16. ÓRA

Az adatok kezelése

Ezen az órán az adatellenőrzésben és az adatműveletekben fogunk egy kicsit elmélyedni. Újra áttekintjük az adattípusokat. A PHP ugyan automatikusan kezeli ezeket, de elengedhetetlen, hogy értsük az adatkezelést, ha nagy, megbízható hálózati alkalmazásokat kell írnunk. Visszatérünk a tömbökhöz is és végül megismerkedünk a PHP fejlettebb adatkezelési, adatosztályozási eljárásaival.

Az óra folyamán a következőkről tanulunk:

- Hogyan alakítsuk át az egyes adattípusokat más adattípusokká?
- Hogyan alakítja át automatikusan a PHP az adattípusokat a kifejezésekben?
- Milyen további módjai vannak az adattípusok ellenőrzésének?
- Miért fontos megértenünk az adattípusokat?
- Hogyan ellenőrizzük, hogy egy változónak van-e értéke?
- Hogyan lehet másként bejárni egy tömböt?
- Hogyan deríthetjük ki, hogy egy tömb tartalmaz-e egy bizonyos elemet?
- Hogyan alakítsuk át egy tömb minden elemét?
- Hogyan lehet saját szempontok alapján tömbrendezést végezni?

Újra az adattípusokról

A negyedik órában már tanultunk néhány dolgot a PHP adattípusairól. Van azonban még valami, ami eddig kimaradt. Ebben a részben a változók adattípusának ellenőrzésére használt függvények közül ismerkedünk meg néhánnyal, majd sorra vesszük azokat a feltételeket, amelyek mellett a PHP automatikusan elvégzi helyettünk az adattípus-átalakításokat.

Egy kis ismétlés

Azt már tudjuk, hogy a PHP változóinak értéke egész szám (integer), lebegőpontos szám (double), karakterlánc (string), logikai érték (boolean), objektum vagy tömb lehet. A `gettype()` függvénnyel bármelyik változótípust lekérdezhetjük. A függvény bemenete egy tetszőleges típusú változó, kimenete pedig a változó adattípusát leíró szöveg:

```
$adat = 454;
print gettype( $adat );
// azt írja ki, hogy "integer", azaz egész szám
```

A változók adattípusát közvetlen típusátalakítással vagy a `settype()` függvénnyel módosíthatjuk. Az adattípusok átalakításához a zárójelbe tett adattípust a megváltoztatandó változó vagy érték elé írjuk. A folyamat közben a változó tartalmát a legcsekélyebb mértékben sem módosítjuk, ehelyett egy átalakított másolatot kapunk vissza. A következő kódrészlet egy tizedestörtet alakít át egész számmá.

```
$adat = 4.333;
print ( integer ) $adat;
// "4"-et ír ki
```

Az `$adat` változó lebegőpontos szám típusú maradt, mi csupán a típusátalakítás eredményét írattuk ki.

A változók adattípusának módosításához a `settype()` függvényt használhatjuk, amelynek bemenete az átalakítandó változó és a céladattípus neve.

```
$adat = 4.333;
settype( $adat, "integer" );
print $adat;
// "4"-et ír ki
```

Az `$adat` változó most már egy egész számot tartalmaz.

Összetett adattípusok átalakítása

Néhány egyszerű (skaláris és szöveges) adattípus közötti átváltást már láttuk részleteiben is. Mi történik viszont akkor, ha egyszerű (egész és nem egész számok) és összetett adattípusok (objektumok és tömbök) között kell típust váltani?

Amikor egy egyszerű adattípust tömbbé alakítunk, olyan tömb jön létre, melynek első eleme az eredeti érték lesz:

```
$szoveg = "ez az én szövegem"
$tomb_szoveg = (array) $szoveg;
print $tomb_szoveg[0];
// kiírja az "ez az én szövegem" karakterláncot
```

Amikor a skaláris vagy szöveges változókat objektumokká alakítjuk, egy egyetlen `scalar` nevű tulajdonságot tartalmazó objektum jön létre, ez tartalmazza az eredeti értéket:

```
$szoveg = "ez az én szövegem"
$obj_szoveg = (object) $szoveg;
print $obj_szoveg->scalar;
// kiírja az "ez az én szövegem" karakterláncot
```

A dolog a tömbök és objektumok közötti váltáskor válik igazán izgalmassá. Amikor egy tömböt objektummá alakítunk, egy olyan új objektum jön létre, amelyben a tömb egyes kulcsainak egy-egy tulajdonság felel meg.

```
$cimek = array ( "utca" => "Főutca", "varos" => "Nagyfalva" );
$obj_cimek = ( object ) $cimek;
print $obj_cimek->utca;
// azt írja ki, hogy "Főutca"
```

Fordított esetben, egy objektum tömbbé alakításakor, olyan tömb jön létre, amelyben minden objektum tulajdonságnak megfelel egy tömbelem. A metódusokat az átalakítás figyelmen kívül hagyja.

```
class Pont
     {
     var $x;
     var $y;
     function Pont( $x, $y )
          {
          $this->x = $x;
          $this->y = $y;
```

```
        }
    }
$pont = new Pont( 5, 7 );
$tomb_pont = (array) $pont;
print $tomb_pont["x"];
// azt írja ki, hogy "5"
```

Az adattípusok automatikus átalakítása

Ha olyan kifejezést hozunk létre, amelyben két különböző adattípus szerepel operandusként, a PHP automatikusan átalakítja az egyiket a számíthatóság kedvéért. Valószínűleg ezt már igénybe vettük, anélkül, hogy tudtunk volna róla. Az űrlapokból származó változók mindig karakterláncok, de ezeket használhattuk igaz–hamis értékű kifejezésekben is vagy számolásoknál, ha éppen arra volt szükségünk.

Tegyük fel, hogy egy felhasználót megkérdezünk, hány órát tölt a hálózaton hetente, és a választ az `$orak` nevű változóban tároljuk. Ez kezdetben szövegként kerül tárolásra.

```
$orak = "15"
if ($orak > 15)
    print "Ilyen gyakori felhasználónak akár árengedmény
           is járhatna";
```

Az `$orak` ellenőrzésekor a `"15"` karakterlánc egész számmá alakul, így a kifejezés eredménye `true` (igaz) lesz.

Az automatikus átalakítás szabályai viszonylag egyszerűek. Egész számokból vagy lebegőpontosakból álló környezetben a karakterláncok tartalmuknak megfelelően kerülnek átalakításra. Ha egy karakterlánc egész számmal kezdődik, a szám az annak megfelelő érték. Így a következő kifejezés eredménye 80:

```
4 * "20mb"
```

Ha a karakterlánc nem számmal kezdődik, 0-vá alakul. Így a következő sor 0-át ad eredményül:

```
4 * "körülbelül 20mb"
```

Ha a karakterláncban a számot egy pont követi, akkor az lebegőpontos értékké alakul. Ennek megfelelően a következő sor eredménye 4,8 lesz:

```
4 * "1.2"
```

Az ++ és -- műveletek karakterláncokra való alkalmazása különleges eset. A karakterlánc növelése, mint ahogy azt elvártuk, 1-et ad a karakterlánc átalakított értékéhez, de csak abban az esetben, ha a karakterlánc kizárólag számokból áll. Maga az új érték azonban karakterlánc marad:

```
$szoveg = "4";
$szoveg++;
print $szoveg; // kiírja az 5-öt
print gettype( $szoveg ); // azt írja ki, hogy "string"
```

Ha egy betűket tartalmazó karakterláncot próbálunk növelni, egyedül az utolsó karakter kódja fog megnőni:

```
$szoveg = "alszol";
$szoveg++;
print $szoveg; // azt írja ki, hogy "alszom"
```

Vessük össze ezt a karakterlánc növelésének egy másik megközelítésével:

```
$szoveg = "alszol";
$szoveg += 1;
print $szoveg; // 1-et ír ki
print gettype( $szoveg ); // azt írja ki, hogy "integer",
                          azaz egész szám
```

Az előző példában a `$szoveg` változó először 0-vá (egész számmá) alakul, aztán adjuk hozzá az 1-et. Ennek a műveletnek az eredménye 1, amelyet újra a `$szoveg` változóba helyezünk. A változó most már egy egész számot fog tartalmazni.

Az egész számok és a lebegőpontosak közötti automatikus átváltás még ennél is egyszerűbb. Ha egy kifejezés bármelyik operandusa lebegőpontos, a PHP a másik operandust is lebegőpontossá alakítja és az eredmény is lebegőpontos lesz:

```
$eredmeny = ( 1 + 20.0);
print gettype( $eredmeny );
// azt írja ki, hogy "double", azaz lebegőpontos
```

Fontos megjegyeznünk, hogy ha az automatikus átalakítás egy kifejezés kiértékeléséhez szükséges, a kifejezés egyik operandusa sem változik meg. Ha egy operandust át kell alakítani, akkor annak átalakított másolata fog szerepelni a kifejezés részeként.

Az adattípusok ellenőrzése

Már le tudunk kérdezni adattípusokat a `gettype()` függvénnyel, ami a hibakeresésnél jól jöhet, hiszen minden változóról meg tudjuk mondani, hogy milyen típusú. Gyakran viszont csak azt szeretnénk megnézni, hogy a változó egy bizonyos adattípusú értéket tartalmaz-e. A PHP ehhez az egyes adattípusoknak megfelelő függvényeket biztosít, amelyek bemenete egy változó vagy egy érték, visszatérési értéke pedig logikai. A függvények listáját a 16.1. táblázat tartalmazza.

16.1. táblázat Az adattípusok ellenőrzésének függvényei

Függvény	*Leírás*
`is_array()`	Igaz (`true`) értéket ad vissza, ha a paraméter tömb
`is_bool()`	Igaz (`true`) értéket ad vissza, ha a paraméter logikai érték
`is_double()`	Igaz (`true`) értéket ad vissza, ha a paraméter lebegőpontos szám
`is_int()`	Igaz (`true`) értéket ad vissza, ha a paraméter egész szám
`is_object()`	Igaz (`true`) értéket ad vissza, ha a paraméter objektum
`is_string()`	Igaz (`true`) értéket ad vissza, ha a paraméter karakterlánc

Ezek a függvények egy kicsit megkönnyítik az adattípusok ellenőrzését. Az

```
if ( gettype( $valtozo ) == "array" )
   print "ez egy tömb";
```

megegyezik a

```
if ( is_array( $valtozo ) )
   print "ez egy tömb";
```

kóddal. (A `$valtozo` ellenőrzésének második módja egy kicsit tömörebb.)

Az adattípus-váltás további módjai

Eddig az adattípus-váltás két módjával ismerkedtünk meg: vagy egy értéket alakítunk át meghatározott adattípussá, vagy a `settype()` függvényt használjuk. Ezek mellett a PHP bizonyos függvényekkel lehetővé teszi, hogy értékeket egész számmá, lebegőpontos számmá vagy szöveggé alakítsunk át. E függvények bemenete tömbön és objektumon kívül bármilyen adattípus lehet, kimenetük pedig az átalakított érték. Leírásukat a 16.2. táblázatban találhatjuk meg.

16.2. táblázat Az adattípus-váltás függvényei

Függvény	*Leírás*
`doubleval()`	Bemenete egy érték, kimenete egy azonos értékű lebegőpontos szám
`intval()`	Bemenete egy érték, kimenete egy azonos értékű egész szám
`strval()`	Bemenete egy érték, kimenete egy azonos értékű karakterlánc

Miért olyan fontosak az adattípusok?

A PHP nem követeli meg tőlünk, hogy egy változó létrehozásakor megadjuk annak adattípusát, de elvégzi az adattípus átalakításokat helyettünk, ha a kifejezésekben különböző adattípusú változókat használunk. Ha a PHP ennyire leegyszerűsíti az életünket, akkor miért van szükségünk mégis az adattípusok nyomon követésére? Azért, hogy megelőzhessük a hibákat. Képzeljünk el egy olyan függvényt, amely egy tömb kulcsait és értékeit írja ki egy böngészőbe. A PHP-ben a függvényparaméterek típusát sem adhatjuk meg, így nem írhatjuk elő, hogy a meghívó kód egy tömböt adjon át nekünk, amikor a függvényt meghatározzuk.

```
function tombKiir( $tomb )
      {
      foreach ( $tomb as $kulcs => $ertek )
          print "$kulcs: $ertek<P>";
      }
```

A következő függvény jól működik, ha tömb paraméterrel hívják meg.

```
tombKiir ( array(4, 4, 55) );
```

Ha figyelmetlenségünkben skaláris paramétert adunk át neki, az hibához vezet, amint azt a következő példa is mutatja:

```
tombKiir ( 4 );
// Warning: Non array argument supplied for foreach() in
// /home/httpd/htdocs/proba2.php on line 5
// azaz "Figyelmeztetés: a foreach() függvénynek nem tömb
 ➥ paramétert adtunk át
// a /home/httpd/htdocs/proba2.php program 5.
 ➥ sorában"
```

Azzal, hogy ellenőrizzük a kapott paraméter adattípusát, sokkal alkalmazkodóbbá tesszük a függvényt. Azt is megtehetjük, hogy a függvény szép csöndben visszatér, ha skaláris értéket kap:

```
function tombKiir( $tomb )
      {
      if ( ! is_array( $tomb ))
         return false;
      foreach ( $tomb as $kulcs => $ertek )
          print "$kulcs: $ertek<P>";
      return true;
      }
```

Most már a hívó kód ellenőrizheti a függvény visszaadott értékeit, hogy megállapíthassa, végre tudta-e hajtani feladatát.

Adatátalakítást is használhatunk, hogy a skaláris adatot tömbként használhassuk fel:

```
function tombKiir( $tomb )
      {
      if ( ! is_array( $tomb ))
         $tomb = (array) $tomb;
      foreach ( $tomb as $kulcs => $ertek )
          print "$kulcs: $ertek<P>";
      return true;
      }
```

A `tombKiir()` függvény nagyon rugalmassá vált, most már tetszőleges adattípust képes feldolgozni, akár objektumot is.

Az adattípus ellenőrzése akkor is jól jöhet, ha a függvények visszaadott értékeit szeretnénk ellenőrizni. Néhány nyelv, például a Java, minden metódusa mindig egy előre meghatározott adattípust ad vissza. A PHP-nak nincsenek ilyen megkötései, viszont az ilyen rugalmasság alkalmanként kétértelműségekhez vezethet.

Erre láttunk egy példát a tizedik órában. A `readdir()` függvény hamis (`false`) értéket ad vissza, amikor eléri a beolvasott könyvtár végét, minden más esetben viszont a könyvtár egyik elemének nevét tartalmazó karakterláncot. Hagyományosan a következőhöz hasonló szerkezetet használnánk, ha egy könyvtár elemeit szeretnénk beolvastatni:

```
$kvt = opendir( "konyvtar" );
while ( $nev = readdir( $kvt ) )
         print "$nev<br>";
closedir( $kvt );
```

Ha azonban a `readdir()` által visszaadott egyik alkönyvtár neve `"0"`, a `while` utasítás kifejezése erre a karakterláncra `false` (hamis) értéket ad vissza és befejezi a felsorolást. Ha ellenőrizzük a `readdir()` visszaadott értékének adattípusát, elkerülhetjük ezt a problémát:

```
$kvt = opendir( "konyvtar" );
while ( is_string( $nev = readdir( $kvt ) ) )
         print "$nev<br>";
closedir( $kvt );
```

A változók meglétének és ürességének ellenőrzése

Az adattípusok ellenőrzése hasznos lehet, ám előzőleg meg kell bizonyosodnunk róla, hogy a változó egyáltalán létezik-e, és meg kell néznünk, milyen értéket tartalmaz. Ezt az `isset()` függvénnyel tehetjük meg, amelynek bemenete egy változó, kimenete pedig `true` (igaz), ha a változó tartalmaz valamiféle értéket:

```
$nincsertek;
if ( isset( $nincsertek ) )
     print "a \$nincsertek változónak van értéke";
else
     print "a \$nincsertek változónak nincs értéke";
// kiírja, hogy "a $nincsertek változónak nincs értéke"
```

Azt a változót, amelyet már bevezettünk, de amelynek még nem adtunk értéket, a függvény nem beállítottként, érték nélküliként fogja kezelni.

Egy veszélyre azonban hadd hívjuk fel a figyelmet: ha 0-át vagy üres karakterláncot rendelünk egy változóhoz, a függvény azt már beállítottként, értékkel rendelkezőnek fogja tekinteni:

```
$nincsertek = "";
if ( isset( $nincsertek ) )
     print "a \$nincsertek változónak van értéke";
else
     print " a \$nincsertek változónak nincs értéke ";
// kiírja, hogy "a $nincsertek változónak van értéke"
```

Azok a változók, amelyeket a bejövő űrlapok töltenek fel értékkel, mindig beállítottként fognak szerepelni, még akkor is, ha a felhasználó egyetlen mezőt sem töltött ki adattal. Hogy az ilyen helyzetekkel megbirkózhassunk, először ellenőriznünk kell, hogy üres-e a változó. Az `empty()` függvény bemenete egy változó,

kimenete pedig `true` (igaz), ha a változó nincs beállítva vagy olyan adatot tartalmaz, mint a 0 vagy egy üres karakterlánc. Akkor is igaz értéket ad vissza, ha a változó üres tömböt tartalmaz:

```
$nincsertek = "";
if ( empty( $nincsertek ) )
     print "a \$nincsertek változó üres";
else
     print " a \$nincsertek változó adatot tartalmaz";
// kiírja, hogy "a $nincsertek változó üres"
```

További tudnivalók a tömbökről

A hetedik órában már bemutattuk a tömböket és a tömbök kezeléséhez szükséges függvényeket. Ebben a részben további függvényekkel és ötletekkel ismerkedhetünk meg.

Tömbök bejárása más megközelítésben

A PHP 4 új eszköze a `foreach` utasítás, amellyel tömbök elemeit olvashatjuk be. A könyvben lévő példák legnagyobb részében ezt használjuk. A PHP 3 esetében még egész másképpen kellett bejárni a tömböket. Ha a PHP 3-nak is megfelelő programokat szeretnénk írni vagy szeretnénk érteni a PHP 4 előtti forráskódokat is, ezzel tisztában kell lennünk.

Tömb létrehozásakor a PHP egy belső mutatót alkalmaz, amely a tömb első elemére mutat. Ennek az elemnek a kulcsához és értékéhez az `each()` függvénnyel férhetünk hozzá. Az `each()` függvény bemenete egy tömbváltozó, kimenete pedig egy négyelemű tömb. Ezek közül az elemek közül kettő számmal indexelt, kettő pedig a `"key"` (kulcs) és az `"value"` (érték) címkét viseli. A függvény meghívása után a belső mutató a vizsgált tömb következő elemére fog mutatni, kivéve, ha elérte a tömb végét, ilyenkor `false` (hamis) értéket ad vissza. Hozzunk létre egy tömböt és próbáljuk ki az `each()` függvényt:

```
$reszletek = array( "iskola" => "Képzőművészeti", "tantargy"
 ➥ => "Térbeli ábrázolás" );
$elem = each( $reszletek );
print "$elem[0]<br>"; // azt írja ki, hogy "iskola"
print "$elem[1]<p>"; // azt írja ki, hogy "Képzőművészeti"
print "$elem["key"]<br>"; // azt írja ki, hogy "iskola"
print "$elem["value"]<br>"; // azt írja ki,
                                hogy "Képzőművészeti"
```

A `$reszletek` nevű tömböt két elemmel hozzuk létre, ezután átadjuk az `each()` függvénynek és a visszaadott értéket az `$elem` nevű tömbbe helyezzük. Az `$elem` tartalmazza a `$reszletek` tömbváltozó első elemének kulcsát és értékét.

Az `each()` által visszaadott tömböt kicsit nehézkes skaláris változókhoz rendelni, de szerencsére a PHP `list()` függvénye megoldja ezt a problémát. A `list()` függvény tetszőleges számú változót elfogad bemenetként és ezek mindegyikét a jobb oldalán megadott tömbváltozó megfelelő értékeivel tölti fel:

```
$tomb = array( 44, 55, 66, 77 );
list( $elso, $masodik ) = $tomb;
print "$elso"; // azt írja ki, hogy "44"
print "<BR>";
print "$masodik"; // azt írja ki, hogy "55"
```

Vegyük észre, hogy a `list()` függvénnyel könnyedén másolhatjuk át az előző példa tömbjének elemeit a külön változókba.

Használjuk arra a `list()` függvényt, hogy az `each()` minden egyes meghívásakor két változónak adjon értéket.

```
$reszletek = array( "iskola" => "Képzőművészeti",
                    "tantargy" => "Térbeli ábrázolás" );
while ( list( $kulcs, $ertek ) = each( $reszletek ) )
     print "$kulcs: $ertek<BR>";
```

Bár a kód működni fog, valójában még egy sor hiányzik. Ha tömbünkre már használtuk a belső mutatót módosító függvények egyikét, az már nem a tömb elejére mutat. A `reset()` függvénnyel visszaállíthatjuk a mutatót a tömb kezdetére. Ez a függvény paraméterként egy tömbváltozót vár.

Így az alábbi ismerősebb szerkezet

```
foreach( $reszletek as $kulcs => $ertek );
      print "$kulcs: $ertek<BR>";
```

egyenértékű a következővel:

```
reset( $reszletek );
while ( list( $kulcs, $ertek ) = each( $reszletek ) )
     print "$kulcs: $ertek<BR>";
```

Elem keresése tömbben

A PHP 4-et megelőzőleg ha azt szerettük volna megtudni, hogy egy elem előfordul-e egy tömbben, addig kellett bejárnunk a tömböt, míg megtaláltuk az elemet vagy elértük a tömb végét. A PHP 4-ben azonban már rendelkezésünkre áll az `in_array()` függvény. Két paramétere van, az egyik a keresett érték, a másik az a tömb, amelyben keresni kívánunk. A függvény `true` (igaz) értéket ad vissza, ha megtalálja a keresett értéket, egyébként `false` (hamis) értéket kapunk.

```
$reszletek = array( "iskola" => "Képzőművészeti", "tantargy"
 ➥ => "Térbeli ábrázolás" );
if ( in_array( "Térbeli ábrázolás", $reszletek ) )
    print "A kurzus további intézkedésig
           felfüggesztve<P>\n";
```

Elemek eltávolítása a tömbből

Az `unset()` függvénnyel elemeket is eltávolíthatunk egy tömbből. A függvény bemenetéül egy változót vagy egy tömbelemet vár, majd azt minden teketória nélkül megsemmisíti. Ha a paraméter egy tömb egy eleme, a tömböt automatikusan lerövidíti.

```
unset( $proba["cim"] );
unset( $szamok[1] );
```

Az `unset()` függvény egyetlen csapdája az lehet, hogy a tömb indexei nem követik a tömb megváltozott méretét. Az előző példa tömbje a `$szamok[1]` elem eltávolítása után a következőképpen fest:

```
$szamok[0]
$szamok[2]
$szamok[3]
```

Szerencsére a `foreach()` függvénnyel ezen is gond nélkül végiglépkedhetünk.

Függvények alkalmazása a tömb összes elemére

A kifejezésekben szereplő skaláris változókat könnyen módosíthatjuk, egy tömb összes elemét megváltoztatni már egy kicsit nehezebb. Ha például a tömb összes elemének értékéhez egy számot akarunk adni, azt úgy tehetnénk meg, hogy a tömb összes elemén végiglépkedve frissítjük az értékeket. A PHP azonban ennél elegánsabb megoldást kínál.

Az `array_walk()` függvény egy tömb minden elemének kulcsát és értékét átadja egy, a felhasználó által meghatározott függvénynek. A függvény bemenete egy tömbváltozó, egy, a függvény nevét megadó karakterlánc érték, és egy elhagyható harmadik paraméter, amelyet a választott függvénynek szeretnénk még átadni.

Vegyünk egy példát. Van egy adatbázisból kinyert ártömbünk, de mielőtt munkához kezdenénk vele, az összes árhoz hozzá kell adnunk a forgalmi adót.
Először azt a függvényt hozzuk létre, amely hozzáadja az adót:

```
function ado_hozzaado( &$ertek, $kulcs, $adoszazalek )
      {
      $ertek +=( ($adoszazalek/100) * $ertek );
      }
```

Az `array_walk()` számára készített összes függvénynek egyértéket, egy kulcsot és egy elhagyható harmadik paramétert kell várnia.

Ha paraméterként nem értéket szeretnénk átadni, hanem egy változót, melyben tükröződhetnek a függvény okozta változtatások a függvény hatókörén kívül is, a függvény-meghatározásban egy ÉS jelet (`&`) kell az adott paraméter elé írni. Ez fogja biztosítani, hogy ha a függvényen belül módosítjuk az értéket, az megjelenik a függvényen kívül a tömbben is. Ez az oka annak is, hogy példánkban az `ado_hozzaado()` függvénynek nincs szokásos értelemben vett visszatérési értéke.

Most hogy megvan a függvényünk, máris meghívhatjuk az `array_walk()` függvényt a megfelelő paraméterekkel:

```
function ado_hozzaado( &$ertek, $kulcs, $adoszazalek )
      {
      $ertek +=( ($adoszazalek/100) * $ertek );
      }
$arak = array( 10, 17.25, 14.30 );
array_walk( $arak, "ado_hozzaado", 10 );
foreach( $arak as $ertek )
print "$ertek<BR>";
// kimenete:
// 11
// 18.975
// 15.73
```

A `$arak` tömbváltozót az `ado_hozzaado()` függvény nevével együtt átadjuk az `array_walk()` függvénynek. Az `ado_hozzaado()` függvénynek tudnia kell az érvényes adókulcsot. Az `array_walk()` harmadik, elhagyható paramétere átadódik a megnevezett függvénynek és ezzel elérjük, hogy az értesüljön az adókulcsról.

Tömbök egyéni rendezése

A kulcs vagy érték alapján történő rendezéssel már megismerkedtünk, nem mindig szoktunk azonban ilyen egyszerűen rendezni tömböket. Előfordulhat, hogy többdimenziós tömbbe beágyazott értékek alapján vagy a szokványos alfanumerikus összehasonlítástól eltérő szempont szerint szeretnénk rendezni.

A PHP lehetővé teszi, hogy magunk határozzuk meg az összehasonlító tömbrendező függvényeket. Számmal indexelt tömbök esetében ilyenkor az `usort()` függvényt kell meghívnunk, melynek bemenete a rendeznivaló tömb és annak a függvénynek a neve, amely egy elempár összehasonlítását képes elvégzi.

Az általunk meghatározott függvénynek két paramétert kell elfogadnia, amelyek az összehasonlítandó tömbértékeket tartalmazzák. Ha a feltételek alapján ezek azonosak, a függvény a `0` értéket kell, hogy visszaadja, ha a tárgytömbben az első paraméternek a második előtt kell jönnie, akkor `-1`-et, ha pedig ez első paraméternek kell a második után szerepelnie, akkor `1`-et.

A 16.1. programban azt láthatjuk, hogyan használjuk az `usort()` függvényt egy többdimenziós tömb rendezésére.

16.1. program Többdimenziós tömb rendezése mező alapján az usort() függvénnyel

```
 1: <?php
 2: $termekek = array(
 3:           array( "nev"=>"HAL 2000",    "ar"=>4500.5  ),
 4:           array( "nev"=>"Modem",       "ar"=>55.5    ),
 5:           array( "nev"=>"Nyomtató",    "ar"=>2200.5  ),
 6:           array( "nev"=>"Csavarhúzó", "ar"=>22.5    )
 7:           );
 8: function arHasonlito( $a, $b )
 9:      {
10:      if ( $a["ar"] == $b["ar"] )
11:          return 0;
12:      if ( $a["ar"] < $b["ar"] )
13:          return -1;
14:      return 1;
15:      }
16: usort( $termekek, "arHasonlito" );
17: foreach ( $termekek as $ertek )
18:      print $ertek["nev"] ":" $ertek["ar"] "<br>\n";
19: ?>
```

Először létrehozzuk a `$termekek` tömböt, amelyet az egyes értékek ár mezője alapján szeretnénk rendezni. Ezután létrehozzuk az `arHasonlito()` függvényt, amelynek két paramétere van, `$a` és `$b`. Ezek tartalmazzák azt a két tömböt, amely a `$termekek` tömb második szintjét alkotja. Összehasonlítjuk az `"ar"` elemeiket és ha a két ár azonos, `0`-át, ha az első kevesebb, mint a másik, `-1`-et, egyébként pedig `1`-et adunk vissza.

Miután meghatároztuk a rendező függvényt és a tömböt is, meghívhatjuk az `usort()` függvényt, amelynek átadjuk a `$termekek` tömböt és az összehasonlító függvény nevét. Az `usort()` ismételten meghívja függvényünket, mindig átadja annak a `$termekek` egy elemét és felcseréli az elemeket, a visszaadott értékeknek megfelelően. Végül végigléptetünk a tömbön, hogy megmutassuk az új elrendezést.

Az `usort()` függvényt számmal indexelt tömbök esetében használjuk. Ha más egyéni rendezést szeretnénk egy asszociatív tömbön végrehajtani, használjuk az `uasort()` függvényt. Az `uasort()` úgy rendez, hogy megtartja a kulcsok és az értékek közötti társítást is. A 16.2. program az `uasort()` használatát egy asszociatív tömb rendezésén keresztül mutatja be.

16.2. program Többdimenziós tömb rendezése mező alapján az uasort() függvénnyel

```
 1: <?php
 2: $termekek = array(
 3:          "HAL 2000" => array( "szin" =>"piros",
                "ar"=>4500.5 ),
 4:          "Modem" => array( "szin" =>"kék",
                "ar"=>55.5 ),
 5:          "Nyomtató" => array( "szin" =>"zöld",
                "ar"=>2200.5 ),
 6:          "Csavarhúzó" => array( "szin" =>"piros",
                "ar"=>22.5 )
 7:          );
 8: function arHasonlito( $a, $b )
 9:      {
10:      if ( $a["ar"] == $b["ar"] )
11:          return 0;
12:      if ( $a["ar"] < $b["ar"] )
13:          return -1;
14:      return 1;
15:      }
16: uasort( $termekek, "arHasonlito" );
```

16.2. program (folytatás)

```
17: foreach ( $termekek as $kulcs => $ertek )
18:      print "$kulcs: " $ertek["ar"] "<br>\n";
19: ?>
```

Az `uksort()` függvénnyel az asszociatív tömbökön a kulcsok alapján végezhetünk egyéni rendezést. Az `uksort()` pontosan ugyanúgy működik, mint az `usort()` és az `uasort()`, azzal a különbséggel, hogy az `uksort()` a tömb kulcsait hasonlítja össze.

A 16.3. programban az `uksort()` függvénnyel az egyes kulcsok karakterszáma alapján rendezünk egy tömböt. Megelőzve a következő óra anyagát, az `strlen()` függvénnyel állapítjuk meg a kulcsok hosszúságát. Az `strlen()` függvény bemenete egy karakterlánc, visszaadott értéke pedig annak hosszúsága karakterben mérve.

16.3. program Asszociatív tömb rendezése kulcshosszúság alapján az uksort() függvénnyel

```
 1: <?php
 2: $ikszek = array(
 3:                  "xxxx" => 4,
 4:                  "xxx" => 5,
 5:                  "xx" => 7,
 6:                  "xxxxx" => 2,
 7:                  "x" => 8
 8:                );
 9: function hosszHasonlito( $a, $b )
10:      {
11:      if ( strlen( $a ) == strlen( $b ) )
12:          return 0;
13:      if ( strlen( $a ) < strlen( $b ) )
14:          return -1;
15:      return 1;
16:      }
17: uksort( $ikszek, "hosszHasonlito" );
18: foreach ( $ikszek as $kulcs => $ertek )
19:      print "$kulcs: $ertek <br>\n";
20:
21: // a kimenet:
```

16.3. program (folytatás)

```
22: // x: 8
23: // xx: 7
24: // xxx: 5
25: // xxxx: 4
26: // xxxxx: 2
27:
28: ?>
```

Összefoglalás

Az óra során a tömbökkel és adattípusokkal kapcsolatos ismeretekben mélyedtünk el. Megtanultuk, mi történik, ha összetett adattípust skalárissá alakítunk és fordítva. Megtudtuk, hogyan kezeli a PHP a különböző adattípusokat egy kifejezésben, hogyan határozza meg automatikusan az eredmény adattípusát helyettünk. Megismerkedtünk számos függvénnyel; például az `is_array()`-jel, amely különféle adattípusokat ellenőriz, vagy az `intval()`-lal, amely az adatot egész értékűvé alakítja át. Megtanultuk, hogyan lehet a PHP-ben hagyományos módon tömböt bejárni, az `each()` és a `list()` függvénnyel. Az `in_array()` függvénnyel képesek vagyunk ellenőrizni, hogy létezik-e egy adott érték egy tömbben, és el tudunk távolítani egy elemet egy tömbből az `unset()` függvénnyel. Az `array_walk()` függvénnyel már egy tömb összes elemén is végezhetünk műveleteket, végül azt is tudjuk, hogyan használjuk az `usort()`, az `uasort()` és az `uksort()` függvényeket arra, hogy a tömbök egyéni rendezését végezzük.

Kérdések és válaszok

A PHP minden tömbkezelő függvényével megismerkedtünk?
Nem, az egész könyv sem lenne elég az összes tömbkezelő függvény bemutatásához. Teljes listájukat és leírásukat a `http://www.php.net/manual/ref.array.php` weboldalon találhatjuk.

Műhely

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

Kvíz

1. Melyik az a függvény, amellyel adattípusokat tetszőleges más adattípussá alakíthatunk?

2. Sikerülhet ez függvény nélkül is?

3. Mit ír ki a következő kód?
```
print "four" * 200;
```

4. Hogyan határoznánk meg, hogy egy adott változó tömb-e?

5. Melyik függvény adja vissza paraméterének értékét egész számként?

6. Hogyan ellenőrizzük, hogy egy változónak adtunk-e már értéket?

7. Hogyan ellenőrizzük, hogy egy változó üres értéket (például 0-át vagy üres karakterláncot) tartalmaz?

8. Melyik függvénnyel törölnénk egy tömb egy elemét?

9. Melyik függvénnyel rendeznénk egy számmal indexelt tömböt?

Feladatok

1. Nézzük végig még egyszer a könyv során megoldott feladatokat. Alakítsunk át minden foreach utasítást úgy, hogy az megfeleljen a PHP 3 követelményeinek is.

2. Hozzunk létre egy vegyes adattípusú tömböt. Rendeztessük a tömböt adattípusok szerint.