II. RÉSZ

A PHP nyelv

4. ÓRA

Az alkotóelemek

Ebben az órában már mélyebben elmerülünk a nyelv rejtelmeiben. Az alapokkal kezdünk, így a kezdő programozónak rengeteg új információt kell majd feldolgoznia. Aggodalomra azonban semmi ok, bármikor vissza lehet lapozni a könyvben. A lényeg, hogy a fejezetben leírtak megértésére törekedjünk, ne csupán a memorizálásra.
A gyakorlott programozóknak is ajánlatos legalábbis átlapozniuk ezen óra anyagát. Az órában a következőket tanuljuk meg:

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

Változók

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

```
$a;
$egy_hosszabb_valtozonev;
$elalszom_zzzzzzzzzzzzz;
```

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

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

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

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

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

```
print $szam1;
```

hatása megegyezik a

```
print 8;
```

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

Dinamikus változók

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

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

az megegyezik azzal, mintha ezt írnánk:

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

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

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

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

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

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

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

azonos a

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

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

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

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

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

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

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

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

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

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

4.1. program Dinamikusan beállított és elért változók

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

4.1. program (folytatás)

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

Hivatkozások a változókra

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

4.2. program Változók érték szerinti hozzárendelése

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

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

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

4.3. program Változóra mutató hivatkozás

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

A 4.2. példaprogramhoz képest a változás mindössze egy karakter.
Az $egyikValtozo elé tett & jel gondoskodik róla, hogy az érték másolata helyett a $masikValtozo-ba a változóra mutató hivatkozás kerül. Ezután a $masikValtozo elérésekor az $egyikValtozo-ra vonatkozó műveletek eredményét láthatjuk. Más szavakkal: mind az $egyikValtozo, mind a $masikValtozo ugyanazt a „tárolódobozt” használja, így értékeik mindig egyenlők. Mivel ez az eljárás kiküszöböli az egyik változóból a másikba történő értékmásolás szükségességét, csekély mértékben növelheti a teljesítményt. Hacsak nem használunk nagyon gyakran értékadásokat, ez a teljesítménybeli növekedés alig érezhető.

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

Adattípusok

A különféle típusú adatok több-kevesebb helyet foglalnak a memóriában, a nyelv pedig mindegyiket némileg más módon kezeli. Ezért néhány programozási nyelv megköveteli, hogy a programozó előre meghatározza a változótípusát. A PHP 4 gyengén típusos, ami azt jelenti, hogy az adattípusokat úgy kezeli, mintha a típus az adathoz rendelt kiegészítő információ lenne. A PHP vegyes megközelítést használ. Egyfelől ez azt jelenti, hogy a változók rugalmasan használhatók: egyszer karakterlánc, másszor esetleg szám lehet bennük. Másfelől, nagyobb méretű programokban zavar forrása lehet, ha egy adott típusú változót várunk egy változóban, miközben valami teljesen más van benne.
A 4.1. táblázat a PHP 4-ben elérhető hat fontosabb adattípust tartalmazza, rövid leírással és példával.

4.1. táblázat Adattípusok

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

Az adattípusok közül a tömböket és az objektumokat későbbre hagyjuk.
A változó típusának meghatározására a PHP 4 beépített `gettype()` függvényét használhatjuk. Ha a függvényhívás zárójelei közé változót teszünk, a `gettype()` egy karakterlánccal tér vissza, amely a változó típusát adja meg. A 4.4. példaprogram egy változóhoz négy különböző típusú értéket rendel, majd meghívja a `gettype()` függvényt.

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

4.4. program Változó típusának vizsgálata

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

Az előbbi program a következő kimenetet eredményezi:

```
integer
string
double
boolean
```

Az integer egész szám, vagyis olyan szám, amelyben nincs tizedesjegy („tizedespont”).

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

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

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

Típus módosítása a settype() segítségével

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

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

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

4.5. program (folytatás)

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

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

Típus módosítása típusátalakítással

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

4.6. program Változó típusának módosítása típusátalakítás segítségével

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

4.6. program (folytatás)

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


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

Műveletjelek és kifejezések

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

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

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

Kapcsoljunk össze két operandust és hozzunk létre egy új értéket:

```
4 + 5
```

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

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

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

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

Hozzárendelés

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

```
$nev = "Kőműves Kelemen";
```

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

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

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

Aritmetikai műveletek

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

4.2. táblázat Aritmetikai műveletek

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

Összefűzés

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

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

kifejezés értéke:

```
"Para Zita"
```

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

További hozzárendelő műveletek

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

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

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

például egyenértékű a következővel:

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

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

4.3. táblázat Néhány összetett hozzárendelő műveletjel

Műveletjel	*Példa*	*Egyenértékű kifejezés*
`+=`	`$x += 5`	`$x = $x + 5`
`-=`	`$x -= 5`	`$x = $x - 5`
`*=`	`$x *= 5`	`$x = $x * 5`
`/=`	`$x /= 5`	`$x = $x / 5`
`%=`	`$x %= 5`	`$x = $x % 5`
`.=`	`$x .= "próba"`	`$x = $x . "próba"`

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

Összehasonlítás

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

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

```
$x < 5
```

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

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

4.4. táblázat Összehasonlító műveletek

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

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

Bonyolultabb összehasonlító kifejezések létrehozása logikai műveletek segítségével

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

```
true || false
```

A fenti eredménye true.

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

```
true && false
```

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

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

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

4.5. táblázat Logikai műveletek

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

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

Egész típusú változók értékének növelése és csökkentése

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

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

De használhatunk összetett értékadó-összeadó műveletjelet is:

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

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

```
$x++;  // $x értéke eggyel nő
```

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

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

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

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

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

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

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

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

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

A műveletek kiértékelési sorrendje

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

```
4 + 5
```

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

```
4 + 5 * 2
```

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

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

```
( 4 + 5 ) * 2
```

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

4.6. táblázat A műveletek elsőbbsége csökkenő sorrendben.

Műveletjelek
! ++ -- (típusátalakítás)
/ * %
+ - .
< <= => >
== === != !==
&&
||
= += -= /= %= .=
and
xor
or

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

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

Állandók

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

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

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

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

4.7. program Állandó létrehozása

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

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

Minden programban használható állandók

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

Összefoglalás

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

Kérdések és válaszok

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

Muszáj követnünk a változók nevére vonatkozó szokásokat?
A cél mindig az, hogy a program egyszerűen olvasható és érthető legyen. Az olyan változónevek, mint az `$abc12345` nem mondanak semmit a változó programbeli szerepéről és elgépelni is könnyű azokat. Ezért érdemes rövid és jellemző neveket választanunk.
Az `$f` név, bár rövid, bizonyára nem árul el semmit a változó szerepéről.
Ezt most talán nehéz elhinni, de amikor egy hónap elteltével próbáljuk meg folytatni a program írását, majd megtapasztaljuk. A `$fajlnev` már sokkal jobb választás.

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

Műhely

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

Kvíz

1. Az alábbiak közül melyek NEM lehetnek változónevek?
```
$egy_felhasznalo_altal_elkuldott_ertek
$44444444444xyz
$xyz44444444444
$_____________szamlalo_____________
$az elso
$fajl-nev
```

2. Hogyan használhatjuk fel az alábbi változót dinamikus változó létrehozására? A változónak adjuk a `4` értéket! Hogyan érhetjük el az új változót?
```
$en_valtozom = "dinamikus";
```

3. Milyen kimenetet eredményez az alábbi programsor?
```
print gettype("4");
```

4. Mit ír ki az alábbi néhány sor?
```
$proba_valtozo = 5.4566;
settype ( $proba_valtozo, "integer");
print $proba_valtozo;
```

5. Az alábbi sorok melyike nem tartalmaz kifejezést?
```
4;
gettype(44);
5/12;
```

6. Az előző kérdésben szereplő sorok melyikében van műveletjel?

7. Az alábbi kifejezés milyen értéket ad vissza?
```
5 < 2
```
Milyen típusú a kérdéses érték?

Feladatok

1. Készítsünk programot, amely legalább öt különböző változót tartalmaz. Adjunk nekik különböző típusú értékeket, majd használjuk a `gettype()` függvényt a változók típusainak meghatározására.

2. Rendeljünk értéket két változóhoz. Használjuk az összehasonlító műveleteket annak eldöntésére, hogy az első változó értéke

   - azonos-e a második változó értékével
   - kisebb-e a második változó értékénél
   - nagyobb-e a második változó értékénél
   - kisebb-e a második változó értékénél vagy egyenlő-e azzal

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