool hasMethod(string name)
    public bool hasProperty(string name)
    public string getFileName()
    public int getStartLine()
    public int getEndLine()
    public string getDocComment()
    public ReflectionMethod getConstructor()
    public ReflectionMethod getMethod(string name)
    public ReflectionMethod[] getMethods()
    public ReflectionProperty getProperty(string name)
    public ReflectionProperty[] getProperties()
    public array getConstants()
    public mixed getConstant(string name)
    public ReflectionClass[] getInterfaces()
    public bool isInterface()
    public bool isAbstract()
    public bool isFinal()
    public int getModifiers()
    public bool isInstance(stdclass object)
    public stdclass newInstance(mixed args)
    public stdclass newInstanceArgs(array args)
    public ReflectionClass getParentClass()
    public bool isSubclassOf(ReflectionClass class)
```

```
    public array getStaticProperties()
    public mixed getStaticPropertyValue(string name [, mixed default])
    public void setStaticPropertyValue(string name, mixed value)
    public array getDefaultProperties()
    public bool isIterateable()
    public bool implementsInterface(string name)
    public ReflectionExtension getExtension()
    public string getExtensionName()
}
?>
```

Megjegyzés: **hasConstant()**, **hasMethod()**, **hasProperty()**, **getStaticPropertyValue()** and **setStaticPropertyValue()** were added in PHP 5.1.0, while **newInstanceArgs()** was added in PHP 5.1.3.

To introspect a class, you will first have to create an instance of the **ReflectionClass** class. You can then call any of the above methods on this instance.

Példa 19-36. Using the **ReflectionClass** class

```
<?php
interface Serializable
{
    // ...
}

class Object
{
    // ...
}

/**
 * A counter class
 */
class Counter extends Object implements Serializable
{
    const START = 0;
    private static $c = Counter::START;

    /**
     * Invoke counter
     *
     * @access public
     * @return int
     */
    public function count() {
        return self::$c++;
    }
}

// Create an instance of the ReflectionClass class
$class = new ReflectionClass('Counter');

// Print out basic information
printf(
    "====> The %s%s%s %s '%s' [extends %s]\n" .
    "      declared in %s\n" .
    "      lines %d to %d\n" .
    "      having the modifiers %d [%s]\n",
    $class->isInternal() ? 'internal' : 'user-defined',
    $class->isAbstract() ? ' abstract' : '',
```

```
$class->isFinal() ? ' final' : '',
$class->isInterface() ? 'interface' : 'class',
$class->getName(),
var_export($class->getParentClass(), 1),
$class->getFileName(),
$class->getStartLine(),
$class->getEndline(),
$class->getModifiers(),
implode(' ', Reflection::getModifierNames($class->getModifiers())))
);

// Print documentation comment
printf("----> Documentation:\n %s\n", var_export($class->getDocComment(), 1));

// Print which interfaces are implemented by this class
printf("----> Implements:\n %s\n", var_export($class->getInterfaces(), 1));

// Print class constants
printf("----> Constants: %s\n", var_export($class->getConstants(), 1));

// Print class properties
printf("----> Properties: %s\n", var_export($class->getProperties(), 1));

// Print class methods
printf("----> Methods: %s\n", var_export($class->getMethods(), 1));

// If this class is instantiable, create an instance
if ($class->isInstantiable()) {
    $counter = $class->newInstance();

    echo '----> $counter is instance? ';
    echo $class->isInstance($counter) ? 'yes' : 'no';

    echo "\n----> new Object() is instance? ";
    echo $class->isInstance(new Object()) ? 'yes' : 'no';
}
?>
```

Megjegyzés: The method **newInstance()** accepts a variable number of arguments which are passed to the function just as in [call_user_func\(\)](#).

Megjegyzés: `$class = new ReflectionClass('Foo');` `$class->isInstance($arg)` is equivalent to `$arg instanceof Foo` or `is_a($arg, 'Foo')`.

ReflectionObject

The **ReflectionObject** class lets you reverse-engineer objects.

```
<?php
class ReflectionObject extends ReflectionClass
{
    final private __clone()
    public object __construct(mixed object)
    public string __toString()
    public static string export(mixed object, bool return)
}
?>
```

ReflectionMethod

The **ReflectionMethod** class lets you reverse-engineer class methods.

```
<?php
class ReflectionMethod extends ReflectionFunction
{
    public __construct(mixed class, string name)
    public string __toString()
    public static string export(mixed class, string name, bool return)
    public mixed invoke(stdclass object, mixed args)
    public mixed invokeArgs(stdclass object, array args)
    public bool isFinal()
    public bool isAbstract()
    public bool isPublic()
    public bool isPrivate()
    public bool isProtected()
    public bool isStatic()
    public bool isConstructor()
    public bool isDestructor()
    public int getModifiers()
    public ReflectionClass getDeclaringClass()

    // Inherited from ReflectionFunction
    final private __clone()
    public string getName()
    public bool isInternal()
    public bool isUserDefined()
    public string getFileName()
    public int getStartLine()
    public int getEndLine()
    public string getDocComment()
    public array getStaticVariables()
    public bool returnsReference()
    public ReflectionParameter[] getParameters()
    public int getNumberOfParameters()
    public int getNumberOfRequiredParameters()
}
?>
```

To introspect a method, you will first have to create an instance of the **ReflectionMethod** class. You can then call any of the above methods on this instance.

Példa 19-37. Using the **ReflectionMethod** class

```
<?php
class Counter
{
    private static $c = 0;

    /**
     * Increment counter
     *
     * @final
     * @static
     * @access public
     * @return int
     */
    final public static function increment()
    {
        return ++self::$c;
    }
}
```

```
}

// Create an instance of the ReflectionMethod class
$method = new ReflectionMethod('Counter', 'increment');

// Print out basic information
printf(
    "====> The %s%s%s%s%s method '%s' (which is %s)\n" .
    "      declared in %s\n" .
    "      lines %d to %d\n" .
    "      having the modifiers %d[%s]\n",
    $method->isInternal() ? 'internal' : 'user-defined',
    $method->isAbstract() ? ' abstract' : '',
    $method->isFinal() ? ' final' : '',
    $method->isPublic() ? ' public' : '',
    $method->isPrivate() ? ' private' : '',
    $method->isProtected() ? ' protected' : '',
    $method->isStatic() ? ' static' : '',
    $method->getName(),
    $method->isConstructor() ? 'the constructor' : 'a regular method',
    $method->getFileName(),
    $method->getStartLine(),
    $method->getEndline(),
    $method->getModifiers(),
    implode(' ', Reflection::getModifierNames($method->getModifiers())))
);

// Print documentation comment
printf("---> Documentation:\n %s\n", var_export($method->getDocComment(), 1));

// Print static variables if existant
if ($statics= $method->getStaticVariables()) {
    printf("---> Static variables: %s\n", var_export($statics, 1));
}

// Invoke the method
printf("---> Invokation results in: ");
var_dump($method->invoke(NULL));
?>
```

Megjegyzés: Trying to invoke private, protected or abstract methods will result in an exception being thrown from the **invoke()** method.

Megjegyzés: For static methods as seen above, you should pass NULL as the first argument to **invoke()**. For non-static methods, pass an instance of the class.

ReflectionProperty

The **ReflectionProperty** class lets you reverse-engineer class properties.

```
<?php
class ReflectionProperty implements Reflector
{
    final private __clone()
    public __construct(mixed class, string name)
    public string __toString()
    public static string export(mixed class, string name, bool return)
    public string getName()
```

```
    public bool isPublic()
    public bool isPrivate()
    public bool isProtected()
    public bool isStatic()
    public bool isDefault()
    public int getModifiers()
    public mixed getValue(stdclass object)
    public void setValue(stdclass object, mixed value)
    public ReflectionClass getDeclaringClass()
    public string getDocComment()
}
?>
```

Megjegyzés: `getDocComment()` was added in PHP 5.1.0.

To introspect a property, you will first have to create an instance of the **ReflectionProperty** class. You can then call any of the above methods on this instance.

Példa 19-38. Using the **ReflectionProperty** class

```
<?php
class String
{
    public $length = 5;
}

// Create an instance of the ReflectionProperty class
$prop = new ReflectionProperty('String', 'length');

// Print out basic information
printf(
    "====> The%s%s%s property '%s' (which was %s)\n" .
    "      having the modifiers %s\n",
    $prop->isPublic() ? ' public' : '',
    $prop->isPrivate() ? ' private' : '',
    $prop->isProtected() ? ' protected' : '',
    $prop->isStatic() ? ' static' : '',
    $prop->getName(),
    $prop->isDefault() ? 'declared at compile-time' : 'created at run-time',
    var_export(Reflection::getModifierNames($prop->getModifiers()), 1)
);

// Create an instance of String
$obj= new String();

// Get current value
printf("----> Value is: ");
var_dump($prop->getValue($obj));

// Change value
$prop->setValue($obj, 10);
printf("----> Setting value to 10, new value is: ");
var_dump($prop->getValue($obj));

// Dump object
var_dump($obj);
?>
```

Megjegyzés: Trying to get or set private or protected class property's values will result in an exception being thrown.

ReflectionExtension

The **ReflectionExtension** class lets you reverse-engineer extensions. You can retrieve all loaded extensions at runtime using the [get_loaded_extensions\(\)](#).

```
<?php
class ReflectionExtension implements Reflector {
    final private __clone()
    public __construct(string name)
    public string __toString()
    public static string export(string name, bool return)
    public string getName()
    public string getVersion()
    public ReflectionFunction[] getFunctions()
    public array getConstants()
    public array getINIEntries()
    public ReflectionClass[] getClasses()
    public array getClassNames()
}
?>
```

To introspect an extension, you will first have to create an instance of the **ReflectionExtension** class. You can then call any of the above methods on this instance.

Példa 19-39. Using the **ReflectionExtension** class

```
<?php
// Create an instance of the ReflectionProperty class
$ext = new ReflectionExtension('standard');

// Print out basic information
printf(
    "Name      : %s\n".
    "Version   : %s\n".
    "Functions : [%d] %s\n".
    "Constants : [%d] %s\n".
    "INI entries : [%d] %s\n".
    "Classes   : [%d] %s\n",
    $ext->getName(),
    $ext->getVersion() ? $ext->getVersion() : 'NO_VERSION',
    sizeof($ext->getFunctions()),
    var_export($ext->getFunctions(), 1),

    sizeof($ext->getConstants()),
    var_export($ext->getConstants(), 1),

    sizeof($ext->getINIEntries()),
    var_export($ext->getINIEntries(), 1),

    sizeof($ext->getClassNames()),
    var_export($ext->getClassNames(), 1)
);
?>
```

Extending the reflection classes

In case you want to create specialized versions of the built-in classes (say, for creating colorized HTML when being exported, having easy-access member variables instead of methods or having utility methods), you may go ahead and extend them.

Példa 19-40. Extending the built-in classes

```
<?php
/**
 * My Reflection_Method class
 */
class My_Reflection_Method extends ReflectionMethod
{
    public $visibility = '';

    public function __construct($o, $m)
    {
        parent::__construct($o, $m);
        $this->visibility= Reflection::getModifierNames($this->getModifiers());
    }
}

/**
 * Demo class #1
 *
 */
class T {
    protected function x() {}
}

/**
 * Demo class #2
 *
 */
class U extends T {
    function x() {}
}

// Print out information
var_dump(new My_Reflection_Method('U', 'x'));
?>
```

Megjegyzés: Caution: If you're overwriting the constructor, remember to call the parent's constructor before any code you insert. Failing to do so will result in the following: *Fatal error: Internal error: Failed to retrieve the reflection object*

Type Hinting

PHP 5 introduces Type Hinting. Functions are now able to force parameters to be objects (by specifying the name of the class in the function prototype) or arrays (since PHP 5.1).

Példa 19-41. Type Hinting examples

```
<?php
// An example class
class MyClass
{
    /**
     * A test function
     *
     * First parameter must be an object of type OtherClass
     */
    public function test(OtherClass $otherclass) {
        echo $otherclass->var;
    }
}
```

```
/**  
 * Another test function  
 *  
 * First parameter must be an array  
 */  
public function test_array(array $input_array) {  
    print_r($input_array);  
}  
}  
  
// Another example class  
class OtherClass {  
    public $var = 'Hello World';  
}  
?>
```

Failing to satisfy the type hint results in a fatal error.

```
<?php  
// An instance of each class  
$myclass = new MyClass;  
$otherclass = new OtherClass;  
  
// Fatal Error: Argument 1 must be an object of class OtherClass  
$myclass->test('hello');  
  
// Fatal Error: Argument 1 must be an instance of OtherClass  
$foo = new stdClass;  
$myclass->test($foo);  
  
// Fatal Error: Argument 1 must not be null  
$myclass->test(null);  
  
// Works: Prints Hello World  
$myclass->test($otherclass);  
  
// Fatal Error: Argument 1 must be an array  
$myclass->test_array('a string');  
  
// Works: Prints the array  
$myclass->test_array(array('a', 'b', 'c'));  
?>
```

Type hinting also works with functions:

```
<?php  
// An example class  
class MyClass {  
    public $var = 'Hello World';  
}  
  
/**  
 * A test function  
 *  
 * First parameter must be an object of type MyClass  
 */  
function MyFunction (MyClass $foo) {  
    echo $foo->var;  
}  
  
// Works  
$myclass = new MyClass;
```

```
MyFunction($myclass);  
?>
```

Type Hints can only be of the [object](#) and [array](#) (since PHP 5.1) type. Traditional type hinting with [int](#) and [string](#) isn't supported.

20. Fejezet. Kivételek (Exceptions)

A PHP 5 kivétel modellje hasonló más programozási nyelvekhez. Egy kivételt dobhatunk (*throw*) és elkaphatunk (*catch*) a PHP-val. A kódot *try* blokkba kell tenni, hogy megkönnyítsük lehetséges kivételek kezelését. minden *try* blokknak rendelkezni kell legalább egy megfelelő *catch* blokkal. Többszörös *catch* blokkokat is használhatunk, így különböző osztályú kivételeket kaphatunk el. A program (ha nincs kivétel a *try* blokkban, vagy ha a kivétel nem illeszkedik egy *catch* blokkra sem) az utolsó *catch* blokk után folytatódik. Kivételek dobhatóak (*throw*) (vagy újradobhatóak) a *catch* blokkban.

Ha kivételt dobtunk, a blokkban a következő kód már nem fut le, és a PHP megkeresi az első *catch* blokkot, amire illeszkedik a kivétel. Ha a kivételt nem sikerült elkapni, a PHP fatális hibát (Fatal Error) ad ki "*Uncaught Exception ...*" szöveggel, ha csak nincs a [set_exception_handler\(\)](#) beállítva.

Példa 20-1. Kivétel küldése (throw Exception)

```
<?php  
try {  
    $error = 'Mindig küldd el ezt a hibát';  
    throw new Exception($error);  
  
    // A köv. kód már nem fut le.  
    echo 'Sose fut le';  
  
} catch (Exception $e) {  
    echo 'Elkapott kivétel: ', $e->getMessage(), "\n";  
}  
  
// Futás folytatása  
echo 'Hello World';  
?>
```

Kivételek kiterjesztése

A felhasználó is hozhat létre saját kivételeket, amiket a beépített *Exception* osztályra kell kiterjeszteni. A tagok és tuladonságok lejjebb megmutatják, mihez lehet hozzáérni az utód osztályban, amit a beépített *Exception* osztálytól örökölt.

Példa 20-2. A beépített Exception osztály

```
<?php  
class Exception  
{  
    protected $message = 'Unknown exception'; // kivétel üzenet  
    protected $code = 0; // felhasználó által megadott hibakód  
    protected $file; // kivétel forrásának fájlneve  
    protected $line; // kivétel forrásának sora  
  
    function __construct($message = null, $code = 0);  
  
    final function getMessage(); // hibaüzenet megszerzése  
    final function getCode(); // kivétel kódja
```

```
final function getFile();                                // forrás fájl
final function getLine();                               // forrás sor
final function getTrace();                             // backtrace() tömb
final function getTraceAsString();                   // formázott "trace" karakterlánc

/* Felülírható */
function __toString();                                // formázott karakterlánc megjelenítéshe
}
?>
```

Ha az osztály utódja a beépített *Exception* osztálynak, és újradefinálja a [konstruktort](#), erősen ajánlott, hogy meghívja a [parent::__construct\(\)](#) metódust hogy biztosítsa minden hozzáférhető adat helyesen legyen tárolva. A [__toString\(\)](#) metódus úrjaírható, hogy tetszőleges legyen az objektum által prezentált karakterlánc.

Példa 20-3. Exception osztály kiterjesztése

```
<?php
/**
 * Saját kivétel osztály létrehozása
 */
class MyException extends Exception
{
    // Újradefinálás, $message nem opcinális
    public function __construct($message, $code = 0) {
        // Kód...

        // legyünk biztosak benne, hogy minden rendben tárolva
        parent::__construct($message, $code);
    }

    // saját __toString() függvény
    public function __toString() {
        return __CLASS__ . ":" . [$this->code] : {$this->message}\n";
    }

    public function customFunction() {
        echo "Saját függvény egy ilyen típusú kivételek\n";
    }
}

/**
 * Osztály kivétel tesztelésére
 */
class TestException
{
    public $var;

    const THROW_NONE      = 0;
    const THROW_CUSTOM   = 1;
    const THROW_DEFAULT  = 2;

    function __construct($avalue = self::THROW_NONE) {

        switch ($avalue) {
            case self::THROW_CUSTOM:
                // throw custom exception
                throw new MyException('1 is an invalid parameter', 5);
                break;
        }
    }
}
```

```
        case self::THROW_DEFAULT:
            // throw default one.
            throw new Exception('2 isn\'t allowed as a parameter', 6);
            break;

        default:
            // No exception, object will be created.
            $this->var = $avalue;
            break;
    }
}

// 1. Példa
try {
    $o = new TestException(TestException::THROW_CUSTOM);
} catch (MyException $e) {          // Elkapva
    echo "Caught my exception\n", $e;
    $e->customFunction();
} catch (Exception $e) {           // Kihagyva
    echo "Caught Default Exception\n", $e;
}

// Futás folytatása
var_dump($o);
echo "\n\n";

// 2. Példa
try {
    $o = new TestException(TestException::THROW_DEFAULT);
} catch (MyException $e) {          // Nem ez a típus
    echo "Caught my exception\n", $e;
    $e->customFunction();
} catch (Exception $e) {           // Elkapva
    echo "Caught Default Exception\n", $e;
}

// Futás folytatása
var_dump($o);
echo "\n\n";

// 3. Példa
try {
    $o = new TestException(TestException::THROW_CUSTOM);
} catch (Exception $e) {           // Elkapva
    echo "Default Exception caught\n", $e;
}

// Futás folytatása
var_dump($o);
echo "\n\n";

// 4. Példa
try {
    $o = new TestException();
} catch (Exception $e) {           // Ez nem kivétel
    echo "Default Exception caught\n", $e;
}
```

```
}
```

```
// Futás folytatása
var_dump($o);
echo "\n\n";
?>
```

21. Fejezet. Referenciák

Mik a referenciák

A referenciák lehetőséget adnak PHP-ben azonos változó tartalom elérésére különböző nevek alatt. Ezek szimbólumtábla bejegyzések, nem olyanok, mint a C nyelv mutatói. PHP-ben a változók neve és tartalma két különböző dolog, tehát ugyanaz a tartalom kaphat különböző neveket. A legjobb hasonlat talán a UNIX állománynevek és állományok rendszere. A változóneveket könyvtár bejegyzésekkel foghatod fel, a változók tartalmát állományokként. A referenciák olyanok, mint UNIXban a hardlinkek.

Mit lehet referenciaikkal tenni

A PHP referenciák lehetőséget adnak arra, hogy egy értékhez két nevet lehessen rendelni. Ez azt jelenti, hogy a következő programban:

```
<?php
$a =& $b;
?>
```

Az *\$a* és *\$b* nevek ugyanarra az értékre hivatkoznak.

Megjegyzés: Az *\$a* és a *\$b* nevek teljesen egyenrangúak. Nem arról van szó, hogy az *\$a* a *\$b*-re mutat, vagy fordítva, hanem arról, hogy az *\$a* és a *\$b* név ugyanannak az értéknek két elnevezése.

Megjegyzés: Ha egy referenciákat tartalmazó tömböt másolunk, a referenciák másolódnak, nem az értékek. Ugyanez igaz az érték szerinti paraméterátadásnál is.

Ugyanez a forma használható az olyan függvényeknél, amelyek referenciát adnak vissza, vagy a *new* operátor használatakor (a PHP 4.0.4 és későbbi verziókban):

```
<?php
$obj =& new valamilyen_osztaly();
$size =& valozo_kereses($valami);
?>
```

Megjegyzés: Ha nem használd az *&* -t, akkor az osztálypéldány másoláta adódik át. A *\$this* objektumon belüli használatával ugyanazon az objektumpéldányon dolgozol. Ha az értékadás során az *&* -t elhagyod, akkor az objektumról másolat készül és a *\$this* már ezen a másolaton fog dolgozni. Van, amikor ez nem kívánatos, mivel általában egy példányon szeretnénk dolgozni a jobb memóriahasználat és teljesítmény érdekében.

Az *@* operátort nem lehet együtt használni a *&new* utasítással, amivel el lehetne nyomni a konstruktörben fellépő hibákról történő jelzéseket. Ez a jelenlegi Zend motor egyik korlátja, és ezért fordítási hibát (parse error-t) fog jelezni, ha ilyet talál.

Figyelem

Ha egy függvényen belül egy *global* jelzővel ellátott változóhoz referenciát rendelsz hozzá, az a referencia csak a függvény belsejében lesz látható. Ennek elkerülésére használ a *\$GLOBALS* tömböt.

Példa 21-1. Globális változók hivatkozása függvény belsejében

```
<?php
$var1 = "Példa változó";
$var2 = "";

function globalis_referenciak($globals_hasznalata)
{
    global $var1, $var2;
    if (! $globals_hasznalata) {
        $var2 =& $var1; // csak a függvény belsejében látható
    } else {
        $GLOBALS["var2"] =& $var1; // globális hatásörben látható
    }
}

globalis_referenciak(false);
echo "var2 értéke '$var2'\n"; // var2 értéke ''
globalis_referenciak(true);
echo "var2 értéke '$var2'\n"; // var2 értéke 'Példa változó'
?>
```

Gondolj a *global \$var*;-ra úgy, mint a *\$var =& \$GLOBALS['var']*; egy rövidítése. Ezért egy másik referencia hozzárendelése a *\$var*-hoz csak a helyi változó referenciáját változtatja meg.

Megjegyzés: Ha egy referenciákkal rendelkező változót adsz meg a [foreach](#) utasításban, a referencia is fognak módosulni.

Példa 21-2. Referenciák és a foreach

```
<?php
$ref = 0;
$SOR =& $ref;
foreach (array(1, 2, 3) as $SOR) {
    // csinál valamit
}
echo $ref; // 3 - a bejárt tömb utolsó eleme
?>
```

Figyelem

Az összetettebb tömbök inkább átmásolódnak mint hogy referenciák készüljenek róluk. Ennek következtében a következő példa nem úgy fog működni ahogy azt elárunk.

Példa 21-3. Referenciák összetett tömbökre

```
<?php
$top = array(
    'A' => array(),
    'B' => array(
        'B_b' => array(),
    ),
);

$top['A']['parent'] = &$top;
$top['B']['parent'] = &$top;
$top['B']['B_b']['data'] = 'test';
print_r($top['A']['parent']['B']['B_b']); // array()
?>
```

A a referenciákat paraméterátadáskor is lehet használni. Ebben az esetben a meghívott függvény egy lokális változója és a hívó környezet egy változója ugyanazt az értéket fogja képviselni. Például:

```
<?php
function ize(&$valtozo)
{
    $valtozo++;
}

$a = 5;
ize($a);
?>
```

Ez a kód az `$a` változó értékét 6-ra állítja. Ez azért történik meg, mivel az `ize` függvényben a `$valtozo` egy referencia a `$a` változó értékére. A részletes leírást a [referenciákénti paraméterátadás](#) c. fejezetben olvashatod.

A referenciák harmadik felhasználási módja a [referencia visszatérési-érték](#).

Mit nem lehet referenciákkal tenni

Mint korábban írtuk, a referenciák nem mutatók. A következő konstrukció ezért nem a vártnak megfelelően viselkedik:

```
<?php
function ize(&$valtozo)
{
    $valtozo =& $GLOBALS['valami'];
}
ize($valami);
?>
```

Az `ize` függvényben a `$valtozo` változó a `$valami` értékéhez lesz kötve, de utána ezt megváltoztatjuk a `$GLOBALS['valami']` értékére. Nincs lehetőség a referenciák segítségével a `$valami` más értékhez kötésére a hívó környezetben, mivel a `$valami` nem áll rendelkezésre az `ize` függvényben. Ott a `$valtozo` reprezentálja az értékét, amely csak változó tartalommal bír és nem név-érték kötéssel a hívó szimbólumtáblájában. A függvény által kijelölt változó hivatkozásához használhatod a [referencia visszatérítést](#).

Referenciákénti paraméterátadás

A függvényeknek változókat referenciáként is át lehet adni, így a függvény tudja módosítani a hívó környezetben definiált értéket. Ez a következőképpen oldható meg:

```
<?php
function ize(&$valtozo)
{
    $valtozo++;
}

$a = 5;
ize($a);
// $a itt 6
?>
```

Figyeld meg, hogy nincs referencia jelzés a függvényhíváskor, csak a függvény definíciójában. Ez önmagában elég a megfelelő működéshez. A PHP újabb verzióiban, egy figyelmezhetést fogsz kapni,

hogy a referencia szerinti átadás hívásidőben (Call-time pass-by-reference) elavult, ha & jelet használsz ilyen esetben: *foo(&\$a);*.

A következők szerepelhetnek referenciakénti paraméterátadásban:

- Változó, például *ize(\$a)*
- New utasítás, például *ize(new osztaly())*
- Egy függvény által visszaadott referencia, például:

```
<?php
function &valami()
{
    $a = 5;
    return $a;
}
ize(valami());
```

Lásd még a [referencia visszatérési-érték](#) leírását.

Minden más kifejezést kerülni kell referencia szerinti paraméterátadáskor, mivel az eredmény határozatlan lesz. A következő példákban a referencia szerinti paraméterátadás hibának minősül:

```
<?php
function valami() // Figyeld meg, nincs & jel!
{
    $a = 5;
    return $a;
}
ize(valami()));

ize($a = 5); // Kifejezés, nem változó
ize(5); // Konstans, nem változó
?>
```

Ezek a meghatározások a PHP 4.0.4 és későbbi verzióira érvényesek.

Refencia visszatérési-érték

A refencia visszatérési-érték pl. olyan változók megtalálásakor lehet hasznos, amelyekről referenciát kell készíteni. Ne használj referencia szerinti visszaadást teljesítménynövelési célból, a motor elég okos ahhoz, hogy magától optimizálja a kódot, csak akkor adj vissza referenciát, ha ésszerű technikai okod van rá. Ha referenciát kell visszaadni visszatérési értékként, akkor használd az alábbi formát:

```
<?php
function &valozo_kereses ($param)
{
    /* ...kód... */
    return $megtalalt_valozo;
}

$size =& valozo_kereses ($valami);
$size->x = 2;
?>
```

Ebben a példában a *valozo_kereses* egy objektumot keres meg, és a megtalált objektum egy tulajdonságát állítjuk át - helyesen. A referenciák használata nélkül a másolatának egy tulajdonságán tettük volna mindenzt - hibásan.

Megjegyzés: A paraméter átadással ellentétben, itt a & jelet minden meg kell adnod a referenciaivisszaadás jelöléséhez. Így nem egy másolatot kapsz, és az *Size* változóra nézve referencia hozzárendelés történik, nem pedig érték hozzárendelés (értékmásolás).

Megjegyzés: Ha ilyen szyntaxssal akarsz referenciát visszaadni függvényből: *return (\$found_var);*, akkor ez nem fog működni, mivel ez egy kifejezés eredményét adja vissza, és nem egy változót referencia szerint. Csak változót adhatsz vissza referencia szerint, semmi mást.

Referenciák megszüntetése

Amikor megszüntetsz egy referenciát, csak megszakítod a változónév és az érték közötti kapcsolatot. Ez nem azt jelenti, hogy a váltózó értékét törlöd. Például:

```
<?php  
$a = 1;  
$b =& $a;  
unset($a);  
?>
```

nem fogja megszüntetni a *\$b* nevet, csak az *\$a* nevet, így az érték a *\$b* néven továbbra is elérhető.

Ismét érdemes a Unix **unlink** parancsával és az állományrendszerrel való hasonlatosságra gondolni.

A PHP által használt referenciák

Sok konstrukció a PHP-ben referenciák segítségével valósul meg, azért minden fentebb tárgyalt kérdés ezekre az elemekre is igaz. Néhány olyan konstrukciót, mint a referencia átadást vagy visszatérést már kifejtettünk, más referenciákat használó konstrukciók:

global referenciák

Amikor egy változót a **global \$valtozo** formával globálisként használsz, tulajdonképpen egy referenciát képzel a megfelelő globális változóra, azaz a következő kódnak megfelelő történik:

```
<?php  
$valtozo =& $GLOBALS['valtozo'];  
?>
```

Ez például azt is jelenti, hogy a *\$valtozo* törlése nem fogja törölni a globális változót.

\$this

Egy objektum metódusban a *\$this* mindig az aktuális példányra egy referencia.

IV. Biztonság

Tartalom

22. [Bevezetés](#)
23. [Általános szempontok](#)
24. [CGI futtatható állományként telepített PHP](#)

25. [Apache modulként telepített PHP](#)
 26. [Fájlrendszer biztonság](#)
 27. [Adatbázis biztonság](#)
 28. [Hibakezelés](#)
 29. [Globálisan is elérhető változók \(Register Globals\) használata](#)
 30. [Felhasználótól érkező adatok](#)
 31. [Magic Quotes](#)
 32. [A PHP elrejtése](#)
 33. [Fontos aktuálisnak maradni](#)
-

22. Fejezet. Bevezetés

A PHP egy igen hatékony nyelv és feldolgozó program, akár kiszolgálómodulként, akár egy különálló CGI futtatható állományként működik. Képes elérni fájlokat, futtatni parancsokat és hálózati kapcsolatokat nyitni a szerveren. Ezek a tulajdonságok alapesetben veszélyessé is tehetik más, a webszerveren futó alkalmazások számára. A PHP-t azonban úgy fejlesztették, hogy biztonságosabb legyen CGI programok írására, mint a Perl vagy C nyelvek. A PHP a fordítási és futásidejű beállítások helyes megválasztásával, és megfelelő programírási módszerek betartásával a szabadság és biztonság kívánt kombinációját biztosítja a fejlesztők számára.

Mivel sokféleképpen és sok mindenre lehet használni a PHP-t, számos konfigurációs lehetőség van a működésének szabályozására. A lehetőségek nagy száma garantálja, hogy a PHP-t sokféleképpen fel lehet használni, de egyben azt is jelenti, hogy ezek és a webkiszolgáló beállításainak kombinációi kritikus helyzeteket teremthetnek.

A beállítások sokszínűsége egyenlő mértékű a kódok sokszínűségével. A PHP használható teljes szerver-alkalmazások készítésére, egy shell felhasználó minden lehetőségével, vagy használható egyszerű 'server side include'-oknál, kis kockázattal egy szigorúan ellenőrzött rendszerben. Az, hogy hogyan kell kialakítani egy környezetet, milyen biztonságosan, nagyban a PHP fejlesztőn múlik.

Ez a fejezet néhány biztonsági tanácsot tárgyal, a különböző beállítási lehetőségeket és azokat a helyzeteket tárja fel, amelyekben ezeket biztonsággal lehet használni. Utána néhány kódolási szempontot is érint a különböző szintű védelem szempontjából.

23. Fejezet. Általános szempontok

A teljesen biztonságos rendszer kialakítani tulajdonképpen lehetlen, ezért a védelmi szakterületen alkalmazott megközelítés a kockázat és a használhatóság közti egyensúly megteremtésére törekszik. Ha minden a felhasználó által küldött adat két biometrikus érvényesítést (pl. retina- és ujjlenyomatvizsgálatot) igényel, akkor igen magas szintű a rendszer "felelősségre vonhatósága" (accountability). Ez azonban azt jelentné, hogy félórába telne kitölteni egy meglehetősen összetett űrlapot, ami arra ösztökelné a felhasználókat, hogy valahogyan megkerüljék ezt a védelmet.

A legjobb védelem gyakran a kevésbé alkalmatlankodó és nem annyira feltűnő fajta, amely megfelel a követelményeknek anélkül, hogy megakadályozná a felhasználókat a munkájuk elvégzésében vagy túlterhelné a program íróit annak túlzott mérvű bonyolultsága. Valójában néhány biztonsági támadás pusztán a kiaknázása az olyasfajta túlságosan is kiépített védelemnek, amely hajlamos elerodálódni az idővel.

Egy mondatot érdemes megjegyezni: A rendszer csakis annyira jól védett, amennyire a leggyengébb láncszeme. Ha minden tranzakcióról feljegyzés készül idő, hely és tranzakciótípus alapján is, de a felhasználót csak egy egyszerű süti (cookie) alapján azonosítja a rendszer, akkor a felhasználók és a

naplózott tranzakciók közti összefüggések érvényessége, megbízhatósága igen gyenge.

Tesztelés során figyelembe kell venni, hogy képtelenség minden lehetőséget kipróbalni már a legegyszerűbb oldalak esetén is. A programozó által várt adatok teljesen különbözök azoktól és minden összefüggést nélkülöznek azokkal, amelyeket egy zsémbelődő alkalmazott képes elküldeni, vagy amelyeket egy szoftverkalóz (cracker) több havi munkájával állít össze, vagy amit egy házimacska a billentyűzetén végiggyalogolva bevisz. Ezért a legjobb a programot logikai nézőpontból megközelíteni, hogy sikerüljön észrevenni, hol jöhetnek elő nem várt adatok és azok a továbbiakban hogyan módosulhatnak, tünhetnek el vagy erősödhetnek fel a hatásuk.

Az Internet tele van olyan emberekkel, akik azzal akarnak maguknak nevet szerezni, hogy feltörök az oldalaikat, tönkreteszik a programjaikat, nem helyéervaló tartalommal töltik fel azokat, mellesleg egy - két izgalmas(?) napot szerezve ezzel Neked. Nem számít, hogy kis vagy nagy webhelyről van szó, elég indok a támadásra, hogy az rá van kapcsolva a hálóra, van egy szerver, amelyhez csatlakozni lehet. Sok kódtörő program nem foglalkozik a méretekkel, egyszerűen csak nagy mennyiségi IP blokkokra vadászik áldozatokat keresve ezzel magának. Próbálj meg nem egy lenni közülük!

24. Fejezet. CGI futtatható állományként telepített PHP

Lehetséges támadások

A PHP CGI futtatható állományként való használata egy telepítési lehetőség azok számára, akik valami oknál fogva nem szeretnék a PHP-t modulként a szerverbe integrálni (pl. Apache), vagy a PHP-t más CGI wrapper-ekkel szeretnék használni biztonságos chroot és setuid környezet kialakítása érdekében. Ez a forma magával vonja azt, hogy a PHP a szerver cgi-bin könyvtárába legyen telepítve. A CERT advisory [CA-96.11](#) azt tanácsolja, hogy ne tegyél feldolgozó (interpreter) programot a cgi-bin könyvtárba. Bár a PHP használható mint egy egyedülálló feldolgozó program, a PHP-t úgy terveztek, hogy az ilyen telepítésekben adódó támadásokat kivédje:

- Rendszerfájlok elérése: `http://domain.nev/cgi-bin/php?/etc/passwd`

Az URL lekérési információja (query information), ami a kérdőjel (?) után található, parancssori paraméterként kerül átadásra a feldolgozónak. Általában a feldolgozók megnyitják, és lefuttatják az első paraméterként adott fájlt.

Ha a PHP CGI futtatható állományként hívódik meg, nem veszi figyelembe a parancssori paramétereiket.

- Bármilyen web dokumentum elérése a szerveren: `http://domain.nev/cgi-bin/php/titkos/doc.html`

Az elérési út információ (path information) az URL része, a futtatható fájl neve után lévő /titkos/doc.html a CGI program által megnyitásra és futtatásra kerülő fájl elérésének meghatározására használatos. Tipikusan néhány webkiszolgáló beállítási lehetőség (Apache-ban: Action) használatos a kérések átirányítására a dokumentumhoz, mint a `http://domain.nev/titkos/szkript.php` a PHP értelmező számára. Ezzel a beállítással a szerver először ellenőri az elérési engedélyeket a /titkos könyvtárra, és ezután állítja elő az átirányító kérést a `http://domain.nev/cgi-bin/php/titkos/szkript.php` oldalra, amit így már a PHP feldolgoz. Azonban ha eredetileg is ebben a formában volt megadva a kérés, nem történik elérési ellenőrzés a /titkos/szkript.php fájusra, csak a /cgi-bin/php fájusra. Ilyen módon bárki, aki elérheti a /cgi-bin/php címet, egyben tetszőleges védett dokumentumot is elérhet.

A PHP esetében az [–enable-force-cgi-redirect](#) fordítási paraméter, a [doc_root](#) és [user_dir](#)

konfigurációs lehetőségek használhatóak ennek kivédésére, ha a szerver dokumentumainak könyvtárfájában van olyan könyvtár, ami elérési korlátozásokkal bír. Nézd meg az alábbi lehetőségeket a különböző kombinációkhöz!

1. eset : csak publikus fájlok

Ha a szerveren nincs olyan tartalom, ami jelszó vagy IP alapú védelemmel van ellátva, nincs szükség ezekre a konfigurációs beállításokra. Ha a kiszolgáló nem engedélyezi az átirányításokat, illetve ha nincs módja biztonságos átirányítással küldeni a kérést a PHP számára, megadhatod az [--enable-force-cgi-redirect](#) opciót a "configure" szkript számára. Meg kell győződni arról, hogy a PHP szkriptjeid nem függnek egy speciális szkript-hívási formától sem, mint a `http://domain.nev/cgi-bin/php/dir/szkript.php` vagy a `http://domain.nev/dir/szkript.php`.

Az átirányítás beállítása Apache alatt az AddHandler és Action direktívákkal történik (lásd lentebb).

2. eset : az --enable-force-cgi-redirect használata

Ez a fordítási paraméter megakadályozza, hogy bárki meghívja a PHP-t egy `http://domain.nev/cgi-bin/php/titkos/szkript.php` URL-el. Ehelyett a PHP csak akkor fog elfogadni egy ilyen kérést ha egy szerver átirányításban kapta.

Apache esetében tipikusan a következő direktívákkal történik a beállítás:

```
Action php-script /cgi-bin/php  
AddHandler php-script .php
```

Ez a lehetőség csak az Apache web szerverrel tesztelt és azon múlik, hogy az Apache beállítja a nem standard *REDIRECT_STATUS* CGI környezeti változót ha átirányított kérésről van szó. Ha a webkiszolgálód semmilyen módon nem közli, hogy ez egy direkt vagy átirányított kérés volt-e, nem használhatod ezt az opciót, így valamelyik másik módot kell használnod.

3. eset : a doc_root vagy user_dir beállítása

Aktív tartalom elhelyezése a normál dokumentumok között, (pl. szkriptek és futtatható állományok) veszélyes gyakorlat lehet. Ha például valamilyen beállítási hiba miatt a szkriptek ahelyett, hogy lefutnának hagyományos HTML dokumentumokként jelennek meg, mindenki számára tisztán láthatóvá válnak kódolási technikáid és titkos információik, mint például adatbázis jelszavaid. Ezért néhány rendszeradminisztrátor inkább egy külön könyvtárat jelöl ki, ami csak a PHP CGI által elérhető, és így minden felhasználónak kerül előttük a szkript kódja.

Ha a fent leírt átirányítás azonosítási mód nem működik, fontos, hogy egy különálló szkript `doc_root`-ot határozz meg, ami nem azonos a web `doc_root`-al.

A PHP szkript dokumentumok gyökérkönyvtárát a [doc_root](#) konfigurációs beállítással határozhatod meg a [konfigurációs fájlban](#), vagy a *PHP DOCUMENT_ROOT* környezeti változóban adhatod meg ezt az értéket. Ha ez be van állítva a PHP CGI verziójá a fájl elérési útját a `doc_root` és a kérés elérési út információja (path information) alapján állítja elő, ami azt jelenti, hogy ezen a könyvtáron kívül nem futtatható fájl. (kivéve a `user_dir` esetét).

Egy másik itt használható opció a [user_dir](#). Ha ez nincs megadva, csak a `doc_root` szabályozza a megnyitható fájlok körét. Ekkor egy `http://domain.nev/~user/doc.php` URL nem a "user" nevű felhasználó home könyvtárában lévő fájlt keresi, hanem a `~user/doc.php` fájlt

keresi a doc_root alatt (igen, egy tilde karakterrel kezdődő könyvtárban [~]).

Ha a user_dir meg van adva, például public_php, akkor a fenti `http://domain.nev/~user/doc.php` kérés a `doc.php` nevű fájlt fogja megnyitni a "user" nevű felhasználó home könyvtárában lévő `public_php` könyvtárban. Ha a "user" home könyvtára `/home/user`, a lefuttatandó fájl a `/home/user/public_php/doc.php` lesz.

A user_dir kifejtés a doc_root beállítástól függetlenül működik, úgyhogy a dokumentum gyökér és felhasználói könyvtár beállításokat külön is használhatod.

4. eset : PHP feldolgozó a web könyvtárfán kívül

Rendkívül biztonságos lehetőség a PHP feldolgozót valahol a webről látható könyvtárakon kívülre tenni. Például az `/usr/local/bin` könyvtárba. Az egyetlen igazi hátránya ennek az opciónak az, hogy minden PHP szkript első sorának egy ehhez hasonló sort kell megadnod:

```
#!/usr/local/bin/php
```

ami meghatározza, hogy hol található a PHP feldolgozó, ami lefuttatja majd ezt a kódot. Ráadásul minden PHP szkriptnek futási jogot kell adni. Azaz úgy kell eljárni, mint bármilyen más nyelven megírt CGI programmal, amit Perl, sh vagy és a `#!/` shell-escape mechanizmust használja önmaga futtatására.

Ahhoz, hogy ebben az esetben a PHP helyesen kezelje a `PATH_INFO` és a `PATH_TRANSLATED` információkat, a PHP feldolgozót az [`-enable-discard-path`](#) "configure" paraméterrel kell fordítani.

25. Fejezet. Apache modulként telepített PHP

A PHP-t Apache modulként használva örökli az Apache-t futtató felhasználó (tipikusan a "nobody") jogait. Ennek többféle hatása van a biztonságra és az azonosításra. PHP-n kereszttüli adatbázis elérés esetén például, az adatbázist elérhetővé kell tenni ezen felhasználó számára is, kivéve ha az adatbázisnak beépített azonosítása van. Ez azt jelenti, hogy egy rosszindulatú szkript elérheti, és módosíthatja az adatbázist, akár felhasználói név és jelszó nélkül is. Lehetséges, hogy egy keresőrobot beleakadjon az adatbázis-adminisztrációs oldalak egyikébe, és kiürítse az összes adatbázist. Természetesen lehet ez ellen védekezni Apache azonosítási technikákkal, vagy LDAP segítségével megvalósított saját elérési modellekkel, vagy külön .htaccess fájlokkal, stb., és ezeket a saját PHP kód részévé is lehet tenni így.

Általában, ha a biztonságot akkora szintre tudjuk emelni, hogy a PHP felhasználó (ebben az esetben az Apache-é) igen kis kockázattal fut, akkor nem képes például akármilyen fájlok írására a user könyvtárakba. Letilthatjuk számára egy adatbázis elérését vagy megváltoztatását. Tipikusan ebben a helyzetben már azokat a fájlokat sem tudja írni, amit kellene, vagy egyaránt nem tud végrehajtani jó és rosszindulatú adatbázis tranzakciókat egyaránt.

Gyakori hiba ezen a ponton, hogy az Apache-nak root jogokat adnak vagy valamelyen egyéb módon bővíti az Apache jogait / lehetőségeit.

Az Apache felhasználó jogainak root szintre bővítése különösen veszélyes, és tönkreteheti a teljes rendszert, tehát sudo, chroot vagy más hasonló eszközök használata elkerülendő, ha nem vagy biztonsági szakember.

Van néhány egyszerűbb megoldás is. Az [`open_basedir`](#) használatával szabályozni lehet, hogy mely könyvtárakat olvashatja a PHP. Ki lehet jelölni ún. csak apache-s területeket, hogy minden web alapú művelet ide, csak ezekre a nem rendszer- és nem felhasználói fájlokra korlátozódjon.

26. Fejezet. Fájlrendszer biztonság

A PHP tiszeli a rendszerbe épített biztonsági megoldásokat, különös tekintettel a fájlok és könyvtárak hozzáférési jogosultságaira. Ez lehetőséget ad arra, hogy megszabd, mely fájlok olvashatóak a rendszerben. A mindenki számára olvasható fájloknál ügyelni kell arra, hogy ne tartalmazzanak olyan fontos adatot, amit nem szabad elolvasnia akármelyik felhasználónak a rendszeren.

Mivel a PHP úgy készült, hogy felhasználói szintű fájlrendszer hozzáférést ad, lehetséges olyan program készítése, amely a rendszerfájlokat olvassa, mint pl. az /etc/passwd fájl, ethernet kapcsolatokat módosít, nagyméretű nyomtatási feladatokat küld, stb. Ez maga után von egy nyilvánvaló következtetést, nevezetesen minden esetben meg kell győzdeni a programokban arról, hogy a helyes fájlokat olvassa illetve írja a program.

Nézzük a következő szkriptet, ahol a felhasználó megadja, hogy le szeretne törölni egy fájlt a könyvtárában. Ez többnyire egy webes felületet jelent, ahol egy PHP program használatos fájlkezelésre, ezért az Apache-t futtató felhasználónak engedélyezni kell a fájlok törlését a felhasználó könyvtárában.

Példa 26-1. A helytelen változó használat ...

```
<?php  
// egy fájl törlése a user könyvtárából  
$usernev = $_POST["user_antal_beadott_nev"];  
$konyvtar = "/home/$usernev";  
$storlendo_file = "$userfile";  
unlink ($konyvtar/$storlendo_file);  
echo "$storlendo_file törölve!";  
?>
```

Mivel a \$usernev egy HTML ürlapból érkezik, a felhasználó beírhat tetszőleges felhasználói nevet és fájlnevet, így akár más könyvtárat is manipulálhatja. Ebben az esetben általában valamilyen felhasználó-azonosítási eljárást kell alkalmazni. Lássuk, mi történik, ha a beadott változók a "../etc/" és a "passwd". A kód akkor így alakulna (az adatokat behelyettesítve):

Példa 26-2. ... fájlrendszer támadáshoz vezethet

```
<?php  
// egy fájl törlése akárhonnan, ahol a PHP usernek  
// joga van erre. Ha a PHP root userként fut:  
$usernev = "../etc/";  
$konyvtar = "/home/../etc/";  
$storlendo_file = "passwd";  
unlink ("/home/../../etc/passwd");  
echo "/home/../../etc/passwd törölve!";  
?>
```

Két fontos komponensre kell odafigyelni, hogy megelőzhessük az ilyen problémákat:

- Csak korlátozott jogok beállítása a PHP-t futtató felhasználónak.
- minden felhasználótól bejövő adat ellenőrzése.

Egy jobban átgondolt, tökéletesített szkript:

Példa 26-3. Biztonságosabb fájl ellenőrzés

```
<?php  
// egy fájl törlése akárhonnan, ahol a PHP usernek  
// joga van erre.  
$usernev = $_SERVER['REMOTE_USER']; // ez a user azonosított neve  
// (ha volt előtte azonosítás)  
  
$konyvtar = "/home/$usernev";
```

```
$storlendo_file = basename("$userfile"); // elérési trükközés eldobása  
unlink ($konyvtar/$storlendo_file);  
  
$fp = fopen("/home/logging/filedelete.log","+a"); // törlés naplázása  
$logstring = "$usernev $konyvtar $storlendo_file";  
fputs ($fp, $logstring);  
fclose($fp);  
  
echo "$storlendo_file törölve!";  
?>
```

Mindamellett, ez sem mentes a hiányosságoktól. Ha az aktuálisan használt hitelesítési módszer (authentication) megengedi a felhasználóknak, hogy saját loginnevet válasszanak, és az egyikük a "../etc/" -t választja, akkor a rendszer ugyanolyan védtelessé válik. Ebből kiindulva a jobban testreszabott ellenőrzés a következőképp alakulna:

Példa 26-4. Biztonságosabb fájlnév-ellenőrzés

```
<?php  
$usernev = $_SERVER['REMOTE_USER']; // hitelesítés  
$homedir = "/home/$usernev"; $homedir = "/home/$usernev";  
  
if (!ereg('^[^/][^/]*$', $userfile))  
    die('rossz fájlnév'); // vége, nincs feldolgozás  
  
if (!ereg('^[^/][^/]*$', $usernev))  
    die('rossz usernev'); // vége, nincs feldolgozás  
//stb...  
?>
```

A használt operációs rendszertől függően széles a védeni kívánt fájlok skálája, beleértve az eszköz hivatkozásokat (/dev/ vagy COM1), konfigurációs fájlokat (/etc/ és az .ini fájlok), jól ismert tárolóhelyek (/home/, My Documents), stb. E sokaság miatt könnyebb egy olyan rendszert készíteni, ahol minden tiltunk azon kívül, amelyet kifejezetten megengedünk.

27. Fejezet. Adatbázis biztonság

Mostanában, a dinamikus tartalmat szolgáltató web alkalmazások sarokkövének számítanak az adatbázisok. Mivel nagyon kényes, titkos adatok tárolására szolgálhatnak ezek az adatbázisok, erősen megfontolandó, miképp védjük meg ezeket.

Információk tárolásához vagy visszakereséséhez csatlakozni kell az adatbázishoz, egy érvényes lekérdezést kell küldeni, az eredményt ki kell olvasni, és le kell zárnai a kapcsolatot. Manapság ebben a párbeszédben a Structured Query Language (SQL) a leggyakrabban használt lekérdezőnyelv. Figyeld meg, miként lehet [SQL lekérdezéseket megbabralni!](#)

Mint látható, a PHP egymagában, magától nem képes megvédeni az adatbázist. A következő bekezdések célja, hogy betekintést adjanak az alapokba, hogyan kell adatbázisokat elérni és módosítani egy PHP programon belül.

Tartsd észben a következő egyszerű szabályt: tagoltan védekezni. Minél több helyen minél többet teszel a biztonság növeléséért, annál kisebb a valószínűsége, hogy a támadók sikerrel járjanak, és kiteregessek titkos adataidat, vagy visszaéljenek velük. A jó adatbázis- és alkalmazástervezés mindig a legnagyobb félelmek figyelembevételéről ismerszik meg.

Adatbázis-tervezés

Az első lépés mindenkor az adatbázis létrehozása, hacsak nem egy kívülállóét kell használni. Az adatbázis létrehozásakor az a tulajdonosáé lesz, azé, aki lefuttatta az utasításokat. Általában csak a tulajdonos - esetleg az ún. superuser - jogosult bármiféle az adatbázis elemeit érintő műveletre. Annak érdekében, hogy más felhasználók is hozzáférjenek, jogokat kell nekik biztosítani.

Az alkalmazásoknak soha nem szabad a tulajdonosaként vagy superuserként csatlakozni az adatbázishoz, mert ezek bármilyen utasítást és lekérdezést tetszés szerint futtathatnak, pl. a szerkezeti módosítást (táblák megszüntetése) vagy táblák komplett törlése.

Létre lehet hozni különböző, szigorúan korlátozott jogosultásgú adatbázis- felhasználókat, melyek mindegyike az adatbázis manipulációjának egy-egy különböző nézőpontjáért felelősek. Mindig csak a legszükségesebb jogokat szabad engedélyezni, és el kell kerülni, hogy ugyanazt a felhasználót használjuk szerepeiben egymástól különböző esetekben. Ez azt jelenti, hogy ha a behatoló meg is szerzi valamelyik ilyen minősítést (hitelesítési információt = felhasználói név + jelszó), akkor is csak akkora változást tud okozni, mint az alkalmazás maga.

Nem kell minden feladatfüggő szabályozást a webalkalmazásban (PHP szkriptben) kódolni, ehelyett inkább használ az adatbázis lehetőségeit: view-k (nézetek), trigger-ek, rule-ok (szabályok). Ha a rendszer fejlődik, és más alkalmazásokat is csatlakoztatni kell az adatbázishoz, akkor mindegyiknél újra kellene programozni ezeket a szabályokat. Mindezen felül a triggerek arra is jók, hogy átlátszó módon és automatikusan kezeljenek egyes mezőket az adatbázisban, amelyek gyakran bepillantást adnak abba, hogy mi is történik/történt egy tranzakció közben, vagy nagyon hasznosnak bizonyulhatnak hibakeresés során.

Kapcsolódás az adatbázishoz

Elképzelhető, hogy SSL-n keresztül szeretnél kapcsolódni az adatbázishoz, hogy a kiszolgáló és ügyfél közti teljes kommunikáció titkosításával növeld a védelmet. Használhatsz ssh-t is erre a célra. Akármelyik is áll, nagyon nehéz lesz a forgalom lehallgatásából információkat kinyerni ezek után.

Titkosított tárolás

SSL/SSH az ügyfél és kiszolgáló közt mozgó adatokat védi, és nem védi az adatbázisban tárolt megmaradó adatokat. Az SSL - kapcsolati protokoll.

Mihelyst a támadó közvetlen hozzáférést szerzett az adatbázishoz - megkerülve a webszervert -, a tárolt adatok védtelennek váltak, és visszaélhet velük, ha csak maga az adatbázis nem védi valahogy azokat. Az adatok titkosítása kellőképp enyhíti ezt a veszélyt, de jelenleg nagyon kevés adatbázis kezelő támogatja a titkosítást.

Ez a legkönyebben saját titkosító csomag írásával oldható meg, amelyet utána a PHP szkriptből el lehet érni. Ebben az esetben a PHP segítséget nyújthat néhány kiterjesztéssel, mint például az [Mcrypt](#) vagy az [Mhash](#), amelyek nagyon sokféle titkosító algoritmust fednek le. A szkript az adatbázisban való tárolás előtt titkosítja a tárolni kívánt adatot, majd visszakereséskor visszafejt azokat. Nézd meg a hivatkozott fejezeteket további példákért, hogyan kell a titkosítást végrehajtani.

Olyan teljesen rejtett adatok esetén, amelyeknek nyílt ábrázolásukra nincs szükség, mert nem lesznek kiíratva, a hashelés alkalmazása is meggondolandó. A hashelés jól ismert példája az, hogy a jelszavak helyett, azoknak csak MD5 hash értékét tárolják az adatbázisban. Lásd még: [crypt\(\)](#) és [md5\(\)](#)!

Példa 27-1. Hashelt jelszó mező használata

```
<?php

// jelszó hash értékének tárolása
$query = sprintf("INSERT INTO users(name, pwd) VALUES ('%s', '%s');",
                  addslashes($username), md5($password));
$result = pg_query($connection, $query);

// lekérdezés, vajon a felhasználó a helyes jelszót adta-e meg
$query = sprintf("SELECT 1 FROM users WHERE name='%s' AND pwd='%s';",
                  addslashes($username), md5($password));
$result = pg_query($connection, $query);

if (pg_num_rows($result) > 0) {
    echo "Üdvözöllek, $username!";
} else {
    echo "$username hitelesítése nem sikerült.";
}

?>
```

SQL "beoltás"

Sok web fejlesztő nincs tudatában annak, hogy hogyan lehet megbabrálni az SQL utasításokat, ezért az SQL utasításokat megbízható parancsoknak feltételezik. Ez azt jelenti, hogy az SQL lekérdezésekkel ki lehet játszani a hozzáférés szabályozásokat, meg lehet kerülni a szabályos engedélyezési folyamatokat, és néha az SQL lekérdezésekkel a gazdagépen operációs rendszer szintű hozzáférést is lehet létrehozni.

A "közvetlen SQL utasítás befecskendezés" olyan módszer, amellyel a támadó a régi SQL utasításokat módosítja vagy újakat ad hozzájuk annak érdekében, hogy titkos információkhoz jusson hozzá, vagy felülírja azokat, vagy veszélyes rendszer szintű parancsokat futtasson az adatbázis gazdagépénen. Ez olyan alkalmazások esetén tehető meg, amelyek a felhasználótól származó adatokból és statikus paraméterekből állítanak össze SQL lekérdezéseket. Sajnos, a következő példák minden megtörtént eseteken alapulnak.

Az, hogy az adatbázishoz superuserként (olyan személyként, aki superusert képes létrehozni) csatlakozott az alkalmazás, és a bevitt adatok ellenőrzésének hiánya odavezethet, hogy a támadó superuser hozzáférést hozhat létre az adatbázishoz.

Példa 27-2. A keresési eredmények lapokra tördelése ... és superuserek létrehozása (PostgreSQL)

```
<?php

$offset = $argv[0]; // Vigyázz, nincs beviteli ellenőrzés!
$query = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";
$result = pg_query($conn, $query);
?>
```

A szokványos felhasználó az 'előző', 'következő' linkekre kattint, ahol az `$offset` az URL-be van kódolva. A szkript azt várja, hogy `$offset` decimális szám. Mégis, valaki megpróbálhatja a következő utasítás [urlencode\(\)](#) alakját hozzáfűzni az URL-hez:

```
0;
insert into pg_shadow(username,usesysid,usesuper,usecatupd,passwd)
  select 'crack', usesysid, 't','t','crack'
    from pg_shadow where username='postgres';
--
```

Ha ez megtörténne, akkor a szkript megajándékozná a támadót egy superuser hozzáféréssel. A `0;` arra való, hogy érvényes offset-et biztosítson az eredeti lekérdezésnek.

Megjegyzés: Általános módszer, hogy a -- jellel kényszerítik ki, hogy az SQL elemző figyelmen kívül hagyja a lekérdezésként átadott string fennmaradó részét, mivel ez a megjegyzés szabványos jelölése SQL-ben.

Egy lehetséges módja a jelszavak megszerzésének, hogy kijátszik a kereső oldalak találati listájának lekérdezéseit. A támadónak minden össze annyit kell tennie, hogy végig próbálja melyik elküldött SQL lekérdezésben használt változó nincs megfelelően lekezelve. Ezeket általában egy megelőző ürlapon lehet beállítani, hogy testre szabjuk a *SELECT* utasítás *WHERE*, *ORDER BY*, *LIMIT* és *OFFSET* klauzuláit. Ha a használt adatbáziskezelő támogatja a *UNION* szerkezetet, akkor a támadó esetleg hozzáfűzhet egy teljesen új lekérdezést a már meglevőhöz, hogy kilistázza valamelyik táblában tárolt jelszavakat. Titkosított tárolás erősen ajánlott!

Példa 27-3. Árucikkek listázása ... és néhány jelszóé (valamilyen adatbázis kezelő)

```
<?php  
  
$query = "SELECT id, name, inserted, size FROM products  
          WHERE size = '$size' AND inserted BETWEEN '$min_date' AND '$max_date'  
                ORDER BY $order LIMIT $limit, $offset;";  
$result = odbc_exec($conn, $query);  
  
?>
```

A lekérdezés statikus része egy másik *SELECT* utasítással kombinálható, ami az összes jelszót kilistázza:

```
'  
union select '1', concat(uname||'-'||passwd) as name, '1971-01-01', '0' from usertable;  
--
```

Ha ezt a lekérdezést (a ' és -- megfelelő használatával) valamelyik \$query-ben használt változóhoz sikerülne hozzárendelni, akkor a szörny felébredne.

SQL UPDATE parancsok ugyancsak ki vannak téve az adatbázisok elleni támadásoknak. Ezeket az utasításokat is fenyegetik az előzőekben megismert megrövidítő és hozzáfűző technikák. Ám emellett a támadó meghamisíthatja a *SET* klauzulát is. Ebben az esetben némi séma információval rendelkeznie kell a támadónak, hogy sikerrel járjon. Ezeket az információkat az ürlapváltozók neveiből szerezhetik meg, vagy egyszerűen próbálgatással. Az általánosan használt elnevezések a felhasználói névre és jelszóra nem nagyon különböznek egymástól.

Példa 27-4. Jelszó átírásától ... új jogok megszerzéséig (valamilyen adatbázis kezelő)

```
<?php  
$query = "UPDATE usertable SET pwd='$pwd' WHERE uid='$uid';";  
?>
```

A rosszindulatú felhasználó a ' or uid like '%admin%' -- értéket adja át a \$uid változónak, és ezzel megváltoztatja az adminisztrátor jelszavát, vagy egyszerűen a \$pwd-nek a "hehehe", admin='yes', trusted=100 " (lezáró szóközzel) értéket adva még több jogot szerez magának. Ezt az SQL parancsot ezek így fordítik el:

```
// $uid == '' or uid like'%admin%'; --"  
$query = "UPDATE usertable SET pwd='....' WHERE uid='' or uid like '%admin%'; --';"  
  
// $pwd == "hehehe", admin='yes', trusted=100 "  
$query = "UPDATE usertable SET pwd='hehehe', admin='yes', trusted=100 WHERE ...";
```

Egy ijesztő példa, hogyan lehet az adatbázis gazdagépén operációs rendszerszintű parancsokat futtatni.

Példa 27-5. Az adatbázis-gazdagép operációs rendszere elleni támadás (MSSQL Server)

```
<?php
```

```
$query = "SELECT * FROM products WHERE id LIKE '%$prod%'";
$result = mssql_query($query);

?>
```

Ha a támadó az `a%' exec master..xp_cmdshell 'net user test testpass /ADD' --` értéket küldi el a `$prod` változónak, akkor a `$query` a következőképp alakul:

```
$query = "SELECT * FROM products
          WHERE id LIKE '%a%'
          exec master..xp_cmdshell 'net user test testpass /ADD'--%'";
$result = mssql_query($query);
```

MSSQL Server futtatja a kötegbe fogott SQL utasításokat, köztük azt is, amelyik új felhasználót vesz fel az adatbázis kiszolgálójában. Ha az alkalmazás `sa` jogosultsággal fut és az MSSQLSERVER service megfelelő jogokkal fut, akkor a támadónak most már hozzáférése van ehhez a géphez.

Megjegyzés: A példák nemelyike bizonyos adatbáziskezelőhöz kötődik. Ez nem azt jelenti, hogy hasonló támadás elképzelhetetlen más termékek ellen. Az általad használt adatbázis-kezelő ugyanilyen sérülékeny lehet, akár más módon.

Elhárítási módszerek

Ellenevetésként felmerülhet, hogy a példák többségében a támadónak rendelkeznie kell valamennyi előzetes információval az adatbázis felépítéséről. Ez igaz, de soha nem lehet tudni, hogy mikor, hol, hogyan szerezhetik meg ezeket, és ha ez megtörtént, az adatbázisod védtelenül válik. A behatólók könnyen hozzájuthatnak a program egy darabjához nyílt forráskódú, vagy olyan nyilvánosan elérhető adatbázis-kezelő programcsomag használatakor, amelyik egy fórum vagy tartalomszolgáltató rendszer része. Ez különösen veszélyes lehet, ha ezek kevéssé átgondoltak és gyengén megtervezettek.

Ezek a támadások alapvetően olyan programoknak a kijátszásán alapulnak, amelyek a védelmet/biztonságot figyelmen kívül hagyva születtek. Soha nem lehet megbízni semmilyen bejövő adatban, főleg ha az a kliens oldalról érkezik, még akkor sem, ha az egy általunk megadott süti (cookie), vagy rejtett mező (hidden input) értéke esetleg egy legördülő lista eleme. Még egy olyan ártatlan lekérdezés, mint ami az első példában látható, katasztrófát okozhat.

- Soha ne csatlakozz az adatbázishoz tulajdonosaként vagy superuser-ként. Mindig kevés jogosultsággal rendelkező, testreszabott felhasználókat használj!
- Ellenőrizd a bejövő adat típusát, hogy az a vártak megfelelő-e! A PHP a bevitelt ellenőrző függvények széles körével rendelkezik kezdve a legegyszerűbbektől - pl.: [Változókkal kapcsolatos függvények](#) közül [is_numeric\(\)](#) vagy a [Character Type Functions](#) közül a [ctype_digit\(\)](#) - a [Perl kompatibilis reguláris kifejezések](#) támogatásáig.
- Ha az alkalmazás számot vár, akkor megfontolandó az [is_numeric\(\)](#) függvénytel ellenőrizni a típusát, vagy csendben megváltoztatni a típusát a [settype\(\)](#) függvénytel, vagy szám szerinti ábrázolását használni az [sprintf\(\)](#) függvénytel.

Példa 27-6. A lapozáshoz használt lekérdezés összeállításának biztonságosabb módja

```
<?php

settype($offset, 'integer');
$query = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $offset;";

// Figyelj a %d -re a formázó sztringben, a %s használat értelmetlen lenne
```

```
$query = sprintf("SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET %d;" .  
    ?>
```

- Idézőjelek közé kell tenni minden nem szám jellegű, felhasználótól származó adatot, erre használható az adatbázis-specifikus escape függvény (pl. [mysql_escape_string\(\)](#), [sql_escape_string\(\)](#), stb.). Ha nincs ilyen adatbázis-specifikus escape mechanizmus, akkor hasznosnak bizonyulhatnak az [addslashes\(\)](#) és a [str_replace\(\)](#) függvények (az adatbázistól függően). Lásd még [az első példát!](#) Ahogy a példa is mutatja, a statikus részbe beírt idézőjelek nem elegendőek, mivel ez a lekérdezést könnyen feltörhetővé teszi.
- Semmilyen adatbázisra jellemző információt - különösen szerkezetit - nem szabad kiírni, ha törik, ha szakad. Lásd még: [Hibajelzés](#) és [Hibakezelő és naplózó függvények!](#)
- Tárolt eljársokat és előre definiált kurzorokat is használhatsz, hogy az adatbázis elérést absztrahál annak érdekében, hogy a felhasználók ne közvetlenül a táblákhoz vagy nézetekhez férjenek hozzá. Ennek a megoldás azonban egyéb hatásai vannak.

Ezeken kívül, hasznos lehet a lekérdezések naplózása akár a szkripteken belül, akár ha az adatbázis kezelő maga teszi ezt. Nyilvánvalóan ez nem tud megakadályozni egyetlen ártalmas próbálkozást sem, de segítséget nyújthat annak felderítésében, hogy melyik alkalmazás lett kijátszva. A naplózás önmagában nem, csak a benne megjelenő információkon keresztül válik hasznossá: általában a több részlet, hasznosabb.

28. Fejezet. Hibakezelés

A PHP biztonsági kérdések felől a hibajelzéseknek két oldaluk van. Az egyiket nézve hasznos a védelem növelése szempontjából, a másik szemszögből viszont káros.

Egy szokásos támadási technika minél több információ begyűjtése a rendszerről. Ezt úgy próbálják megoldani, hogy helytelen adatokat küldenek be, és rögzítik a hibaüzenetek típusait és környezetüket. Ez lehetőséget ad a crackernek, hogy elég információt gyűjtsön a rendszerről, és meghatározza a lehetséges gyenge pontokat. Ha például a támadó összeszedegéttel elég információt az előző űrlap kitöltések alapján, akkor megpróbálhatja a változókat felülírni vagy megváltoztatni őket:

Példa 28-1. Változók elleni támadás egy HTML oldallal

```
<form method="post" action="tamadas_celpontja?usernev=rosszsize&jelszo=rosszsize">  
<input type="hidden" name="usernev" value="rosszsize">  
<input type="hidden" name="jelszo" value="rosszsize">  
</form>
```

A PHP által visszaadott hibaüzenetek általában hasznosak a hibákat kereső fejlesztő számára, megjelölve a fájlt, és a függvényt, ami hibás, megadva a megfelelő programsor számát. Ez az összes információ, amit ki lehet nyerni. Nem ritka, hogy egy PHP fejlesztő a [show_source\(\)](#), [highlight_string\(\)](#), vagy [highlight_file\(\)](#) függvényeket a fejlesztés során hibakeresésre is használja, de egy élesben lévő webhelyen ez rejttett változókat, ellenőrizetlen kódokat, és más veszélyes információkat fedhet fel. Kifejezetten veszélyes beépített hibakezelővel rendelkező ismert forrású kódok használata. Ha a támadó ráismer valamilyen általános programozási technikára, akkor megpróbálhatja a nyers erőre alapozva feltörni az oldalt a különböző megszokott hibakereső (debugging) változók elküldésével:

Példa 28-2. Szokványos hibakereső változók kihasználása

```
<form method="post" action="tamadas_celpontja?errors=Y&ampampshowerrors=1&debug=1">  
<input type="hidden" name="errors" value="Y">  
<input type="hidden" name="showerrors" value="1">  
<input type="hidden" name="debug" value="1">
```

```
</form>
```

A hibakezelés módjától függetlenül az a lehetőség, hogy egy rendszerben hibák után kuthatnak, odavezet, hogy a támadók is több információhoz jutnak.

Az általános hibaüzenetek nagyrészéből például beazonosítható, hogy a rendszer PHP-t használ. Ha a támadó egy .html oldalt látott, és ismert hibákat kihasználva meg akarta tudni, hogy milyen alkalmazást használ a rendszer, hibás adatokat beküldve azonosíthatja, hogy az oldalt egy PHP program állította elő.

Egy függvényhiba elárulhatja, hogy a rendszer milyen adatbázismotort használ, vagy hogy milyen programozói stíllussal készült az adott weblap. Ez mélyebb kutatásokra ad lehetőséget nyitott adatbázisportok irányában, vagy tipikus hibák illetve gyengeségek keresését jelentheti. Különböző hibás adatok küldésével a támadó meg tudja állapítani, hogy milyen sorrendben végzed az azonosításokat (a hibák sorszámaiból). Ezzel a gyenge pontok is könnyen megtalálhatóak egy szkriptben.

A fájlrendszer vagy általános PHP hibák jelezhetik, hogy milyen jogokkal rendelkezik a webszerver, és megmutathatják a fájlok elrendezését és struktúráját. A fejlesztő által írt hibás kód súlyosbíthatja a helyzetet, egykor 'rejtett' információk könnyű kiderítését téve lehetővé.

Három megoldási lehetőség adódik erre a problémára. Az első, megvizsgálni alaposan a függvényeket, és megpróbálni elkerülni a hibákat. A második a hibajelzés kikapcsolása a teljes kódon belül. A harmadik a PHP testreszabható hibajelentési funkcióinak használata, hogy saját hibakezelőket definiálj. A már megtett biztonsági intézkedésektől függően esetleg mindenből fenti módszert választható.

Megelőzendő a bajt használhat a PHP beépített [error_reporting\(\)](#) függvénye, amely segít biztonságosabbá tenni a programokat és megtalálni a változók vészelyeket rejtő használati formáit. A bevezetés előtti tesztelés során E_ALL beállítással gyorsan meg lehet találni azokat a pontokat, ahol a változók könnyen és/vagy rosszindulatúan módosíthatók. Ha a program már kész bevezetésére, teljesen kikapcsolhatod a hibajelentést az [error_reporting\(\)](#) 0-ra állításával, vagy kikakolod a hibák megmutatását a `php.ini display_errors` opciójával, ezzel teljesen leszigetelve a kódot a további vizslatásoktól. Ha az utóbbiit választod, meg kell adnod a naplófájl útvonalát az `error_log` ini direktívával, majd a `log_errors` direktívát on-ra kell állítanod.

Példa 28-3. Veszélyes változók felderítése az E_ALL segítségével

```
<?php
if ($usernev) { // nincs inicializálva vagy ellenőrizve használat előtt
    $jo_belepes = 1;
}
if ($jo_belepes == 1) { // ha az előző feltétel hamis, nincs inicializálva vagy ellenőrizve
    fpassthru ("./nagyon/kenyes/adatok/index.html");
}
?>
```

29. Fejezet. Globálisan is elérhető változók (Register Globals) használata

A legvitatottabb változtatás a PHP-ben talán az, amikor [register_globals](#) direktíva alapértelmezett értéke ON-ról OFF-ra változott a PHP [4.2.0](#)-ban. Erre a direktívról való támaszkodás meglehetősen népszerű volt, sokan még azt sem tudták, hogy létezik és azt feltételezték, hogy ez csupán a PHP működésének tudható be. Ez az oldal azt taglalja, hogyan lehet nem biztonságos kódot írni ezzel a direktívával, de tartsd észben azt is, hogy maga a direktíva nem veszélyes, annál inkább az a helytelen használata.

Amikor a register_globals be van állítva, megmérgezi a szkriptjeidet mindenféle változóval, mint például a HTML formokból származókkal. Ehhez hozzájön az, hogy a PHP nem követeli meg a változók inicializálását, így sokkal könnyeb veszélyes kódot írni. Nehéz döntés volt, de a PHP közösségi úgy döntött, hogy ez a direktíva alap állapotban ki kell legyen kapcsolva. Amikor bekapcsolt állapotban van, úgy használják a változókat, hogy valójában nem is tudják honnan származik, csak feltételezik. A szkriptben definiált változók összekeverednek a felhasználótól érkezőkkel, de a register_globals kikapcsolása ezt megváltoztatja. Itt egy példa, amely a register_globals helytelen használatát szemlélteti:

Példa 29-1. Helytelen használat register_globals = on esetben

```
<?php
// $jogosult = true csak akkor ha a felhasználó sikeresen azonosítva van
if (azonositott_felhasznalo()) {
    $jogosult = true;
}

// Mivel nem inicializáltuk az $authorized változót false-ra, előfordulhat,
// hogy definálva lett a register_globals hatására, mint például
// GET beleptet.php?jogosult=1
// így bárki jogosult lehet érzékeny adatok elérésére.
if ($jogosult) {
    include "/nagyon/erzekeny/adat.php";
}
?>
```

Amikor register_globals = on, a fenti logikánkon kicsit javítani kell. Amikor off, a \$jogosult változó nem állítható be a felhasználótól érkező adatokból, tehát minden rendben, bár valójában a változók inicializálása rendszerint jó programozási gyakorlat. Például a fenti kód részletben előbb egy \$jogosult = false értékadást kellett volna írnunk. Ha így teszünk, a kódunk működik a register_globals on és off értékével egyaránt, azaz a felhasználók alapból jogosulatlanok lesznek.

Egy másik példa a [munkamenetkezelés](#)hez kötődik. Amikor register_globals = on, az alábbi példában használhatnánk egy \$usernev változót, de gondolj bele, hogy a \$usernev máshonnan is származhat, mint például GET paraméterből (URL-en keresztül).

Példa 29-2. Példa munkamenetek használatára, register_globals on vagy off

```
<?php
// Nem tudhatjuk, hogy a $usernev honnan származik, de azt igen, hogy
// a $_SESSION a munkamenet adatokat hivatkozza
if (isset($_SESSION['usernev'])) {

    echo "Hello, <b>{$_SESSION['usernev']}</b>";
}

} else {

    echo "Hello, <b>Vendég</b><br />";
    echo "Szeretnél belépní?";

}
?>
```

Okosan használva, még azt képes lehet jelezni, ha hamisítást kíséreltek meg. Ha előre tudható, hogy mely változóknak honnan kell érkezniük, akkor azt is megvizsgálhatod, hogy vajon más módon nem próbálták-e elküldeni ezt a változót. Ez nem garantálja, hogy az adatok nem hamisíthatók, azonban megköveteli a támadótól, hogy az rátaláljon a megfelelő hamisítási módszerre. Az is lehetséges, hogy megelőző intézkedéseket tegyük, hogy figyelmeztetést kapjunk, amikor hamisítást kíséreltek meg. Ha előre tudható, hogy mely változóknak pontosan honnan kell érkezniük, akkor azt is megvizsgálhatod, hogy vajon más módon nem próbálták-e elküldeni ezt a változót. Ez nem

garantálja, hogy az adatok nem hamisíthatók, azonban megköveteli a támadótól, hogy az rátaláljon a megfelelő hamisítási módszerre. Ha nem számít, hogy a beküldött adat honnan jön, használhatod a `$_REQUEST`-et, mivel ez tartalmazza a GET, POST és COOKIE adatok keverékét. Lásd még a kézikönyv [PHP-n kívüli változók](#) használatáról szóló részt.

Példa 29-3. Egyszerű "változómérgezés" feltárása

```
<?php
if (isset($_COOKIE['MAGIC_COOKIE'])) {
    // A MAGIC_COOKIE cookie-ból érkezik.
    // Ne felejtsd el ellenőrizni a cookie adatot!

} elseif (isset($_GET['MAGIC_COOKIE']) || isset($_POST['MAGIC_COOKIE'])) {
    mail("admin@peelda.hu", "Lehetseges betoresi probalkozas",
        $_SERVER['REMOTE_ADDR']);
    echo "Betörési próbálkozás. Adminisztrátor értesítve.";
    exit;

} else {
    // A MAGIC_COOKIE nincs beállítva ebben a kérésben
}
?>
```

A register_globals kikapcsolása természetesen nem jelenti azt, hogy a kódod biztonságos. minden beérkező adatot valamilyen egyéb módon is ellenőrizni kell. Mindig ellenőrizd a felhasználótól érkező adatokat és inicializáld a változóidat! Az inicializálatlan változók felfedezéséhez bekapsolhatod az [error_reporting\(\)](#)-ot hogy kiírja az E_NOTICE szintű hibákat.

A [FAQ](#)-ban olvashatsz arról, hogyan lehet emulálni a register_globals On vagy Off értékét.

A szuperglobális változók használhatósága: A 4.1.0 változattól kezdődően bevezetésre kerültek olyan szuperglobális hatókörű tömbök, mint a `$_GET`, `$_POST`, `$_SERVER`, stb. További infórmációk a [superglobals](#) oldalon olvashatók.

30. Fejezet. Felhasználótól érkező adatok

A legtöbb probléma sok PHP programban nem a nyelvben rejlik, hanem abból fakad, hogy a kód nem a biztonságosságot szem előtt tartva készült. Emiatt minden kellő időt kell szánni annak ellenőrzésére, hogy egy adott kódrészletre milyen hatással lehet egy váratlan hibás adat.

Példa 30-1. Veszélyes változóhasználat

```
<?php
// egy fájl törlése a user könyvtárából... vagy
// talán valaki másából?
unlink ($ordogi_valtozo);

// Az elérés naplózása... vagy egy /etc/passwd bejegyzésé?
fputs ($fp, $ordogi_valtozo);

// Valami egyértelmű dolog futtatása.. vagy rm -rf *?
system ($ordogi_valtozo);
exec ($ordogi_valtozo);
?>
```

Minden alaposan meg kell vizsgálni a felhasználók által beadott adatokat, feltéve a következő

kérdéseket:

- Biztos, hogy ez a szkript csak a kívánt fájlokat fogja módosítani?
- Előfordulhat egy ponton, hogy szokatlan vagy nem kívánatos adat jelenjen meg?
- Használható-e az adott szkript nem kívánatos formában?
- Felhasználható-e más szkriptekkel együtt egy negatív hatás elérésére?
- Megfelelően naplózásra kerülnek-e a tranzakciók (elérések, változtatások)?

Kellőképpen átgondolva a fenti kérdéseket a szkript írásakor, megkímélhet attól, hogy később végig gondolva a problémákat szerencsétlen módon újra kelljen írni a teljes kódot a védelem növelése érdekében. Ezzel sem lehet garantálni a rendszer biztonságát, de segíthet annak növelésében/ fenntartásában.

Számításba lehet venni a register_globals, magic_quotes és más kényelmi szolgáltatások kikapcsolásának a gondolatát is, mivel ezek megfosztanak az adatok forrásának, helyességének, tartalmának ismeretétől. A PHP*t maximális hibajelentési szinten használva - az error_reporting E_ALL beállításával - figyelmeztetést ad előzetes érték nélküli, definiálatlan változókról, ezzel védve attól, hogy véletlenül hibás adatokkal dolgozzon a program.

31. Fejezet. Magic Quotes

Magic Quotes is a process that automatically escapes incoming data to the PHP script. It's preferred to code with magic quotes off and to instead escape the data at runtime, as needed.

What are Magic Quotes

When on, all ' (single-quote), " (double quote), \ (backslash) and NULL characters are escaped with a backslash automatically. This is identical to what [addslashes\(\)](#) does.

There are three magic quote directives:

- [magic_quotes_gpc](#)

Affects HTTP Request data (GET, POST, and COOKIE). Cannot be set at runtime, and defaults to *on* in PHP.

See also [get_magic_quotes_gpc\(\)](#).

- [magic_quotes_runtime](#)

If enabled, most functions that return data from an external source, including databases and text files, will have quotes escaped with a backslash. Can be set at runtime, and defaults to *off* in PHP.

See also [set_magic_quotes_runtime\(\)](#) and [get_magic_quotes_runtime\(\)](#).

- [magic_quotes_sybase](#)

If enabled, a single-quote is escaped with a single-quote instead of a backslash. If on, it completely overrides [magic_quotes_gpc](#). Having both directives enabled means only single quotes are escaped as ". Double quotes, backslashes and NULL's will remain untouched and unescaped.

See also [ini_get\(\)](#) for retrieving its value.

Why use Magic Quotes

- Useful for beginners

Magic quotes are implemented in PHP to help code written by beginners from being dangerous. Although [SQL Injection](#) is still possible with magic quotes on, the risk is reduced.

- Convenience

For inserting data into a database, magic quotes essentially runs [addslashes\(\)](#) on all Get, Post, and Cookie data, and does so automagically.

Why not to use Magic Quotes

- Portability

Assuming it to be on, or off, affects portability. Use [get_magic_quotes_gpc\(\)](#) to check for this, and code accordingly.

- Performance

Because not every piece of escaped data is inserted into a database, there is a performance loss for escaping all this data. Simply calling on the escaping functions (like [addslashes\(\)](#)) at runtime is more efficient.

Although `php.ini-dist` enables these directives by default, `php.ini-recommended` disables it. This recommendation is mainly due to performance reasons.

- Inconvenience

Because not all data needs escaping, it's often annoying to see escaped data where it shouldn't be. For example, emailing from a form, and seeing a bunch of '\' within the email. To fix, this may require excessive use of [stripslashes\(\)](#).

Disabling Magic Quotes

The [magic_quotes_gpc](#) directive may only be disabled at the system level, and not at runtime. In otherwords, use of [ini_set\(\)](#) is not an option.

Példa 31-1. Disabling magic quotes server side

An example that sets the value of these directives to *Off* in `php.ini`. For additional details, read the manual section titled [How to change configuration settings](#).

```
; Magic quotes
;

; Magic quotes for incoming GET/POST/Cookie data.
magic_quotes_gpc = Off

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
magic_quotes_sybase = Off
```

If access to the server configuration is unavailable, use of `.htaccess` is also an option. For example:

```
php_flag magic_quotes_gpc Off
```

In the interest of writing portable code (code that works in any environment), like if setting at the server level is not possible, here's an example to disable [magic_quotes_gpc](#) at runtime. This method is inefficient so it's preferred to instead set the appropriate directives elsewhere.

Példa 31-2. Disabling magic quotes at runtime

```
<?php
if (get_magic_quotes_gpc()) {
    function stripslashes_deep($value)
    {
        $value = is_array($value) ?
            array_map('stripslashes_deep', $value) :
            stripslashes($value);

        return $value;
    }

    $_POST = array_map('stripslashes_deep', $_POST);
    $_GET = array_map('stripslashes_deep', $_GET);
    $_COOKIE = array_map('stripslashes_deep', $_COOKIE);
    $_REQUEST = array_map('stripslashes_deep', $_REQUEST);
}
?>
```

32. Fejezet. A PHP elrejtése

Általában a rejtegetésen és félrevezetésen alapuló védelem az egyik leggyengébb formája a védekezésnek, azonban néha minden apró többlet kívánatos lehet.

Néhány egyszerű módszerrel elrejthető, hogy PHP-t használsz, így lassítva le a támadót, aki fel akarja deríteni a rendszer gyenge pontjait. A `php.ini`-ben az `expose_php = off` beállítással csökkentheted ezeket az információkat.

Másik taktika a webszerver (pl. apache) olyan beállítása a `.htaccess`-en vagy az apache konfigurációs fájlában, hogy különböző típusú fájlokat is PHP-n keresztül futtasson. Más félrevezető fájlkiterjesztések alkalmazására példa:

Példa 32-1. PHP elrejtése más nyelvként

```
# úgy tűnik, mintha PHP helyett más szerver oldali alkalmazás futna
AddType application/x-httpd-php .asp .py .pl
```

vagy egy teljesen ismeretlennel is eltakarható:

Példa 32-2. ismeretlen típusok használata PHP fájlok kiterjesztéseként

```
# PHP kódok teljesen ismeretlen típusúnak tűnnek
AddType application/x-httpd-php .bop .foo .133t
```

vagy html típus mögé is elrejthetők a PHP fájlok, ami csekély teljesítménycsökkenést okoz, mivel minden html oldal átmegy a PHP-n:

Példa 32-3. html típus használata PHP fájlkiterjesztésként

```
# PHP kódok html típusúnak tűnnek
AddType application/x-httpd-php .htm .html
```

Ahhoz, hogy ez jól működjön, minden PHP fájlt át kell nevezni a fenti kiterjesztések valamelyikének megfelelően. Mivel ez is az elrejtőzésen / ismeretlenségen alapuló védelmi forma, kevés hátránya mellett csekély megakadályozó intézkedést jelent.

33. Fejezet. Fontos aktuálisnak maradni

A PHP, mint bármilyen más nagy rendszer állandóan változások és fejlesztések alatt áll. minden új változat kisebb-nagyobb változtatásokat tartalmaz, feljesztve a nyelvet, kijavítva biztonsági hibákat, beállítási kellemetlenségeket, és más olyan elemeket, amik a teljes rendszer stabilitására és biztonságára hatnak.

Mint más rendszerszintű nyelvek és programok esetében, a legjobb hozzáállás a gyakori frissítés, valamint a friss változatokról, és a fellépő változásokról való informálódás.

V. Szolgáltatások

Tartalom

- 34. [HTTP hitelesítés PHP-vel](#)
- 35. [Sütik \(cookie-k\)](#)
- 36. [Sessions](#)
- 37. [Dealing with XForms](#)
- 38. [Fájlfeltöltés kezelése](#)
- 39. [Távoli állományok kezelése](#)
- 40. [Kapcsolatkezelés](#)
- 41. [Állandó adatbázis kapcsolatok](#)
- 42. [Safe Mode](#)
- 43. [Parancssori programozás a PHP-ben](#)

34. Fejezet. HTTP hitelesítés PHP-vel

A HTTP hitelesítési (authentication) funkciók csak akkor elérhetőek PHP-ben, ha az Apache modulként fut, bár régebben ez a CGI módú használat során is működött, azonban ez a lehetőség már megszűnt. Az Apache modulként futó PHP esetében a [header\(\)](#) függvényt kell használni arra, hogy egy "Authentication Required" üzenetet küldjön a kliens böngészőnek, aminek hatására az egy Username/Password bemeneti ablakot nyit meg a felhasználó számára. Ha a látogató kitöltötte a username és password mezőket, az URL, ami a PHP szkriptre mutat, ismét meghívásra kerül, és rendelkezésre állnak a *PHP_AUTH_USER*, *PHP_AUTH_PW* és a *AUTH_TYPE* változók, amik a felhasználói név, jelszó és azonosítási típus értékeit tartalmazzák értelemszerűen. Mind a "Basic", mind a "Digest" (PHP 5.1.0 óta) azonosítási típus támogatott. További információt a [header\(\)](#) függvény dokumentációjában találsz.

PHP verzióhoz kötődő megjegyzés: Az [autoglobális](#) változók, mint például a *\$_SERVER*, a PHP [4.1.0](#) változatától elérhetőek. PHP 3-ban helyette a *HTTP_SERVER_VARS* használható.

Egy egyszerű példa PHP szkript, ami kliens azonosítást vált ki:

Példa 34-1. HTTP azonosítási példa (Basic)

```
<?php
if( !isset($_SERVER['PHP_AUTH_USER']) ) {
    header('WWW-Authenticate: Basic realm="Azonosítsd magad!"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Ez jelenik meg, ha a Cancel gombot nyomja a felhasználó';
    exit;
} else {
    echo "<p>Helló {$_SERVER['PHP_AUTH_USER']}.</p>";
    echo "<p>A megadott jelszavad: {$_SERVER['PHP_AUTH_PW']}</p>";
}
```

```
?>
```

Példa 34-2. HTTP azonosítási példa (Digest)

Az alábbi példamutatja be, hogy lehet összeállítani egy Digest típusú HTTP azonosítást végző szkriptet. A dolog

```
<?php
$realm = 'Ide csak úgy ben nem lépsz!';

// felhasználónév => jelszó
$felhasznalok = array('admin' => 'mypass', 'guest' => 'guest');

if (!isset($_SERVER['PHP_AUTH_DIGEST'])) {
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Digest realm="'. $realm .
        '" qop="auth" nonce="'. uniqid() .'" opaque="'. md5($realm) .'"');
    die('Ez a szöveg jelenik meg, ha a túloldalon Mégsemet (Cancel) nyomtak');
}

// A PHP_AUTH_DIGEST-ben kapottak analizálása
preg_match('/username="(?P<username>.*)"', \s*realm="(?P<realm>.*)"', \s*nonce="(?P<nonce>.*")'

if (!isset($felhasznalok[$digest['username']]))
    die('A felhasználónév nem megfelelő!');

// generate the valid response
$A1 = md5($digest['username'] . ':' . $realm . ':' . $felhasznalok[$digest['username']]);
$A2 = md5($_SERVER['REQUEST_METHOD'] . ':' . $digest['uri']);
$valid_response = md5($A1 . ':' . $digest['nonce'] . ':' . $digest['nc'] . ':' . $digest['cnonce'] . ':';

if ($digest['response'] != $valid_response)
    die('Az azonosítás sikertelen!');

// minden oké, jó a felhasználónév / jelszó páros
echo 'Beléptél ezen a néven: ' . $digest['username'];

?>
```

Megjegyzés: Vigyázni kell a HTTP fejlécek írásakor! A maximális kompatibilitás eléréséhez a "Basic" kulcsszót nagy B betűvel kell kezdeni, a "realm" részt mindenkorban idézőjelbe (és nem aposztrófok közé) kell tenni! Végül pontosan egy szóközt hagy ki a "401" előtt a "HTTP/1.0 401" fejléc sorban.

Egy valós esetben persze nem a \$PHP_AUTH_USER és \$PHP_AUTH_PW kiírása az elérni kívánt cél, így általában a felhasználói név és jelszó ellenőrzése következik. Természetesen lehetőség van ezt egy adatbázis lekérdezéssel megoldani, vagy egy dbm fájlban utánanézni a szükséges adatoknak.

Figyelj az Internet Explorer böngészők hibáira, ezek nem fogadják el tetszőleges sorrendben a HTTP fejléceket. A tesztek azt mutatják, hogy a *WWW-Authenticate* elküldése a *HTTP/1.0 401* előtt megoldja a problémát.

Mivel a hagyományos HTTP azonosítás során a PHP 4.3.0-ás változatától kezdődően a jelszó rejtett az elért szkript előtt, a PHP nem állítja be a PHP_AUTH változókat ha az adott file-ra ha hagyományos azonosítás is engedélyezett és a ??? bekapcsolt állapotban van. Így nem deríthető ki a user jelszava. Ebben az esetben a *REMOTE_USER* változó tartalmazza a már azonosított felhasználó nevét, azaz *\$_SERVER['REMOTE_USER']*-ként férhetünk hozzá.

Beállítási megjegyzés: A PHP az *AuthType* direktíva megléte alapján dönti el, hogy rajta kívülálló azonosítás történik-e.

Vedd észre, hogy ez nem küszöböli ki azt a problémát, hogy más nem azonosítás-köteles URL címeken lévő szkriptek ugyanazon a szerveren megszerezzék a jelszavakat az azonosított URL-ekről.

A Netscape Navigator és Internet Explorer böngészők törlni fogják a böngésző adott oldalhoz tartozó azonosítási tárát (authentication cache), amennyiben egy 401-es szerver üzenetet kapnak. Ez gyakorlatilag kilépteti a user-t, ami azt jelenti, hogy legközelebb ismét meg kell adnia a nevét és jelszavát. Időnként ezt arra használják, hogy lejáratú időt rendelve a belépésekhez egy idő után megszüntessék azokat, vagy egy kilépés gombot biztosítsanak.

Példa 34-3. HTTP azonosítási példa, ami új nevet és jelszót kér

```
<?php
function azonositas() {
    header( "WWW-Authenticate: Basic realm=\"Azonosítási Rendszer Teszt\"");
    header( "HTTP/1.0 401 Unauthorized");
    echo "Érvényes nevet és jelszót kell megadnod, hogy elérд ezt a szolgáltatást!\n";
    exit;
}
if (!isset($_SERVER['PHP_AUTH_USER'])) ||
    ($_POST['SeenBefore'] == 1 && $_POST['OldAuth'] == $_SERVER['PHP_AUTH_USER'])) {
    azonositas();
}
else {
    echo "<p>Üdv néked, {$_SERVER['PHP_AUTH_USER']}<br>";
    echo "Régebben: {$_REQUEST['OldAuth']}</p>";
    echo "<form action='{$_SERVER['PHP_SELF']}' method='post'>\n";
    echo "<input type='hidden' name='JartMarItt' value='1' />\n";
    echo "<input type='hidden' name='RegiUser' value='{$_SERVER['PHP_AUTH_USER']}' />\n";
    echo "<input type='submit' value='Újraazonosítás' />\n";
    echo "</form></p>\n";
}
?>
```

A HTTP Basic azonosítási standard nem követeli meg ezt a viselkedést a böngészők részéről, tehát ne építs rá! Lynx-el végzett tesztek azt mutatták, hogy a Lynx nem törli az azonosítási bizonnyáványokat a 401-es szerver válasz hatására, tehát egy back és forward lépéssel ismét megnyílik az oldal, feltéve, hogy az azonosítási feltételek nem változtak. Ellenben a felhasználó megnyomhatja a '_ billentyűt, hogy törölje az azonosítási információkat.

Szintén fontos megjegyezni, hogy ez a módszer nem vezet eredményre (A PHP 4.3.3 változatáig bezárólag), ha Microsoft IIS szervert használ sz CGI módú PHP-val, az IIS korlátai miatt. Ahhoz, hogy ezt a PHP 4.3.3 változatától kezdve használhasd, bele kell kicsit nyúlni az IIS beállításaiba. A "Directory Security" beállításainál kattints az "Edit" gombra és csak az "Anonymous Access"-t hagyd bekapcsolva.

Egy másik korlátozása az IIS (ISAPI) modulnak, hogy a *PHP_AUTH_** változók nem kapnak értéket. Helyette a *HTTP_AUTHORIZATION* áll rendelkezésre. Egy kis példakód ötletképpen: *list(\$user, \$pw) = explode(':', base64_decode(substr(\$_SERVER['HTTP_AUTHORIZATION'], 6)));*

Megjegyzés az IIS kapcsán:: Ahhoz, hogy a HTTP azonosítás működjön IIS alatt, 0-ra kell állítani a [cgi.rfc2616_headers](#) PHP konfigurációs direktíva értékét (ez az alapértelmezett értéke).

Megjegyzés: Ha [safe mode](#) be van kapcsolva, akkor a *WWW-Authenticate* fejléc *realm* részéhez a szkript UID-ja is hozzáadódik. the header.

35. Fejezet. Sütik (cookie-k)

A PHP támogatja a HTTP cookie-k kezelését. A sütik lehetőséget adnak arra, hogy adatokat tárolj a kliens gépen, így követve vagy azonosítva a visszatérő látogatókat. Sütik beállítására a [setcookie\(\)](#) vagy a [setrawcookie\(\)](#) függvénytellyel nyílik lehetőség. A sütik részei a HTTP fejlécnek, így a [setcookie\(\)](#) függényt azelőtt kell meghívni, mielőtt bármilyen kimenetet küldesz a böngészőnek. Ez a [header\(\)](#) függvénytellyel megegyező korlátozást jelent. A [kimenet szabályozó függvényeket](#) használhatod a kimenet késleltetésére addig, amíg minden sütit és fejlécet elküldtél.

Minden süti, amit a klienstől visszakapsz, automatikusan PHP változóvá válik, pont úgy, mint a *GET* és a *POST* kérésekkel érkező adatok, feltéve, hogy a [register_globals](#) és [variables_order](#) php.ini beállítások ennek megfelelően vannak beállítva. Ha több értéket szeretnél adni egy sütinek, add a szokásos *//* végződést a süti nevéhez.

A PHP 4.1.0 és későbbi változataiban a *\$_COOKIE* nevű mindenhonnan látható változó mindig létrejön, tartalmazva a klienstől érkezett sütiket. A *\$HTTP_COOKIE_VARS* a korábbi verziókban használható, ha a [track_vars](#) php.ini beállítás be van kapcsolva, bár ez a változó nem látszik mindenhonnan. (A 4.0.3-as változattól felfele ez minden be van kapcsolva)

Részletesebb információk, beleértve a böngésző hibákat, a [setcookie\(\)](#) [setrawcookie\(\)](#) függvények dokumentációs oldalain olvashatók.

36. Fejezet. Sessions

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site. All information is in the [Session reference](#) section.

37. Fejezet. Dealing with XForms

[XForms](#) defines a variation on traditional webforms which allows them to be used on a wider variety of platforms and browsers or even non-traditional media such as PDF documents.

The first key difference in XForms is how the form is sent to the client. [XForms for HTML Authors](#) contains a detailed description of how to create XForms, for the purpose of this tutorial we'll only be looking at a simple example.

Példa 37-1. A simple XForms search form

```
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
         xmlns="http://www.w3.org/2002/xforms">
  <h:head>
    <h:title>Search</h:title>
    <model>
      <submission action="http://example.com/search"
                  method="post" id="s"/>
    </model>
  </h:head>
  <h:body>
    <h:p>
      <input ref="q"><label>Find</label></input>
      <submit submission="s"><label>Go</label></submit>
    </h:p>
  </h:body>
```

```
</h:html>
```

The above form displays a text input box (named *q*), and a submit button. When the submit button is clicked, the form will be sent to the page referred to by *action*.

Here's where it starts to look different from your web application's point of view. In a normal HTML form, the data would be sent as *application/x-www-form-urlencoded*, in the XForms world however, this information is sent as XML formatted data.

If you're choosing to work with XForms then you probably want that data as XML, in that case, look in *\$HTTP_RAW_POST_DATA* where you'll find the XML document generated by the browser which you can pass into your favorite XSLT engine or document parser.

If you're not interested in formatting and just want your data to be loaded into the traditional *\$_POST* variable, you can instruct the client browser to send it as *application/x-www-form-urlencoded* by changing the *method* attribute to *urlencoded-post*.

Példa 37-2. Using an XForm to populate *\$_POST*

```
<h:html xmlns:h="http://www.w3.org/1999/xhtml"
         xmlns="http://www.w3.org/2002/xforms">
<h:head>
  <h:title>Search</h:title>
  <model>
    <submission action="http://example.com/search"
                method="urlencoded-post" id="s"/>
  </model>
</h:head>
<h:body>
  <h:p>
    <input ref="q"><label>Find</label></input>
    <submit submission="s"><label>Go</label></submit>
  </h:p>
</h:body>
</h:html>
```

Megjegyzés: As of this writing, many browsers do not support XForms. Check your browser version if the above examples fails.

38. Fejezet. Fájlfeltöltés kezelése

POST metódusú feltöltések

Ez a szolgáltatás egyaránt lehetővé teszi a látogatónak szöveges és bináris fájlok feltöltését. A PHP azonosítási és fájlkezelési képességeivel teljes felügyeletet lehet gyakorolni afelett, hogy ki tölthet fel állományokat, és azokkal mi történjen.

A PHP alkalmas fájl feltöltést fogadni bármilyen RFC-1867 kompatibilis böngészőtől (mint a Netscape Navigator 3 vagy későbbi és a Microsoft Internet Explorer 3 Microsoft javítással, vagy későbbi IE javítás nélkül).

Kapcsolódó konfigurációs megjegyzés: Lásd még: [file_uploads](#), [upload_max_filesize](#), [upload_tmp_dir](#), [post_max_size](#) és a [max_input_time](#) direktívákat a `php.ini`-ben!

A PHP támogatja a PUT metódust is, amit a Netscape Composer és a W3C Amaya kliensek használnak. Lásd a [PUT metódusú feltöltések](#) részét.

Példa 38-1. Állományfeltöltő űrlap

Az állomány feltöltési lehetőség egy különleges módon kialakított űrlappal biztosítható, amely nagyjából így néz ki:

```
<!-- Az adatkódolás típusát meg kell adni az enctypeban, ahogy a példa is mutatja -->
<form enctype="multipart/form-data" action="_URL_" method="post">
  <!-- A MAX_FILE_SIZE meg kell előzze a fájlkiválasztó űrlapelemet -->
  <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
  <!-- A name-ben megadott név szerinti tömbelemben kapunk adatokat a $_FILES tömbben -->
  Állomány elküldése: <input name="userfile" type="file" />
  <input type="submit" value="OK" />
</form>
```

A példában szereplő *_URL_* a feldolgozást végző PHP fájra kell, hogy mutasson.

A *MAX_FILE_SIZE* rejtett mező a file típusú input mező előtt kell, hogy szerepeljen, és azt adja meg, hogy mekkora a maximális fájl méret (byte-okban megadva), amit a kliens jogosult feltölteni. Ez csak egy javasolt érték a böngészők számára, amit a PHP még szintén felülvizsgálhat. A böngésző oldalon ezt az értéket igen könnyű megnövelni, ezért céleszerű minden esetben a PHP-vel is a fájlok körmére nézni a méret tekintetében. Ellentétben az űrlapban beállított a PHP beállítása szerinti maximum méretet nem lehet semmi esetre sem túllépni semmilyen trükkkel sem. Mindenesetre érdemes ezt a maximum értéket az űrlapba helyezni, hogy az átlagfelhasználó ne várjon feleslegesen perceket arra hogy kiderüljön, tűl nagy fájlt akar a szerverre tukmálni.

Megjegyzés: Fontos, hogy a fájlfeltöltő űrlapokban szerepeljen a *enctype="multipart/form-data"* meghatározás is, ellenkező esetben a feltöltés nem fog működni.

A PHP 4.1.0-ás változata óta létezik a *\$_FILES* globálisan elérhető tömb (korábbi változatokban használ a *\$_HTTP_POST_FILES* tömböt). Ez a tömb tartalmazza a feltöltött fájlok minden adatát.

A *\$_FILES* tartalma a fenti példa alapján a következő. Megjegyezző, hogy az alábbi felsorolás arra épít, hogy a az állomány feltöltő mező neve *userfile*, ahogy a fenti példában látható. Ennek természetesen mi bármilyen más nevet is adhatunk.

\$_FILES['userfile']['name']

Az állomány eredeti neve a távoli kliensgépen.

\$_FILES['userfile']['type']

A feltöltött állomány MIME típusa, ha a böngésző átadta ezt az információt, pl.: "image/gif".

\$_FILES['userfile']['size']

A feltöltött állomány mérete bajtokban.

\$_FILES['userfile']['tmp_name']

Annak az ideiglenes állománynak a neve, amely a szerveren tárolja a feltöltött állomány tartalmát.

\$_FILES['userfile']['error']

Az állomány feltöltés során keletkezett [hiba kódja](#). A PHP 4.2.0 változatától használható.

Az állományok alapbeállításban a szerver szokásos ideiglenes könyvtárában tárolódnak, ha nem

adtál meg másat az `upload_tmp_dir` beállítással a `php.ini` fájlban. A szerver alapbeállítású könyvtára megváltoztatható a `TMPDIR` környezeti változóval abban a környezetben, ahol a PHP fut. PHP szkriptből a `putenv()`-el való átállítás nem működik. Ez a környezeti változó annak ellenőrzésére is használható, hogy más műveletek is végezhetőek-e a feltöltött állományokon.

Példa 38-2. Fájlfeltöltések ellenőrzése

Érdemes még rátekinteni az `is_uploaded_file()` és a `move_uploaded_file()` függvényekre is. Az itt következő példa egy űrlap által kezdeményezett fájlfeltöltést fog lekezelni.

```
<?php
// A PHP 4.1.0 előtti verzióiban a $_FILES helyett a
// $HTTP_POST_FILES használandó

$uploaddir = '/var/www/uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);

echo '<pre>';
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
    echo "A fájl megfelelő. sikeresen feltöltésre került.\n";
} else {
    echo "Lehetséges fájlfeltöltés-támadás!\n";
}

echo 'Íme egy kis információ hibakereséshez:';
print_r($_FILES);

echo '</pre>';

?>
```

A PHP programnak, amely megkapja a feltöltött állományt, gondoskodnia kell arról is, hogy a kívánt műveleteket elvégezze az állománnyal. Például törölheti azt, ha az túl nagy, vagy túl kicsi, figyelembe véve a `$_FILES['userfile']['size']` változó értékét, vagy meghatározhatja a `$_FILES['userfile']['type']` alapján, hogy ez a fájl megfelel-e egy meghatározott fájltípusnak, és ha nem, törölheti. PHP 4.2.0-tól kezdve a `$_FILES['userfile']['error']` használható arra, hogy a **hibakódoknak** megfelelően változzon szkript működését. Bármilyen legyen a cél a feltöltött állománnyal, a PHP szkriptnek kell gondoskodnia arról, hogy elmozgassa egy biztonságos helyre, vagy törölje az ideiglenes könyvtárból az adott ideiglenes állományt.

Ha az űrlapban nem lett fájl kijelölve feltöltésre, a PHP a `$_FILES['userfile']['size']` értékeként 0-át ad vissza, valamint a `$_FILES['userfile']['tmp_name']` tartalma "none" lesz.

Az ideiglenes állomány törlésre kerül az ideiglenes könyvtárból a kérés végrehajtásának végeztével, ha nem lett elmozgatva vagy átnevezve.

Példa 38-3. Példa: Több fájl feltöltése egy tömbben

A PHP támogatja a **HTML adattömbök** használatát fájl típusú űrlapelemek esetén is.

```
<form action="" method="post" enctype="multipart/form-data">
<p>Képek:</p>
<input type="file" name="kepek[]" />
<input type="file" name="kepek[]" />
<input type="file" name="kepek[]" />
<input type="submit" value="Feltöltés" />
</p>
</form>

<?php
foreach ($_FILES["kepek"]["error"] as $key => $error) {
```

```
if ($error == UPLOAD_ERR_OK) {
    $tmp_name = $_FILES["kepek"]["tmp_name"][$key];
    $name = $_FILES["kepek"]["name"][$key];
    move_uploaded_file($tmp_name, "data/$name");
}
?>
```

Hibaüzenetek magyarázatai

PHP 4.2.0-tól kezdve PHP az állományhoz tartozó tömbben a megfelelő hibakódot is visszaadja. Ezt a hibakódot a feltöltéskor létrehozott tömb *error* eleme tárolja. Másképp megfogalmazva, a hiba a *\$_FILES['userfile']['error']* hivatkozás értékeként olvasható ki.

UPLOAD_ERR_OK

érték: 0; nincs hiba, a feltöltés sikeres.

UPLOAD_ERR_INI_SIZE

érték: 1; a feltölteni kívánt fájl túl nagy a *php.ini*-ben megadott upload_max_filesize direktíva által megengedett héz képest.

UPLOAD_ERR_FORM_SIZE

érték: 2; a feltölteni kívánt fájl túl nagy a HTML űrlapban megadott *MAX_FILE_SIZE* értékhez képest.

UPLOAD_ERR_PARTIAL

érték: 3; az állomány csak részben lett feltöltve.

UPLOAD_ERR_NO_FILE

érték: 4; nem történt állomány-feltöltés.

UPLOAD_ERR_NO_TMP_DIR

érték: 6; Az átmeneti tárolókönyvtár nem létezik. (PHP 4.3.10 illetve 5.0.3-tól lett bevezetve)

Megjegyzés: Ezek állandókként is használhatók PHP 4.3.0-tól kezdve.

Tipikus csapdák

A *MAX_FILE_SIZE* nem tartalmazhat nagyobb értéket, mint az upload_max_filesize beállítás értéke. Az alapbeállítás 2 megabyte.

Ha a memóriakorlát be van állítva, esetleg növelni kell a memory_limit értékét. Gondoskodni kell a memory_limit kellően nagyra állításáról!

Ha a max_execution_time túl kicsire van állítva, a program futása során lefelhet az idő. Ezért

gondoskodni kell a *max_execution_time* kellően nagyra állításáról! Az állományok feltöltése nem számít bele a *max_execution_time* időbe, de egy nagyobb állomány feldolgozása lehetséges, hogy több időt vesz igénybe a program számára.

Megjegyzés: A *max_execution_time* csupán csak a szkriptek lefutásának idejére vonatkozik. minden olyan tevékenység, amely a szkript futásán kívül történik, mint például a *system()*, *sleep()* függvényhívás, az adatbázis lekérések feldolgozása adatbázis oldalon, vagy történetesen a fájlfeltöltés ideje nem számít bele a futtatási időbe.

Figyelem

A *max_input_time* adja meg másodpercekben azt a leghosszabb időt, amennyi időt a szkript eltölthet az adatok átvételével. Ilyen a fájlfeltöltés is. Nagyobb állományok feltöltése esetén nem árt, ha a PHP által alapértelmezett 60 másodperces időt nagyobbra állítjuk.

Ha a *post_max_size* túl kicsi, nagy állományok nem tölthetők fel. Gondoskodni kell a *post_max_size* kellően nagyra állításáról!

Ellenőrizni kell minden, hogy pontosan mely állományokon végez műveletet a program. A felhasználók esetleg más könyvtárakhoz is hozzáférhetnek.

A CERN httpd szerver úgy tűnik, hogy eldob minden a klienstől kapott Content-type MIME fejlécben az első szóközt követően. Amíg ez fennáll, a CERN httpd szerver nem fogja támogatni a fájl feltöltéseket.

Köszönhetően a nagyon sokféle állomány elnevezési lehetőségek, konvenciónak nem garantálható, hogy a különleges nevű (pl. szóközt tartalmazó) állományok feltöltése minden megfelelően működik.

A fájl típusú adatok nem keverhetők közös tömbbe más, normál adatokkal (az *ize[]* elnevezésre gondolva itt).

Több állomány egyidejű feltöltése

Lehetséges több állomány egyidejű feltöltése is, az *input* elemek *name* paramétereinek különbözőre állításával.

Úgyszintén lehetőség van több megegyező nevű űrlap elemmel is több állomány feltöltésére. Ebben az esetben a kapcsolódó információkat tömbökben adja vissza a PHP. Ehhez a hagyományos tömbhivatkozást kell alkalmazni, mint minden más űrlapelemnél:

Megjegyzés: Több állomány egyidejű feltöltése a PHP 3.0.10 óta lehetséges.

Példa 38-4. Több fájl egyidejű feltöltése

```
<form action="file-feltolt.php" method="post" enctype="multipart/form-data">
  Az alábbi fájlok elküldése:<br />
  <input name="userfile[]" type="file" /><br />
  <input name="userfile[]" type="file" /><br />
  <input type="submit" value="OK" />
</form>
```

Amikor a fenti űrlap adatai elküldésre kerülnek, a *\$_FILES['userfile']* *\$_FILES['userfile'][name]* és *\$_FILES['userfile']['size']* változók értéket kapnak. A *\$HTTP_POST_FILES* tömbben ugyanezek elérhetőek a PHP 4.1.0 előtti verziókban. Ezek minden számokkal indexelt tömbök a tömbben beküldötteknek megfelelő értékekkel. A PHP 3-ban a *\$HTTP_POST_VARS* használható. Ha a *register_globals* be van kapcsolva, globális változókat is létrehozásra kerülnek.

Páldául ha a */home/test/review.html* és */home/test/xwp.out* állományok kerültek

feltöltésre, akkor a `$_FILES['userfile']['name'][0]` tartalma `review.html` és a `$_FILES['userfile']['name'][1]` tartalma `xwp.out`. Hasonló módon a `$_FILES['userfile']['size'][0]` a `review.html` fájl méretét tartalmazza, stb.

`$_FILES['userfile']['name'][0],` `$_FILES['userfile']['tmp_name'][0],`
`$_FILES['userfile']['size'][0]` és `$_FILES['userfile']['type'][0]` szintén elérhetőek.

PUT metódusú feltöltések

A PUT feltöltés támogatása a PHP 3 és a PHP 4 között megváltozott. PHP 4-ben a PUT-al küldött adatok a szabványos bemenetről olvasva gyűjthetők be.

Példa 38-5. HTTP PUT által feltöltött állományok lementése PHP 4-ben

```
<?php
/* A PUT adatok a szabványos bemeneten érkeznek */
$putdata = fopen("php://stdin", "r");
/* Egy fájl nyitása írásra */
$fp = fopen("myputfile.ext", "w");

/* Egyszerre 1KB olvasása és fájlba írása */
while ($data = fread($putdata, 1024)) {
    fwrite($fp, $data);
}

/* Az adatfolyan és a fájl zárása */
fclose($fp);
fclose($putdata);
?>
```

Megjegyzés: A további bekezdések csupán a PHP 3-ra vonatkoznak már.

A PHP támogatja a HTTP PUT metódust is, amit például a Netscape Composer és a W3C Amaya használ. A PUT kérések sokkal egyszerűbbek, mint az eddig tárgyalt feltöltések. A következőképpen néz ki:

```
PUT /eleresi/ut/filenev.html HTTP/1.1
```

Ez hagyományosan azt jelenti, hogy a kliens a küldött adatokat az `/eleresi/ut/filenev.html` fájlba szeretné elmenteni a webgyökér alatt. Az nyilvánvalóan nem lenne jó megoldás az Apache vagy a PHP részéről, ha bárkinek megengendné, hogy felülírja a fájlokat a web könyvtáradban. Éppen ezért a PUT kérések kezeléséhez be kell állítani a webszerver számára, hogy egy PHP szkriptnek küldje az ilyen bemenetet. Apache alatt ezt a *Script* direktívával teheted meg. Ez elhelyezhető szinte minden ponton az Apache konfigurációs fájlodban. Egy gyakori hely erre egy `<Directory>` blokk belseje, vagy esetleg egy `<Virtualhost>` blokk belseje. Például egy ilyen sor megoldja a feladatot:

```
Script PUT /put.php
```

Ez beállítja az Apache számára a PUT kérések kezelésére a `put.php`-t abban a környezetben, ahol ezt a sort elhelyezted a konfiguráláskor. Ez természetesen feltételezi, hogy a `.php` kiterjesztést a PHP kezeli és a PHP aktív.

A `put.php` fájlból aztán valami hasonlót tehetsz:

```
<?php copy($_POST[FILE_NAME], $DOCUMENT_ROOT . $REQUEST_URI); ?>
```

Ez a kérés által meghatározott helyre másolja a küldött fájlt. Valós helyzetben természetesen szükséges valamilyen ellenőrzés, és/vagy felhasználóazonosítás, mielőtt esetleg felülírod egyik

fontos fájlodat. A PHP a [POST metódushoz](#) hasonlóan egy ideiglenes fájlban tárolja a feltöltött fájlt. Amikor a kérés teljesítése befejeződött, ez az ideiglenes fájl törlődik. Ez azt jelenti, hogy a PUT kéréseket feldolgozó szkripteknek ezt a fájlt el kell mozgatnia másra, ha meg szeretné tartani a feltöltött fájlt. Az ideiglenesen létrehozott fájl elérési útját a fájl nevével a `$PHP_PUT_FILENAME` változó tartalmazza, és a javasolt célt a `$REQUEST_URI` változó tartalmazza (bár ez lehet más is nem Apache szervereken). Ez a cél az, amit a kliens meghatározott. Neked nem kell feltétlenül ezt a helyet elfogadnod, lehet, hogy neked az a kényelmesebb (és biztonságosabb), hogy a feltöltött fájlokat egy speciális upload könyvtárban tárolod.

39. Fejezet. Távoli állományok kezelése

Amennyiben az `allow_url_fopen` be van kapcsolva a `php.ini`-ben, HTTP és FTP URL-eket lehet paraméterként átadni majdnem minden olyan függvénynek, amelyek fájlnevet kér paraméterül, beleértve az [include\(\)](#), [include_once\(\)](#), [require\(\)](#) és [require_once\(\)](#) utasításokat is. További információkért a használható protokollokkal kapcsolatban nézz el ide: [M Függelék](#).

Megjegyzés: PHP 4.0.3 és régebbi verziókban az ilyen URL-ek értelmezéséhez szükséges a `--enable-url-fopen-wrapper` beállítás bekapsolása.

Megjegyzés: A Windows-os, 4.3 előtti változatok PHP nem támogatják a távoli állományelérést a következő függvények esetén: [include\(\)](#), [include_once\(\)](#), [require\(\)](#) és [require_once\(\)](#), valamint az `imagecreatefromXXX` függvények esetén. Ezekről többet: [LXII. Képmanipuláló függvények Referencia](#)

Ezt a lehetőséget lehet használni például egy távoli webszerveren lévő fájlt megnyitására, majd a kívánt adatok kigyűjtésére, vagy arra, hogy csak egyszerűen a saját oldalad kinézetével, stílusával tálald.

Példa 39-1. Egy távoli weboldal címsorának megállapítása

```
<?php
$file = fopen( "http://www.example.com/", "r" );
if (! $file) {
    echo "<p>Nem lehet megnyitni a külső file-t!\n";
    exit;
}
while (!feof ($file)) {
    $line = fgets( $file, 1024 );
    /* Ez csak akkor jó, ha a cím és a körbezáró tag-ek egy sorban vannak */
    if ( eregi( "<title>(.*)</title>", $line, $out ) ) {
        $title = $out[1];
        break;
    }
}
fclose($file);
?>
```

Lehetőség van egy FTP szerveren tárolt fájlba írásra is, feltéve, hogy megfelelő jogokkal rendelkező user-ként lépsz be. Ezzel a módszerrel csak új fájlok hozhatók létre, ha már létezik a megadott nevű állomány akkor a [fopen\(\)](#) hívása sikertelen lesz. Ha nem 'anonymous' felhasználóként szeretné belépní, a felhasználói nevet és jelszót az URL részeként kell megadni a alábbi formában: 'ftp://felhasznalo:jelszo@ftp.pelda.hu/eleresi/ut/alma.txt'. (Ugyanezt a módszert használható akkor is, ha olyan állományokat kell elérni HTTP-n keresztül, amelyek a Basic azonosítást igénylik.)

Példa 39-2. Adat tárolása távoli gépen

```
<?php
```

```
$file = fopen( "ftp://ftp.example.com/incoming/outputfile", "w" );
if (! $file) {
    echo "<p>Nem lehet megnyitni a külső file-t írásra.\n";
    exit;
}
/* Itt írunk a file-ba */
fwrite( $file, "$_SERVER['HTTP_USER_AGENT']\n" );
fclose( $file );
?>
```

Megjegyzés: A fenti példa alapján talán azt hihetnénk, hogy ilyen technikát kell használni például távoli naplózáshoz. Sajnálatos módon azonban ez nem működik, mert a [fopen\(\)](#) hívása sikertelen lesz, ha a távoli állomány már létezik. Az ehhez hasonló elosztott, távoli naplózáshoz a [syslog\(\)](#) függvény szolgáltatásait kell igénybe venni.

40. Fejezet. Kapcsolatkezelés

Megjegyzés: Az alábbi fejezetek csak a PHP 3.0.7-es és későbbi verzióira vonatkoznak!

A PHP futása közben nyilvántartja a kapcsolati státuszt. Hárrom lehetséges állapotot van:

- 0 - NORMAL (Normál)
- 1 - ABORTED (Megszakított)
- 2 - TIMEOUT (Időtúllépéses)

Amikor egy PHP szkript fut, alapállapotban a NORMAL állapot aktív. Ha a távoli kliens bontja a kapcsolatot, az ABORTED statátusz jelzése lesz aktív. Ez tipikusan akkor áll elő, ha a látogató a STOP gomb-ot használja a böngészőjében. Ha a PHP által felügyelt időkorált kerül túllépésre (lásd a [set_time_limit\(\)](#) függvényt), a TIMEOUT állapot válik aktívvá.

Eldöntheted, hogy ha a kliens bontja a kapcsolatot, a szkript is leálljon-e vagy sem. Néha hasznos lehet, ha a szkriptjeid mindenig végigfutnak, annak ellenére, hogy a kliens már nem fogadja a kimenetet. Alapbeállításban azonban a szkript is befejezi a futását, ha a kliens bontja a kapcsolatot. Ez a viselkedés az `ignore_user_abort` php.ini beállítással, valamint az ennek megfelelő "php_value ignore_user_abort" Apache .conf direktívával állítható, vagy az [ignore_user_abort\(\)](#) függvényivel. Ha nem konfigurálod úgy a PHP-t, hogy hagyja figyelmen kívül a kliens kapcsolatbontását, a szkriptjeid le fognak állni ilyen esetekben. Egyetlen kivétel ez alól, ha egy 'shutdown' függvényt definiálsz a [register_shutdown_function\(\)](#)-al. Egy ilyen beállítással, ha a látogató lenyomja a STOP gombot, a szkripted következő kimenet-küldési kísérletére a PHP a 'shutdown' függvényt fogja meghívni. A 'shutdown' függvény abban az esetben is meghívásra kerül, ha a szkript normálisan befejezi a futását, tehát ha valami speciális szeretnél tenni, amikor a kliens bontja a kapcsolatot, a [connection_aborted\(\)](#) függvényt használhatod. Ez igazat fog visszaadni, ha a kapcsolatot a kliens bontotta.

A szkripted a belső időmérés következtében is megállhat. Alapbeállításban egy szkript maximum 30 másodpercig futhat. Ez megváltoztatható a `max_execution_time` php.ini direktívával, illetve a megfelelő `php_value max_execution_time` Apache .conf beállítással, valamint a [set_time_limit\(\)](#) függvényivel. Amikor ez az idő letelik, a szkript megáll, és ha a fenti esetben említett 'shutdown' függvény definiált, az kerül meghívásra. Az időtúllépés esetét a [connection_status\(\)](#) függvényel állapíthatod meg. Ez 2-es értéket fog visszaadni, ha időtúllépés miatt hívódott meg a 'shutdown'.

Fontos megjegyezni, hogy az ABORTED és TIMEOUT állapotok egyszerre is aktívak lehetnek, ha a PHP-ben a kliens kapcsolatbontásának figyelmen kívül hagyását kérte. A PHP tudni fogja, hogy a kliens már bontotta a kapcsolatot, de a szkript futni fog tovább. Ha ráadásul eléri az időkorlátot, a

szkript megáll, és a 'shutdown' függvény hívódik meg (ha beállítottál ilyet). Ezen a ponton azt fogod tapasztalni, hogy a [connection_status\(\)](#) függvény 3-as értékkel fog visszatérni.

41. Fejezet. Állandó adatbázis kapcsolatok

Az állandó kapcsolatok olyan összeköttetések, melyek nem szűnnek meg, ha a szkripted futása befejeződik. Ha egy állandó kapcsolatot kérsz, a PHP ellenőrzi, hogy van-e már megegyező kapcsolat (ami még az előző kérésekből maradhatott meg), és ha létezik, akkor azt használja. Ha nem talál ilyet, létrehoz egy kapcsolatot. A megegyező kapcsolat azt jelenti, hogy ugyanaz a host és ugyanaz a felhasználói név és jelszó került felhasználásra.

Ha esetleg nem ismered alaposabban a webszerverek működését, hibás kép alakulhat ki benned az állandó kapcsolatokról. Ezek a kapcsolatok *nem* alkalmasak arra, hogy felhasználói 'session'-öket nyiss ugyanazon az összeköttetésen. *Nem* adnak lehetőséget hatékony tranzakciók felépítésére. Egészen pontosan, hogy alaposabban tisztázzuk a kérdést, az állandó adatbázis kapcsolatok nem adnak *semmilyen* plusz lehetőséget, ami nélkülük nem létezne.

Miért?

A válaszhoz meg kell érteni, hogyan működnek együtt a webszerverek a PHP-vel. Ennek három különböző módja lehetséges.

Az első lehetőség, hogy a PHP-t CGI "wrapper"-ként használod. Ha ezt a módszert használod, minden oldal lekérésekor és feldolgozásakor egy új példány fut le a PHP feldolgozóból. Mivel a szkript futtatása után egy ilyen példány leáll, minden erőforrás, amit lefoglalt (beleértve az adatbázis kapcsolatotokat) megszűnik. Ebben az esetben semmit sem érsz azzal, hogy állandó kapcsolatot próbálsz nyitni, ez az állandóság nem valósul meg.

A népszerűbb második forma, amikor a PHP-t modulként futtatod egy több process-es webszerverben. Egy több process-es webszerver tipikusan rendelkezik egy szülő process-el, ami koordinálja a többi kapcsolódó process (a gyermekek) munkáját, amik valójában a weboldalak kiszolgálását végzik. Ha egy kérés érkezik egy klienstől, egy éppen szabad gyermekprocess kapja meg a kiszolgálásra az utasítást. Ez azt jelenti, hogy ha ugyanaz a kliens egy újabb kapcsolatot kezdeményez, esetleg egy másik gyermekprocesshez jut, mint az első alkalommal. Amennyiben állandó kapcsolator nyitsz, egy későbbi oldal fel tudja használni ugyanezt a kapcsolatot.

A harmadik módszer, hogy a PHP-t plug-in-ként használod egy 'multithreaded' web szerverben. Ez azt jelenti, hogy az ISAPI, WSAPI, és NSAPI (Windows alatt) formák használhatóak a PHP-vel. Ez a PHP 4.0.0 óta lehetséges, és így a PHP alkalmas plug-in szintű együttműködésre a Netscape FastTrack (iPlanet), a Microsoft Internet Information Server (IIS), és az O'Reilly WebSite Pro szerverekkel, valamint más, a fenti standardokat támogató szerverekkel. Ebben az esetben az állandó adatbázis kapcsolatok működése megegyezik a fent leírt több process-es modellel.

Ha az állandó adatbázis kapcsolatok nem nyújtanak plusz szolgáltatásokat, mégis mire jók?

A válasz igen egyszerű: hatékonyság! Az állandó adatbázis kapcsolatok akkor lehetnek hasznosak, ha nagy a feleslegesen adatbázishoz kapcsolódással eltöltött idő. Az, hogy ez valójában milyen esetben van így, rengeteg faktoron múlik. Például azon, hogy milyen típusú adatbázisról van szó, azonos, vagy különböző gépen van-e, mint a szerver, mennyire terhelt az SQL szerver, stb. Lényegében, ha sok időt vesz igénybe a kapcsolódás, az állandó kapcsolatok jelentős segítséget nyújthatnak neked. Egy gyermekprocess így csak egy alkalommal kell, hogy kapcsolódjon, ahelyett, hogy egy ezt kérő oldal minden feldolgozásakor megtenné. Ez azt is jelenti, hogy minden gyermekprocessnek, meglesz a maga állandó kapcsolata a szerver felé. Például, ha 20 különböző gyermekprocess dolgozott fel egy állandó adatbáziskapcsolatot kérő oldalt, 20 különböző állandó kapcsolatod lesz az SQL szerverhez, egy-egy minden gyermektől.

Fontos megjegyezni azonban, hogy ennek lehetnek hátrányos következményei is, ha az adatbázisszerver korlátozott kapcsolatainak számát az állandó kapcsolatok lefoglalják. Ha az adatbázisszervered egyidejűleg maximálisan 16 kapcsolatot képes kezelni, és egy forgalmas időszakban egyszerre 17 process próbál meg kapcsolódni az adatbázishoz, az egyik képtelen lesz erre... Ha olyan hiba van a programodban (például végetelen ciklus), ami nem hagyja a kapcsolat felbontását, egy csak 16 kapcsolattal bíró adatbázis alaposan le lesz foglalva. Nézz utána az adatbázisszervered dokumentációjában, hogy hogyan tudod lekezelni az elhagyott vagy inaktív kapcsolatokat.

Figyelem

Van még néhány faktor, amit érdemes figyelembe venned, ha állandó adatbázis kapcsolatokat használsz. Egy ilyen probléma, hogy ha tábla lezárást (lock) használ sz egy állandó kapcsolaton, és a szkript valamelyen okból nem tudja feloldani a zárat, ezt a kapcsolatot használó további szkriptek nem fognak helyesen működni, és a webszerver vagy adatbázis szerver újraindítására lehet szükség a feloldáshoz. Hasonlóan ha tranzakciókat használ sz, a tranzakció blokk tovább folytatódik a következő megegyező kapcsolatot használó szkriptben, ha a tranzakciót indító szkript nem tudja lezárnai azt. Ezekben az esetekben a [register_shutdown_function\(\)](#) függvényt használhatod, hogy egy egyszerű "takarító" függvényt futtass le a programod végeztével, ami visszavonja a tranzakciókat, és feloldja a tábla zárákat. Jobban teszed azonban, ha úgy kerülök meg a problémát, hogy nem használ sz állandó kapcsolatokat olyan szkriptekben, amik tábla zárákat, vagy tranzakciókat alkalmaznak.

Összefoglalva: az állandó adatbáziskapcsolatokat úgy terveztek, hogy megfeleltethetőek legyenek a hagyományos kapcsolatokkal. Ez azt jelenti, hogy *minden esetben* lehetőséged van az állandó kapcsolatokat hagyományos kapcsolatokra cserélni, és ez nem fogja megváltoztatni a szkriptjeid működését. Ez a lépés megváltoztathatja a szkripted hatékonyságát, de a viselkedését nem!

Lásd még [fbsql_pconnect\(\)](#), [ibase_pconnect\(\)](#), [ifx_pconnect\(\)](#), [ingres_pconnect\(\)](#), [msql_pconnect\(\)](#), [mssql_pconnect\(\)](#), [mysql_pconnect\(\)](#), [ocipllogon\(\)](#), [odbc_pconnect\(\)](#), [ora_pllogen\(\)](#), [pfsockopen\(\)](#), [pg_pconnect\(\)](#), és [sybase_pconnect\(\)](#).

42. Fejezet. Safe Mode

The PHP safe mode is an attempt to solve the shared-server security problem. It is architecturally incorrect to try to solve this problem at the PHP level, but since the alternatives at the web server and OS levels aren't very realistic, many people, especially ISP's, use safe mode for now.

Figyelem

Safe Mode was removed in PHP 6.0.0.

Security and Safe Mode

Táblázat 42-1. Security and Safe Mode Configuration Directives

Name	Default	Changeable	Changelog
safe_mode	"0"	PHP_INI_SYSTEM	
safe_mode_gid	"0"	PHP_INI_SYSTEM	Available since PHP 4.1.0.
safe_mode_include_dir	NULL	PHP_INI_SYSTEM	Available since PHP

Name	Default	Changeable	Changelog
		M	4.1.0.
safe_mode_exec_dir	""	PHP_INI_SYSTEM	
safe_mode_allowed_env_vars	"PHP_"	PHP_INI_SYSTEM	
safe_mode_protected_env_vars	"LD_LIBRARY_PATH"	PHP_INI_SYSTEM	
open_basedir	NULL	PHP_INI_SYSTEM	
disable_functions	""	php.ini only	Available since PHP 4.0.1.
disable_classes	""	php.ini only	Available since PHP 4.3.2.

For further details and definition of the PHP_INI_* constants see [ini_set\(\)](#).

A témába vágó konfigurációs direktívák rövid leírása

safe_mode [boolean](#)

Whether to enable PHP's safe mode.

safe_mode_gid [boolean](#)

By default, Safe Mode does a UID compare check when opening files. If you want to relax this to a GID compare, then turn on *safe_mode_gid*. Whether to use *UID* (**FALSE**) or *GID* (**TRUE**) checking upon file access.

safe_mode_include_dir [string](#)

UID/GID checks are bypassed when including files from this directory and its subdirectories (directory must also be in [include_path](#) or full path must including).

As of PHP 4.2.0, this directive can take a colon (semi-colon on Windows) separated path in a fashion similar to the [include_path](#) directive, rather than just a single directory.

The restriction specified is actually a prefix, not a directory name. This means that "safe_mode_include_dir = /dir/incl" also allows access to "/dir/include" and "/dir/incls" if they exist. When you want to restrict access to only the specified directory, end with a slash. For example: "safe_mode_include_dir = /dir/incl/"

If the value of this directive is empty, no files with different *UID/GID* can be included in PHP 4.2.3 and as of PHP 4.3.3. In earlier versions, all files could be included.

safe_mode_exec_dir [string](#)

If PHP is used in safe mode, [system\(\)](#) and the other [functions executing system programs](#) refuse to start programs that are not in this directory. You have to use / as directory separator on all environments including Windows.

safe_mode_allowed_env_vars [string](#)

Setting certain environment variables may be a potential security breach. This directive contains a comma-delimited list of prefixes. In Safe Mode, the user may only alter environment variables whose names begin with the prefixes supplied here. By default, users will only be able to set environment variables that begin with PHP_ (e.g. PHP_FOO=BAR).

Megjegyzés: If this directive is empty, PHP will let the user modify ANY environment variable!

safe_mode_protected_env_vars [string](#)

This directive contains a comma-delimited list of environment variables that the end user won't be able to change using [putenv\(\)](#). These variables will be protected even if *safe_mode_allowed_env_vars* is set to allow to change them.

open_basedir [string](#)

Limit the files that can be opened by PHP to the specified directory-tree, including the file itself. This directive is *NOT* affected by whether Safe Mode is turned On or Off.

When a script tries to open a file with, for example, [fopen\(\)](#) or [gzopen\(\)](#), the location of the file is checked. When the file is outside the specified directory-tree, PHP will refuse to open it. All symbolic links are resolved, so it's not possible to avoid this restriction with a symlink.

The special value . indicates that the working directory of the script will be used as the base-directory. This is, however, a little dangerous as the working directory of the script can easily be changed with [chdir\(\)](#).

In `httpd.conf`, *open_basedir* can be turned off (e.g. for some virtual hosts) [the same way](#) as any other configuration directive with "php_admin_value *open_basedir* none".

Under Windows, separate the directories with a semicolon. On all other systems, separate the directories with a colon. As an Apache module, *open_basedir* paths from parent directories are now automatically inherited.

The restriction specified with *open_basedir* is actually a prefix, not a directory name. This means that "*open_basedir* = /dir/incl" also allows access to "/dir/include" and "/dir/incls" if they exist. When you want to restrict access to only the specified directory, end with a slash. For example: "*open_basedir* = /dir/incl/"

Megjegyzés: Support for multiple directories was added in 3.0.7.

The default is to allow all files to be opened.

disable_functions [string](#)

This directive allows you to disable certain functions for [security](#) reasons. It takes on a comma-delimited list of function names. *disable_functions* is not affected by [Safe Mode](#).

This directive must be set in `php.ini`. For example, you cannot set this in `httpd.conf`.

disable_classes [string](#)

This directive allows you to disable certain classes for [security](#) reasons. It takes on a comma-delimited list of class names. `disable_classes` is not affected by [Safe Mode](#).

This directive must be set in `php.ini`. For example, you cannot set this in `httpd.conf`.

Availability note: This directive became available in PHP 4.3.2

See also: [register_globals](#), [display_errors](#), and [log_errors](#).

When [safe_mode](#) is on, PHP checks to see if the owner of the current script matches the owner of the file to be operated on by a file function or its directory. For example:

```
-rw-rw-r--    1 rasmus    rasmus      33 Jul  1 19:20 script.php
-rw-r--r--    1 root       root       1116 May 26 18:01 /etc/passwd
```

Running this `script.php`

```
<?php
  readfile('/etc/passwd');
?>
```

results in this error when safe mode is enabled:

```
Warning: SAFE MODE Restriction in effect. The script whose uid is 500 is not
allowed to access /etc/passwd owned by uid 0 in /docroot/script.php on line 2
```

However, there may be environments where a strict *UID* check is not appropriate and a relaxed *GID* check is sufficient. This is supported by means of the [safe_mode_gid](#) switch. Setting it to *On* performs the relaxed *GID* checking, setting it to *Off* (the default) performs *UID* checking.

If instead of [safe_mode](#), you set an [open_basedir](#) directory then all file operations will be limited to files under the specified directory. For example (Apache `httpd.conf` example):

```
<Directory /docroot>
  php_admin_value open_basedir /docroot
</Directory>
```

If you run the same `script.php` with this [open_basedir](#) setting then this is the result:

```
Warning: open_basedir restriction in effect. File is in wrong directory in
/docroot/script.php on line 2
```

You can also disable individual functions. Note that the [disable_functions](#) directive can not be used outside of the `php.ini` file which means that you cannot disable functions on a per-virtualhost or per-directory basis in your `httpd.conf` file. If we add this to our `php.ini` file:

```
disable_functions readfile,system
```

Then we get this output:

```
Warning: readfile() has been disabled for security reasons in
/docroot/script.php on line 2
```

Figyelem

These PHP restrictions are not valid in executed binaries, of course.

Functions restricted/disabled by safe mode

This is a still probably incomplete and possibly incorrect listing of the functions limited by [safe mode](#).

Táblázat 42-2. Safe mode limited functions

Function	Limitations
dbmopen()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
dbase_open()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
filepro()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
filepro_rowcount()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
filepro_retrieve()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
ifx_*	sql_safe_mode restrictions, (!= safe mode)
ingres_*	sql_safe_mode restrictions, (!= safe mode)
mysql_*	sql_safe_mode restrictions, (!= safe mode)
pg_lo_import()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
posix_mkfifo()	Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
putenv()	Obey the safe_mode_protected_env_vars and safe_mode_allowed_env_vars ini-directives. See also the documentation on putenv()
move_uploaded_file()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
chdir()	Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
dl()	Ez a függvény nem használható, ha a safe mode be van kapcsolva.
backtick operator	Ez a függvény nem használható, ha a safe mode be van kapcsolva.
shell_exec() (functional equivalent of backticks)	Ez a függvény nem használható, ha a safe mode be van kapcsolva.
exec()	You can only execute executables within the safe_mode_exec_dir . For practical reasons it's currently not allowed to have .. components in the path to the executable. escapeshellcmd() is executed on the argument of this function.
system()	You can only execute executables within the safe_mode_exec_dir . For

Function	Limitations
	practical reasons it's currently not allowed to have .. components in the path to the executable. escapeshellcmd() is executed on the argument of this function.
passthru()	You can only execute executables within the safe_mode_exec_dir . For practical reasons it's currently not allowed to have .. components in the path to the executable. escapeshellcmd() is executed on the argument of this function.
popen()	You can only execute executables within the safe_mode_exec_dir . For practical reasons it's currently not allowed to have .. components in the path to the executable. escapeshellcmd() is executed on the argument of this function.
fopen()	Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
mkdir()	Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
rmdir()	Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
rename()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
unlink()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
copy()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program. (on source and target)
chgrp()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
chown()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program.
chmod()	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. In addition, you cannot set the SUID, SGID and sticky bits

Function	Limitations
<u>touch()</u>	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program.
<u>symlink()</u>	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program. (note: only the target is checked)
<u>link()</u>	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program. (note: only the target is checked)
<u>apache_request_headers()</u>	In safe mode, headers beginning with 'authorization' (case-insensitive) will not be returned.
<u>header()</u>	In safe mode, the uid of the script is added to the <i>realm</i> part of the <i>WWW-Authenticate</i> header if you set this header (used for HTTP Authentication).
<u>PHP_AUTH variables</u>	In safe mode, the variables <i>PHP_AUTH_USER</i> , <i>PHP_AUTH_PW</i> , and <i>AUTH_TYPE</i> are not available in <i>\$_SERVER</i> . Regardless, you can still use <i>REMOTE_USER</i> for the USER. (note: only affected since PHP 4.3.0)
<u>highlight_file(), show_source()</u>	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program. (note: only affected since PHP 4.2.1)
<u>parse_ini_file()</u>	Ellenőrzi, hogy az állományok/könyvtárak, amelyekkel dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkeznek-e, mint az éppen futó program. Ellenőrzi, hogy a könyvtár, amelyben dolgozni szeretnél, ugyanazzal a felhasználói azonosítóval (UID) rendelkezik-e, mint az éppen futó program. (note: only affected since PHP 4.2.1)
<u>set_time_limit()</u>	Has no effect when PHP is running in <u>safe mode</u> .
<u>max_execution_time</u>	Has no effect when PHP is running in <u>safe mode</u> .
<u>mail()</u>	In safe mode, the fifth parameter is disabled. (note: only affected since PHP 4.2.3)
Any function that uses <i>php4/main/fopen_wrappers.c</i>	??

43. Fejezet. Parancssori programozás a PHP-ben

A 4.3-as verziótól felfelé a PHP támogat egy új *SAPI* (Server Application Programming Interface) típust, a *CLI*-t ami a *Command Line Interface* rövidítése. A nevéből következik, hogy ez a *SAPI* típus leginkább shell (vagy akár desktop!) alkalmazások fejlesztéséhez használatos a PHP-ben. Van néhány különbség a *CLI SAPI* és a többi *SAPI* között, ezek magyarázatára a későbbiekben rátérünk. Nem árt megjegyezni, hogy a *CLI* és a *CGI* két különböző *SAPI* bár elég hasonló a viselkedésmódjuk.

A *CLI SAPI* először a *PHP 4.2.0* verziójában jelent meg, de akkor még csak kísérleti jelleggel. Használatához a *--enable-cli* kapcsolóval kellett indítani a *./configure* scriptet installáláskor. A *PHP 4.3.0* megjelenésétől megszűnt a *CLI SAPI* kísérleti státusza és a *--enable-cli* alapértelmezésben be van állítva. Kikapcsolásához a *--disable-cli* konfigurációs paramétert használhatod.

A *PHP 4.3.0*-tól a *CLI/CGI* binárisok neve, helye és létezése függ attól, hogyan lett a PHP telepítve. Alapértelmezésben, a **make** parancs végrehajtásakor, a *CGI* és a *CLI* is létre lesz hozva és a *sapi/cgi/php* illetve a *sapi/cli/php* könyvtárakba lesznek helyezve a PHP forráskönyvtáron belül. Észreveheted, hogy minden kettőnek a neve *php*. Ami a **make install** folyamán történik, az a *configure* opcióktól függ. Ha ki lett választva egy *SAPI* modul, mint pl. *aspx*, vagy a *--disable-cgi* meg volt adva, akkor a **make install** idejére a *{PREFIX}/bin/php* helyen a *CLI* lesz, egyébként pedig a *CGI*-t teszi oda. Például ha *--with-apxs* benne van a *configure* sorban, akkor a *CLI*-t másolja a *{PREFIX}/bin/php* helyre a **make install** idejére. Ha felül akarod bírálni a *CGI* bináris telepítését, akkor a **make install** után add ki a **make install-cli** parancsot. Egy másik lehetőség, hogy megadod a *--disable-cgi* *configure* opciót.

Megjegyzés: Mivel mind a *--enable-cli* mind a *--enable-cgi* alapértelmezésben be van állítva, az, hogy a *configure* sorban az *--enable-cli* meg van adva, nem feltétlenül jelenti azt, hogy a *CLI* lesz a *{PREFIX}/bin/php* helyre másolva a **make install** idejére.

A *PHP 4.2.0* és *PHP 4.2.3* közötti windows-os csomagok a *CLI*-t *php-cli.exe*-ként tartalmazták ugyanabban a könyvtárban, mint a *CGI* *php.exe*. A *PHP 4.3.0*-tól kezdődően a windows-os csomag a *CLI*-t *php.exe*-ként tartalmazza egy *cli* nevű könyvtárban, azaz *cli/php.exe*. A *PHP 5*-től kezdődően a *CLI* a főkönyvtárban van és a neve *php.exe*. A *CGI* verzió a *php-cgi.exe*.

A *PHP 5*-től egy új *php-win.exe* nevű fájl is található a csomagban. Ez csak annyiban különbözik a *CLI* verziótól, hogy a kimenetre nem ír ki semmit, így nem jelenít meg konzolt (nem jelenik meg a dos ablak a képernyőn). Ez a viselkedésmód hasonló a *php-gtk*-éhoz. A *configure* opciók között az *--enable-cli-win32* kell szerepeljen.

Milyen SAPI-m van?: Egy shell-be beírva a *php -v* elárulja hogy a *php* CGI vagy CLI. Lásd még a **[php_sapi_name\(\)](#)** függvényt valamint a ***PHP_SAPI*** konstanst.

Megjegyzés: A *PHP 4.3.2*-től egy létrejött egy unix *man* oldal. Ezt a **man php** shell-be beírt parancssal nézheted meg.

Lényegesebb különbségek a *CLI SAPI* és a többi *SAPI* között:

- A *CGI SAPI*-val ellentétben soha nem ír ki fejléceket.

Habár a *CGI SAPI* lehetőséget ad a HTTP fejlécek letiltására egy kapcsolóval, de ez nem egyenértékű a *CLI SAPI* által nyújtott megoldással.

A *CLI* alapértelmezetten "csendes" (quiet) módban indul, bár a *-q* és a *--no-header* kapcsolót megtartották a kompatibilitás érdekében, hogy régebbi CGI szkripteket is problémamentesen

lehessen futtatni.

Nem cseréli fel az aktuális könyvtárat az éppen futó szkript könyvtárára. (A *-C* és a *--no-chdir* kapcsolót azért megtartották a kompatibilitás érdekében)

Egyszerű, szöveges hibaüzenetek (nincs HTML formázás).

- Vannak `php.ini` utasítások, melyeket a *CLI SAPI* egyszerűen figyelmen kívül hagy, mivel nincs közük a shell környezethez:

Táblázat 43-1. Figyelmen kívül hagyott `php.ini` utasítások

Utasítás	<i>CLI SAPI</i> default value	Magyarázat
html_errors	FALSE	A shellben lehetetlen elolvasni a hibaüzeneteket, ha azok zavaros <i>HTML</i> elemekkel tarkítva száguldannak át a képernyőn. Emiatt ez az utasítás alapértelmezetten FALSE .
implicit_flush	TRUE	Általában azt akarjuk, hogy a print() , echo() és a hasonszűrű függvények mindenkorban írjanak a kimenetre és ne puffereljenek semmit. De használható a output buffering utasítás, ha a kimenet késleltetése vagy manipulálása a cél.
max_execution_time	0 (korlátlan)	Minden eshetőségre felkészülve a PHP nem korlátozza a shell szkriptek futásidélét. Ez érthető, hiszen még egy webes szkript általában nagyon gyorsan lefut, addig a shellprogramok nagyon hosszú ideig futhatnak.
register_argc_argv	TRUE	Mivel ez a beállítás TRUE , <i>CLI SAPI</i> -ban minden rendelkezésre állnak az <i>argc</i> (az alkalmazásnak átadott argumentumok száma) és az <i>argv</i> (az aktuális argumentumok tömbje) változók. A PHP 4.3.0-tól a <i>\$argc</i> és a <i>\$argv</i> PHP változók <i>CLI SAPI</i> használata esetén létrejönnek és feltöltődnek a megfelelő értékekkel. Azelőtt ezen változók létrehozása ugyanúgy műköött mint a <i>CGI</i> és <i>MODUL</i> verziókban, ahol a register_globals be kellett legyenek kapcsolva. A \$_SERVER vagy a \$_HTTP_SERVER_VARS tömböt minden használhatod függetlenül a verziótól és a <i>register_globals</i> beállítástól.

Megjegyzés: Ezek az utasítások nem adhatók meg a fentiek előtérül különböző értékekkel a konfigurációs `php.ini` fájlban vagy egyéb saját `php.ini`-ben (ismert, hogy több `php.ini` is használható, akár könyvtáraként más). Ez egy korlátozás, mert alapértelmezett értékeik azután aktiválódnak, miután a konfigurációs fájlok lefutottak. Azonban ezek az értékek változhatnak a szkript futása alatt (ami nincs hatással minden említett utasításra, pl. [register_argc_argv](#)).

- Hogy kényelmesebbé tegyék a parancssori programozást, néhány konstanst előre definiáltak:

Táblázat 43-2. Specifikus CLI konstansok

konstansok	Leírás
STDIN	Egy, már megnyitott stream (folyam) a <i>stdin</i> -re (standard input - bemenet). A

konstansok	Leírás
	<p>stream következőképpen történő megnyitásától a megkímél bennünket:</p> <pre><?php \$stdin = fopen('php://stdin', 'r'); ?></pre> <p>Ha egyetlen sort akarsz olvasni a <i>stdin</i>-ról, haszálд ezt:</p> <pre><?php \$line = trim(fgets(STDIN)); // egy sor olvasása a STDIN-ről fscanf(STDIN, "%d\n", \$number); // szám olvasása a STDIN-ről ?></pre>
STDOUT	<p>Egy, már megnyitott stream az <i>stdout</i>-ra (standard output - kimenet). A stream következőképpen történő megnyitásától a megkímél bennünket:</p> <pre><?php \$stdout = fopen('php://stdout', 'w'); ?></pre>
STDERR	<p>Egy, már megnyitott stream az <i>stderr</i>-re (ez a hibaüzenetek kiíratásáért felelős). A stream következőképpen történő megnyitásától a megkímél bennünket:</p> <pre><?php \$stderr = fopen('php://stderr', 'w'); ?></pre>

A fentieknek megfelelően nincs szükség arra, hogy pl. megnyiss egy streamet az *stderr*-ért, hanem egyszerűen csak használд a konstanst a stream forrás helyett:

```
php -r 'fwrite(STDERR, "stderr\n");'
```

Ezeket a stameket nem kell külön lezárni, a PHP ezt automatikusan megtesz a szcript futásának befejeztekor.

- A *CLISAPI* nem cseréli az aktuális könyvtára a futó szkript könyvtárára!

Szemléletes példa erre a *CGISAPI* sajátosságára:

```
<?php
// Ez a mi kis tesztprogramunk amelynek neve: test.php
echo getcwd(), "\n";
?>
```

A *CGI* verzió használatakor a következőkre számíthatunk:

```
$ pwd //Linux/Unix alatt kiírja az aktuális könyvtárat
/tmp //az aktuális könyvtár a /tmp

$ php -q másik_könyvtár/test.php
/tmp/másik_könyvtár
```

Ebből tisztán látszik, hogy a PHP lecserélte az aktuális könyvtárat a futtatott szkriptére.

Ugyanez a *CLISAPI*-val:

```
$ pwd
/tmp

$ php -f másik_könyvtár/test.php
```

```
/tmp
```

Ez sokkal nagyobb rugalmasságot biztosít a parancssori programok írása során a PHP-ben.

Megjegyzés: A fenti példában a *CGI SAPI* ugyanúgy viselkedik, mint a *CLI SAPI*, ha a *-C* kapcsolóval indítod a szkriptet a parancssorból.

Az alábbi listát a parancssori opciókról a PHP generálta. Ezt bármikor kilistáztathatod ha a PHP-t a *-h* kapcsolóval indítod parancssorból:

```
Usage: php [options] [-f] <file> [--] [args...]
php [options] -r <code> [--] [args...]
php [options] [-B <begin_code>] -R <code> [-E <end_code>] [--] [args...]
php [options] [-B <begin_code>] -F <file> [-E <end_code>] [--] [args...]
php [options] -- [args...]

-a           Run interactively
-c <path>|<file> Look for php.ini file in this directory
-n           No php.ini file will be used
-d foo[=bar] Define INI entry foo with value 'bar'
-e           Generate extended information for debugger/profiler
-f <file>   Parse <file>.
-h           This help
-i           PHP information
-l           Syntax check only (lint)
-m           Show compiled in modules
-r <code>    Run PHP <code> without using script tags <?..?>
-B <begin_code> Run PHP <begin_code> before processing input lines
-R <code>    Run PHP <code> for every input line
-F <file>    Parse and execute <file> for every input line
-E <end_code> Run PHP <end_code> after processing all input lines
-H           Hide any passed arguments from external tools.
-s           Display colour syntax highlighted source.
-v           Version number
-w           Display source with stripped comments and whitespace.
-z <file>    Load Zend extension <file>.

args...      Arguments passed to script. Use -- args when first argument
            starts with - or script is read from stdin
```

A *CLISAPI*-val háromféleképpen indíthat sz el egy PHP programot:

1. Hogyan lehet a PHP-vel fájlokat futtatni.

```
php my_script.php
php -f my_script.php
```

Mindkét módon (vagy használva a *-f* kapcsolót vagy nem) futtatja a *my_script.php* nevű szkriptet. Bármilyen futtathatsz, a PHP szkriteknek nem muszáj *.php* kiterjesztésüknek lenniük, bármilyen nevű és kiterjesztésű fájl futtatható.

2. PHP kód futtatása közvetlenül a parancssorból.

```
php -r 'print_r(get_defined_constants());'
```

Különösen figyelni kell a shell változók helyettesítésére és az idézőjelekre!

Megjegyzés: A példát figyelmesen szemlélve észrevehetjük, hogy nincsenek nyitó és záró tagok. Az *-r* kapcsolóval ezekre nincs szükség. Ha mégis használod őket, az hibához fog vezetni.

3. PHP kód futtatása a standard inputon (*stdin*) keresztül.

Ez a módszer dinamikussá teszi a PHP kód létrehozását és egyből a futtatható binárisba táplálja a kódot, amint a következő (képzeletbeli) példában láthatjuk:

```
$ valami_szkript | valami_filter | php | sort -u >eredmeny.txt
```

A három programfuttatási módszert nem lehet egymással kombinálni.

Mint minden shellprogram, a PHP bináris képes argumentumokat fogadni, viszont az általad írt PHP szkript is. Bárminnyi argumentumot megadhatsz a szkriptednek, ezek számát nem korlátozza a PHP (A shellben van egy bizonyos határ a megadható argumentumok számát illetően, de az általában bőségesen elég). A szkriptnek átadott argumentumokat a *\$argv* globális tömb tartalmazza. A tömb nulladik eleme mindenkor a szkript neve. (Ez a - karakterrel abban az esetben, ha a PHP kód az *-r* kapcsolóval lett indítva a parancssorból.) A másik globális tömb a *\$argc*, ami a *\$argv* tömb elemeinek számát tartalmazza (de ez **nem** egyenlő a szkriptnek átadott argumentumok számával).

Amikor különféle opciókkal akarod futtatni egy szkriptet, az argumentumoknak nem szabad - karakterrel kezdődniük. Ha mégis kiteszed a - jelet, akkor abból könnyen probléma lehet, mert a PHP úgy veszi, mintha a saját opciói közül adnál meg egyet. Hogy ezt elkerüld, használ a -- szeparátort, és az utána következő az argumentumokat a PHP változtatás nélkül továbbítja a szkriptnek.

```
# Ez nem fogja futtatni a megadott kódot, csak szemléltetésre jó.  
$ php -r 'var_dump($argv);' -h  
Usage: php [options] [-f] <file> [args...]  
[...]  
  
# Ez átadja a szkriptnek a '-h' argumentumot és megakadályozza a PHP-t abban,  
# hogy a sajátjának higgye.  
$ php -r 'var_dump($argv);' -- -h  
array(2) {  
    [0]=>  
        string(1) "-"  
    [1]=>  
        string(2) "-h"  
}
```

Azonban van egy másik módja a PHP paracssori futtatásának. Lehet írni olyan programokat, melyek a *#!/usr/bin/php* sorral kezdődnek és ezt követi a "normál" PHP kód, a szabványos PHP kezdő- és záró tagokkal. Ha megfelelően beállítottad a fájl futtatói jogosultságát (pl. **chmod +x test**), úgy futtathatod a programdat, mint egy normál shell vagy perl szkriptet:

```
#!/usr/bin/php  
<?php  
var_dump($argv);  
?>
```

Feltéve, hogy ennek a fájlnak *test* a neve (és az éppen aktuális könyvtárunkban van, Linux alatt), a következőket tehetjük:

```
$ chmod +x test  
$ ./test -h -- foo  
array(4) {  
    [0]=>  
        string(6) "./test"  
    [1]=>  
        string(2) "-h"  
    [2]=>  
        string(2) "--"  
    [3]=>  
        string(3) "foo"  
}
```

Látható, hogy ebben az esetben semmi problémát nem okozott az, hogy a szkriptnek a - karakterrel

adtuk át az argumentumokat.

A hosszú opciók a PHP 4.3.3-tól léteznek.

Táblázat 43-3. Parancssori opciók

Opció	Hosszú opció	Leírás
-a	--interactive	A PHP-t interaktív módban futtatja.
-c	--php-ini	<p>Ha nem a megszokott helyén van, akkor megadhatjuk ezzel a kapcsolóval, hogy egy saját INI fájlot (aminek nem muszáj <code>php.ini</code> nevet adni!), pl.:</p> <pre>#ezzel megmondjuk a PHP-nek, hogy a /saját/könyvtárban találja a \$ php -c /saját/könyvtár/ maszek.php</pre> <p>#ez pedig utasítja PHP-t, hogy a /saját/könyvtárban/ levő saját.ini vegye alapul a maszek.php szkript futtatásakor.</p> <pre>\$ php -c /saját/könyvtár/saját.ini maszek.php</pre> <p>Ha ezt az opciót nem adod meg, az állományt az alapértelmezett helyeken keresi.</p>
-n	--no-php-ini	A <code>php.ini</code> teljes figyelmen kívül hagyása. Ez a kapcsoló a PHP 4.3.0 óta létezik.
-d	--define	<p>Ezzel az opcióval bármilyen konfigurációs utasítást, ami csak a <code>php.ini</code> érvényesíthetünk a szkript futásának idejére. Az általános formája a következő:</p> <pre>-d konfigurációs_utasítás[=érték]</pre> <p>Példák (a sorok az olvashatóság érdekében vannak tördelve):</p> <pre># Az értéket elhagyva a megadott konfigurációs utasításhoz az "1" # rendeli. \$ php -d max_execution_time -r '\$foo = ini_get("max_execution_time"); var_dump(\$foo); string(1) "1" # Ha csak egy szóközt nyomunk az érték helyett, akkor a konfigurációs # utasításnak az "" értéket adja. \$ php -d max_execution_time= -r '\$foo = ini_get("max_execution_time"); var_dump(\$foo); string(0) "" # A konfigurációs utasítás értéke az lesz, amit az egyenlőségjel # követ. Az érték előtt először a konfigurációs utasításnak az "" értéket # kell adnia. \$ php -d max_execution_time=20 -r '\$foo = ini_get("max_execution_time"); var_dump(\$foo); string(2) "20" \$ php -d max_execution_time=fogalmamsincsen -r '\$foo = ini_get("max_execution_time"); var_dump(\$foo); string(15) "fogalmamsincsen"</pre>
-e	--profile-info	Aktiválja a részletes információs módot, amelyet a debugger/profiler használ.
-f	--file	Értelmezi és futtatja az <code>-f</code> kapcsoló után megadott fájlt. Ez a kapcsoló opcionális, de legalább egy fájl futtatásához kötelező.
-h és -?	--help és --usage	Ezzel az opcióval lehet információt szerezni az aktuális parancssori opciókról és a hibakódokról.
-i	--info	Ez a parancssori opció meghívja a phpinfo() függvényt és kiírja az eredményét. Ha kiadni egy <code>php -i</code> parancsot és figyelmesen elolvastani a hibaüzeneteket a táblázatban, akkor a kimenet HTML formázott, így a parancssorban szintén megadhatunk egy HTML fájlba (<code>php -i >phpinfo.html</code>) és nézhetjük meg egy böngészővel.

Opció	Hosszú opció	Leírás
-l	--syntax-check	<p>Segítségével kényelmesen elvégezhető a szintaktikai ellenőrzés egy megadott PHP kimenetre kiírja, hogy <i>No syntax errors detected in <filename></i> És a shell vissza <i>Errors parsing <filename></i>, majd kiírja a standard kimenetre a megérdelemelt hiba 255 lesz.</p> <p>Ez az opció nem jelzi a végzetes hibákat (mint pl. egy definiálatlan függvény). Ha végzetes hibát is akarsz találni. ;)</p> <p>Megjegyzés: Ez az opció nem használható együttesen az <i>-r</i> kapcsolóval.</p>
-m	--modules	<p>Eme opció használatával a PHP kilistázza a beépített (és betöltött) PHP és Zend modulait.</p> <pre>\$ php -m [PHP Modules] xml tokenizer standard session posix pcre overload mysql mbstring ctype [Zend Modules]</pre>
-r	--run	<p>Ez az opció teszi lehetővé, hogy PHP parancsokat adjunk ki közvetlenül a parancssorba (<i><?php</i> és <i>?></i>) nem kellenek és szintaktikai hibához fog vezetni, ha mégis alkalmazunk.</p> <p>Megjegyzés: Óvatosan kell bánni ezzel a parancssori PHP futtatási helyettesítés miatt összeakadjon a shelllel.</p> <p>Példa egy ilyen hibára</p> <pre>\$ php -r "\$foo = get_defined_constants();" Command line code(1) : Parse error - parse error, unexpected T_VARIABLE</pre> <p>Itt az a probléma, hogy az sh/bash shell elvégezte a változó helyettesítést mert dupálta a \$foo-t, azonban a \$foo nem definiált, a shell behelyettesíti egy üres értékre, vagyis az a hiba, amit adtuk volna ki a PHP-nek:</p> <pre>\$ php -r " = get_defined_constants();"</pre> <p>Az a helyes megoldás, ha egyszerű idézőjeleket (') használunk. Az ilyen idézőjel foglalkozik az sh/bash.</p> <pre>\$ php -r '\$foo = get_defined_constants(); var_dump(\$foo);' array(370) { ["E_ERROR"]=> int(1) ["E_WARNING"]=> int(2) ["E_PARSE"]=> int(4) ["E_NOTICE"]=> int(8) ["E_CORE_ERROR"]=> [...]</pre>

Opció	Hosszú opció	Leírás
		Ha nem sh/bash shellt használsz, egyéb problémákkal is találkozhatsz. Jelentsd Problémákat okozhat, ha shell változókat akarsz a kódba integrálni vagy blackslai.
		Megjegyzés: A <i>-r</i> kapcsoló csak a <i>CLI</i> SAPI-ban áll rendelkezésre, a <i>CGI</i> SAPI-ban nem.
-B	--process-begin	Az stdin feldolgozása előtt végrehajtandó PHP kód. PHP 5-től létezik.
-R	--process-code	Minden bemeneti sorra végrehajtandó PHP kód. PHP 5-től létezik. Van két speciális változó, amely ebben a módban használható: <i>\$argn</i> és <i>\$argi</i> . Az <i>\$argn</i> minden bemeneti sorra érvényes, az <i>\$argi</i> pedig a sornak a sorszámát tartalmazza.
-F	--process-file	Minden bemeneti sorra végrehajtandó PHP fájl. PHP 5-től létezik.
-E	--process-end	Minden bemeneti sor feldolgozása után végrehajtandó PHP kód. PHP 5-től létezik. Példa a <i>-B</i> , <i>-R</i> és <i>-E</i> opciók használatára: egy projekt sorainak megszámolása. \$ find my_proj php -B '\$l=0;' -R '\$l += count(@file(\$argn));' -E 'print \$l;' Total Lines: 37328
-s	--syntax-highlight és --syntax-highlighting	Színesben kiemelt forrását írja ki a szkriptnek. Ez az opció egy saját algoritmust használ a fájl elemzéséhez, amivel <i>HTML</i> kimenetre. Ez a kód tartalmazza a színes kiemeléshez szükséges és az egyéb formákat (<i>...J </code></i>) <i>HTML</i> formában, de nem tartalmazza a <i>HTML</i> fejléceket.
		Megjegyzés: Ez az opció nem használható együtt az <i>-r</i> kapcsolóval.
-v	--version	Kiírja a PHP, PHP SAPI, és a Zend verziószámát a standard kimenetre, pl: \$ php -v PHP 4.3.0 (cli), Copyright (c) 1997-2002 The PHP Group Zend Engine v1.3.0, Copyright (c) 1998-2002 Zend Technologies
-w	--strip	Kommentek és felesleges sorközök nélkül listázza ki a kódot. Megjegyzés: Ez az opció nem használható együtt az <i>-r</i> kapcsolóval.
-z	--zend-extension	Betölti a Zend bővítményt. Ha csak a futtatandó szkript nevét adtuk meg utána, a bővítményt a rendszeren alapértelmezett függvénykönyvtár (library) útvonalában (<i>/etc/ld.so.conf</i> fájlból van definiálva a Linux rendszereken). Ha megadunk ezt veszi alapul, nem pedig a rendszer általit. Relatív útvonalat is megadhatunk a PHP-képest hol keressük a bővítményt.

A PHP futtatható állomány segítségével PHP szkripteket webszervertől függetlenül lehet futtatni. Ha egy Unix rendszeren vagy, a PHP szkripteket elejére egy speciális sort kell beillesztened, majd az állományt futtathatóvá kell tenned, így a rendszer tudni fogja melyik programmal kell futtatni a szkriptet. Windows-on a *.php* állományokhoz hozzárendelheted a *php.exe* programot, vagy készíthetsz egy batch fájlt, amellyel a szkriptet PHP-n keresztül tudod futtatni. A Unix rendszerek miatt beszűrt első sor nem fog problémát okozni Windows-on, így írhatsz platformfüggetlen programot is. Alább található egy példa parancssor PHP program írására.

Példa 43-1. Parancssorból futtatható szkript (script.php)

```
#!/usr/bin/php
<?php
```

```
if ($argc != 2 || in_array($argv[1], array('--help', '-help', '-h', '-?'))) {  
?>
```

Ez egy parancssori PHP szkript egy opcióval.

Használat:

```
<?php echo $argv[0]; ?> <opción>
```

Az <opción> lehet bármilyen szó, amit szeretnél kiíratni. --help, -help, -h, vagy -? opciókra ezt a súgót kapod.

```
<?php  
} else {  
    echo $argv[1];  
}  
?>
```

A fenti szkriptben használtuk a speciális első sort, hogy ezt a fájlt a PHP futtassa. Most a CLI verziót használjuk, így nem lesznek HTTP fejlécek kiírva. Van két változó, amit használhatsz parancssori PHP alkalmazások írásakor: \$argc and \$argv. Az első megadja az argumentumok számát + 1 (a futó szkript nevét). A második egy tömb, amely az argumentumokat tartalmazza, a szkript nevével kezdődően 0-tól számozva (\$argv[0]).

A fenti programban megvizsgáljuk, hogy több vagy kevesebb mint egy argumentum van-e megadva. Ekkor vagy ha az argumentum --help, -help, -h vagy -?, akkor kiíratjuk a súgó üzenetet, a szkriptnevet dinamikusan írjuk ki. Ha más argumentumot kapunk, akkor azt kiírjuk.

Ha a fenti szkriptet Unix-on szeretnéd futtatni, futtathatóvá kell tenned, majd egyszerűen meghívhatod mint például **script.php echothis** vagy **script.php -h**. Windows-on készíthetsz egy batch fájlt erre a célra:

Példa 43-2. Batch fájl parancssori PHP szkript futtatására (script.bat)

```
@c:\php\cli\php.exe script.php %1 %2 %3 %4
```

Feltételezve, hogy a fenti programot **script.php**-nek nevezted el, és a **php.exe** CLI programod a **c:\php\cli\php.exe** helyen van, ez a batch fájl lefutattja neked a további opciókkal: **script.bat echothis** vagy **script.bat -h**.

Lásd még a [Readline](#) kiterjesztés dokumentációját, ahol további függvényeket találsz a parancssori PHP alkalmazásaid fejlesztéséhez.