**Változók**

A változók a jövőbeni referenciáknak tárolnak el értékeket. Ez az érték megváltozhat, amikor mi akartjuk; ezért nevezzük őket változóknak. Lássuk ezeket egy példában. Ments el ezt a kódot az index.php fájlodban:

<?php

$a = 1;

$b = 2;

$c = $a + $b;

Echo $c; // 3

A fenti kódban nekünk három változónk van: $a aminek az értéke 1, $b aminek 2, és $c ami az $a és az $b összegét tárolja el. A böngésző az $c értékét kéne kiírja, ami 3.

Hozzárendelni egy értéket egy változóhoz azt jelenti, hogy értéket adni neki, ezt az egyenlőség jellel lehet megtenni, mint ahogy a fenti példában is látszik. Ha nem rendelünk hozzá értéket, akkor azt a PHP észre fogja venni, amikor megnézi annak tartalmát. Ilyenkor csak egy figyelmeztető üzenetet kapunk, hogy valami nincs rendben, de ez nem egy nagy baj, és a kódunkat is folytathatjuk. Annak a változónak amihez nem volt rendelve semmilyen érték, annak az értéke null lesz.

PHP-ben a változók mindig $ jellel kezdődnek, majd ez után adjuk meg a vátozó nevét. Hogy érvényes legyen a változó neve mindig betűvel vagy egy alúlvonallal kell kezdődjön, amit akármilyen betűk, számok és/vagy alúlvonalak kombinációja követhet. Nézzünk néhány példát:

<?php

$\_some\_value = ’abc’; // valid

$1number = 12.3; // not valid!

$some$signs% = ’&^%’; // not valid!

$go\_2\_home = „ok”; // valid

$go\_2\_Home = ’no’; // this is a different variable

$isThisCamelCase = true; // camel case

Emlékezzünk, hogy minden ami a // után van az kommentbe van, és a PHP figyelmen kívűl hagyja azt.

Ebben a kódrészletben láthatjuk, hogy azok a változó nevek, mint $\_some\_value és $go\_2\_home érvényesek. $1number és $some$signs% nem érvényesek, mivel egy számmal kezdődnek, vagy érvénytelen szimbólumokat tartalmaznak. Mivel a nevek kis- és nagybetű érzékenyek, ezért a $go\_2\_home és a $go\_2\_Home két különböző változónak számít. És végül ott van a CamelCase konvenció, amit sok programozó preferál.

**Adattípusok**

A számokon kívűl még több mindent is hozzá lehet rendelni a változókhoz. A PHP-nak nyolc primitív típusa van, de most egyelőre nézzük meg a négy skaláris típust:

* **Boolean:** ez csak igaz és hamis értékeket fogad el
* **Integer:** ide azok a számok tartoznak, amik nem tartalmaznak tizedes pontot/vesszőt, mint például a 2 vagy 5
* **Float:** ezek a tizedesszámok, mint például a 2.3
* **String:** ezek a karakterláncok, amiket simpla vagy dupla idézőjelek közé kell írni, mint ’ez’ vagy „az”

Annak ellenére, hogy a PHP meghatározza ezeket a típusokat, lehetővé teszi a felhasználónak, hogy különböző típusú értékeket ugyanahhoz a változóhoz hozzá tudja rendelni. Nézzük a következő kódban, hogy ez hogyan is műkődik:

<?php

$number = 123;

Var\_dump($number);

$number = ’abc’;

var\_dump($number);

Ha megnézzük az eredményt a böngészőben, akkor a következőt kéne kapjuk:

int(123) string(3) „abc”

A kód először az 123 értéket fogja hozzárendelni a $number változóhoz. Mivel az 123 egy intiger, ezért a változó is egy intiger, int, lesz. Ez az amit látunk, amikor a var\_dump változóba irassuk ki a tartalmát. Ezután, egy másik értéket rendelünk ugyanahhoz a változóhoz, most egy stringet. Mikor újra kiíratjuk, akkor látjuk, hogy a változó típusa megváltozott int-ről string-re, és a PHP nem jelzett ki semmit. Ezt típus manipulációnak nevezzük.

Lássunk egy másik kódot:

<?php

$a = „1”;

$b = 2;

var\_dump($a + $b); // 3

var\_dump($a . $b); // 12

Már tudjuk, hogy a + operátor két szám változó összegét téríti vissza. Később meglátjuk, hogy a . operátort két string összefűzésére használjuk. Így, a fenti kódban egy stringet és egy intigert megadva láthatjuk, hogy a kód próbálja összeadni és össefűzni őket.

Mikor össze próbáljuk őket adni, akkor a PHP tudja, hogy ehhez szükség van két szám értékre, ezért a stringet intigerré próbálja alakítani. Ebben az esetben ez egyszerű, mivel a string egyben egy szám is. E miatt az első eredmény 3 lett (*1 + 2*).

Az utolsó sorban egy string összefűzés van. Nekünk van egy intiger a $b-ben, ezért a PHP azt egy stringgé alakítja át -ami „2”– és összefűzi azt a másik stringgel, az „1”-el. Az eredmémy így „12” lesz.

**Típus manipuláció**

A PHP abban az esetben próbálja meg átváltani egy változó adattípusát, hogy ha egy kontextusban annak a változónak más típus kell. De a PHP magától nem változtatja meg a változókat és azok tpusait. E helyett inkább azok értékeit fogja és azokat változtatja meg, érintetlenül hagyva a változót.

**Operátorok**

A változók használata jó, de ha nem tudjuk őket egymással kapcsolatba hozni, akkor nincs sok amit tehetünk. Az operátorok olyan elemek, amik vesznek egy kifejezést –operandusokat- és egy eljárást hajt végre rajtuk, hogy egy eredményt adjon. A leghétköznapibb példák operátorokra az aritmetikai operátrok, amiket már látunk korábban.

Egy kifejezés lehet akármi, aminek van egy értéke. Változók, számok, vagy szövegek mind példák kifejezésekre, de később meglátjuk, hogy ezek komplikáltabbak lesznek. Az operátorok egy meghatározott értéket várnak el a kifejezésekben, példáu az aritmetikai operátorok vagy intigert vagy floatot várnak el. De mint tudjuk, a PHP foglalkozik a kifejezés típusok átváltásáról, amikor az lehetséges.

Nézzük meg a legfontosabb operátorok csoportját.

**Aritmetikai operátorok**

Mint tudjuk, az aritmetikai operátorok nagyon intuitívak. Összeadás, kivonás, szorzás, és osztás (+,-,\* és /). A modulusz (%) visszaadja két operátor osztásának a maradékát. A hatványozás (\*\*) az első operátort a második operátor hatványára emeli. Végül a negativitás (-) , ami negatívvá teszi az operátort. Ez az utolsó az egyetlen, ami csak egy operátort kér.

Lássunk pár példát:

<?php

$a = 10;

$b = 3;

var\_dump($a + $b); // 13

var\_dump($a - $b); // 7

var\_dump($a \* $b); // 30

var\_dump($a / $b); // 3.3333...

var\_dump($a % $b); // 1

var\_dump($a \*\* $b); // 1000

var\_dump(-$a); // -10

Mint látható, ezeket nagyon egyszerű megérteni!

**Hozzárendelő operátorok**

Már ezt is tudjuk, mert már használtuk egy korábbi példában. A hozzárendelő operátor hozzárendeli egy kifejezés eredményét egy változóhoz. Most már tudjuk, hogy egy kifejezés lehet olyan egyszerű, mint egy szám, vagy például egy aritmetikai operátor sorozat eredménye. A következő példában egy kifejezés eredményét rendeljük hozzá egy változóhoz.

<?php

**$a = 3 + 4 + 5 - 2;**

var\_dump($a); // 10

Van egy pár hozzárendelő operátor, ami parancsként/rövidebb út műkődik. Meg lehet őket úgy íni, hogy összekapcsolunk egy aritmetikai és egy hozzárendelő oerátort. Lássunk pár példát:

$a = 13;

**$a += 14; // same as $a = $a + 14;**

var\_dump($a);

**$a -= 2; // same as $a = $a - 2;**

var\_dump($a);

**$a \*= 4; // same as $a = $a \* 4;**

var\_dump($a);

**Összehasonlító operátorok**

Az összehasonlító operátorok az operátorok leggyakrabban használt csoportjába tartoznak. Itt két operandus van, amiket össze kell egymással hasonlítani, eredményként pedig egy booleant adva vissza, aminek az eredménye igaz vagy hamis lehet.

A négy legintuitívabb összehasonlítók: < (kisebb, mint), <= (kisebb vagy egyenlő), > (nagyobb, mint), és >= (nagyobb vagy egyenlő). Van még egy speciális operátor a < = > ami egy intigert ad vissza boolean helyett. Mikor a-t b-hez hasonlítjuk, az eredmény kisebb, mint 0 lesz, ha a kisebb, mint b, 0 ha a egyenlő b-vel, és nagyobb, mint 0 ha a nagyobb, mint b. Lássunk pár példát:

<?php

var\_dump(2 < 3); // true

var\_dump(3 < 3); // false

var\_dump(3 <= 3); // true

var\_dump(4 <= 3); // false

var\_dump(2 > 3); // false

var\_dump(3 >= 3); // true

var\_dump(3 > 3); // false

var\_dump(1 <=> 2); // int less than 0

var\_dump(1 <=> 1); // 0

var\_dump(3 <=> 2); // int greater than 0

Vannak olyan összehasonlító operátorok, amit meg tudják nekünk mondani ha két kifejezés egyenlő e egymással, vagy sem, de vigyázni kell a típus váltással. Az == (egyenlő) egyenlővé tesz két kifejezést az átváltás után, ez mindkét típust megpróbálja közös típusra váltani, és úgy össszehasonlítani őket. E helyett az === (egyező) operátor két típust átváltás nélkül tud vizsgálni, szóval hiába uyanúgy néz ki a kettő, lehet nem uyanaz a típus, az eredmény hamisat fog így adni. Hasonló vonatkozik a != vagy <> operátorokra (nem egyenlő) és != (nem hasonló):

<?php

$a = 3;

$b = '3';

$c = 5;

var\_dump($a == $b); // true

var\_dump($a === $b); // false

var\_dump($a != $b); // false

var\_dump($a !== $b); // true

var\_dump($a == $c); // false

var\_dump($a <> $c); // true

Láthatjuk, hogy ha azt kérdezzük, hogy egy string és egy intiger egyenlő e egymással, ha ugyanaz a szám, akkor az igaz lesz; a PHP először mindkettőt átváltja ugyanarra a típusra. Másodjára, ha megkérdezzük, akkor azt mondja, hogy különbözőek.

**Logikai operátorok**

A logikai operátorok egy logikai műveletet – más néven bináris műveletet – alkalmaznak az

Operandusokra, visszaadva egy boolean választ. A legtöbbet használtak a ! (nem), && (és), és | | (vagy), az && egy igaz értéket fog visszaadni, ha a két operandus egyenlő, | | egy igaz értéket fog visszaadni, ha mindkét operandus igaz, a ! egy negatív értéket fog visszaadni, ha az igaz volt. Nézzünk pár példát:

<?php

var\_dump(true && true); // true

var\_dump(true && false); // false

var\_dump(true || false); // true

var\_dump(false || false); // false

var\_dump(!false); // true

**Növelő és csökkentő operátorok**

A növelő/csökkentő operátorok is rövidítések, mint a += vagy - =, és csak változóknál műkődnek. Négy féle van belőlük, amik különös figyelmet kívánnak. Az első kettőt már láttuk:

* ++ : bal oldalról, a változót megnöveli 1-el, és visszaadja annak eredményét. Jobb oldalról, előbb a tartalmat adja meg, majd azt megnöveli 1-el.
* - - : ugyanúgy műkődik, mint a ++, csak elvesz 1-et

<?php

$a = 3;

$b = $a++; // $b is 3, $a is 4

var\_dump($a, $b);

$b = ++$a; // $a and $b are 5

var\_dump($a, $b);

A fenti kódban az első művelet a $b-nél történik, használjuk az $a++. A $b-hez az $a első értékét fogja hozzárendelni, ami 3, és csak utána növeli meg 1-el. A második műveletben először az $a-t fogja megnövelni 1-el, ami 5 lesz, majd azt rendeli hozzá a $b-hez.

**Operátorok elsőbbsége**

Több operátort is lehet íni egy kifejezésbe, ami hosszú lesz, és figyelni kell az operátorok helyes használatára is, mivel egyeseknek elsőbbségük van másokkal szemben. Az alábbi táblázat megmutatja az eddig tanult operátorok fontossági sorrendjét:

|  |  |
| --- | --- |
| **Operator** | **Type** |
| \*\* | Arithmetic |
| ++,- - | Increasing/decreasing |
| ! | Logical |
| \*,/,% | Arithmetic |
| +,- | Arithmetic |
| <,<=,>,>= | Comparison |
| ==,!=,===,!== | Comparison |
| && | Logical |
| | | | Logical |
| =,+=,-=,\*=,/=,%=,\*\*= | Assignment |

A fenti táblázat megmutatja, hogy a 3+2\*3 úgy hajtódik végre, hogy előbb a szorzás, utána az összeadás, vagyis az eredmény így 9 lesz 15 helyett. Ha egy bizonyos sorrendben akarunk elvégezni olyan műveleteket, amik fontossági sorrendje különböző, akkor zárójelek közé rakjuk be azokat. Így a (3+2)\*3 úgy fog végrehajtódni, hogy előbb az összeadás utána a szorzás végződik el, az eredmény pedig 15 lesz és nem 9.

Nézzünk egy példát, hogy tisztázzuk ezt a trükkös részletet:

<?php

$a = 1;

$b = 3;

$c = true;

$d = false;

**$e = $a + $b > 5 || $c; // true**

var\_dump($e);

**$f = $e == true && !$d; // true**

var\_dump($f);

**$g = ($a + $b) \* 2 + 3 \* 4; // 20**

var\_dump($g);

Ez a fenti példa végtelen tud lenni, és még mindig nem lenne tisztázva minden művelet, ezér egyszerűen elmagyarázva. Az első kijelölt sorban egy aritmetikat, egy összehasonlító és egy logikai operátor kombináció van. Mivel nincsenek zárójelek, ezért fontossági sorrendben fognak végrehajtódni. Szóval először az $a + $b fog 4-et adni. Majd a 4 > 5 leesz, ami hamisat fog visszaadni. Végül pedig false || $c ($c is true) ami igazat eredményez.

A második kijelölt sor egy kicsit bonyolultabb. A következőképpen oldjuk meg azt, !$d is !false, vagyis true. Ezután $e == true && true. Először az összehasonlítást hajtjuk végre $e==true. Tudva, hogy $e true, az összehasonlítás eredménye igaz lesz.

Próbáljunk otthon is ilyen példákat oldani.

**Műveletek stringekkel**

A stringes műveletek a valóságban nagyon egyszerűek. Az olyan műveletek, mint megnézni ha egy string tartalmaz egy bizonyos elemet vagy hogy egy bizonyos betű benne van e, nagyon könnyű ezeket elvégezni. De a programozásban a stringek csak karakterek összefűzéséből állnak, amiket nem lehet átlátni mikor keresel valamit. E helyett meg kell nézni külön-külön és figyelemmel kísérni annak tartalmát. Ebben az esetben azok a nagyon könnyű műveletek már nem is olyan egyszerűek.

Szerencsére a PHP segítséget nyújt ebben nekünk, az előre definiált függvényeivel. Itt meg lehet találni az összes ilyen függvényt: http://php.net/manual/en/ref.strings.php , de most azokról beszélünk, amiket eddig láthattunk. Itt van egy pár példa:

<?php

$text = ' How can a clam cram in a clean cream can? ';

echo strlen($text); // 45

$text = trim($text);

echo $text; // How can a clam cram in a clean cream can?

echo strtoupper($text); // HOW CAN A CLAM CRAM IN A CLEAN CREAM CAN?

echo strtolower($text); // how can a clam cram in a clean cream can?

$text = str\_replace('can', 'could', $text);

echo $text; // How could a clam cram in a clean cream could?

echo substr($text, 2, 6); // w coul

var\_dump(strpos($text, 'can')); // false

var\_dump(strpos($text, 'could')); // 4

A fenti kódba különböző függvényekkel vizsgáltunk egy stringet:

* strlen : visszaadja eredménynek, hogy hány karaktert tartalmaz a string
* trim : a string-ből kitörli az összes fehér karaktert
* strtoupper és strtolower : ezek a stringet csak kisbetűkre vagy csak nagybetűkre alakítják át
* str\_replace : egy megadott stringel helyettesíti a meglévőt
* substr : felbontja a stringet a megadott paraméterek között
* strpos : a megadott string első előfordulási helyét adja meg, ha az nem létezik akkor hamis eredményt ad vissza

Pluszba még itt van egy operátor a stringeknek (.), ami két stringet vagy két stringgé alakított változót tud összefűzni. Ennek a használata nagyon egyszerű. Például:

<?php

$firstname = 'Hiro';

$surname = 'Nakamura';

echo 'I am ' . $firstname . ' ' . $surname . '!';

Egy másik észrevétel a stringekkel kapcsolatban. Eddig a stringeket szimpla és dupla idézőjelek közé raktuk. De, a szimpla idézőjelek közötti string az marad az eredetiben, míg a dupla idézőjelek közé zárt stringre még pár szabály vonatkozik, mielőtt megjelenítődne az eredmény. Itt van ké példa, amiben különbözik a dupla idézőjeles string a szimplától:

* **Kiugró karakterek**: ezek olyan arakterek, amiket nem lehet könnyen meghatározni. Ilyenek az új sort és a bekezdést jelőlő karakterek. Hogy megírjuk őkat szükségünk van a \ jelre.
* **Változók bővítése**: lehetővé teszi változók beillesztését egy stringbe, de tartalmaznia kell az $ jelet is

Nézzük a következő példát:

<?php

$firstname = 'Hiro';

$surname = 'Nakamura';

echo "My name is $firstname $surname.\nI am a master of time and

space. \"Yatta!\"";

Ez a kód a következő eredményt jeleníti meg a böngészőben:

My name is Hiro Nakamura.

I am a master of time and space. "Yatta!"

Itt a \n beszúr egy új sort. \” hozzáad dupla idézőjelet (ettől is meg kell szabadulni, különben a PHP azt gondolja, hogy be akarod fejezni a stringet), és a $firstname és $surname is behelyettesítődnek.

**Tömbök**

Ha más tapasztalatunk is van a programozással vagy az adatstruktúrákkal kapcsolatban általánosan, akkor talán hallottunk a két leghétköznapibb struktúrával is: a listákkal és a térképekkel. A lista egy rendezett elemek halmaza, míg a térkép egy kulcsokkal meghatározott elemek halmaza. Nézzünk egy példát:

List: ["Harry", "Ron", "Hermione"]

Map: {

"name": "James Potter",

"status": "dead"

}

Az első elem egy névlista, ami három értéket tartalmaz: Harry, Ron, és Hermione. A második egy térkép, ami két értéket definiál: James Potter és dead. Mindegyik egy kulccsal van meghatározva: name és status.

PHP-ban nincsenek külön listák és térképek, csak tömbök. A tömbök pedig olyan adatstruktúrák, amik mindkettőt teljesítik.

**Tömbök inicializálása**

Többféleképpen lehet egy tömböt inicializálni. Lehet egy üres tömböt vagy egy adattömböt inicializálni. Az adatokat is különbözőképpen lehet egy tömbbe írni. Nézzünk néhány példát:

<?php

$empty1 = [];

$empty2 = array();

$names1 = ['Harry', 'Ron', 'Hermione'];

$names2 = array('Harry', 'Ron', 'Hermione');

$status1 = [

'name' => 'James Potter',

'status' => 'dead'

];

$status2 = array(

'name' => 'James Potter',

'status' => 'dead'

);

A fenti példában egy listát és egy térképet határoztunk meg az előző részből. $names1 és $names2 ugyanazok a tömbök, csak külőnböző jelölésekkel. Ugyanez történik a $status1 és $status2-el. Az $empty1 és $empty2 pedig egy üres tömböt írnak le.

Később meglátjuk, hogy a listák úgy lesznek kezelve, mint a térképek. Belülről, a $names1 tömb, és a kulcsai sorba rendezett számok. Ebben az esetben egy másik féle inicializálása ennek a tömbnek lehet:

$names1 = [

0 => 'Harry',

1 => 'Ron',

2 => 'Hermione'

];

Egy tömb kulcsai alfanumerikus értékek lehetnek, mint a stringek vagy a számok. Egy tömb értékei pedig akármik lehetnek: stringek, számok, booleanok, más tömbök, stb. Például ilyen:

<?php

$books = [

'1984' => [

'author' => 'George Orwell',

'finished' => true,

'rate' => 9.5

],

'Romeo and Juliet' => [

'author' => 'William Shakespeare',

'finished' => false

]

];

Ez a tömb egy lista, ami két másik listát-térképet tartalmaz. Mindkét térkép különböző értékeket tartalmaz.

**Tömbök feltöltése**

A tömbök nem megváltoztathatatlanok, akár már inicializáláskor meg lehet őket változtatni. Már azzal is meg lehet változtatni tartalmukat, hogy ha lista vagy térképként kezeljük őket. Térképként kezelni azt jelenti, hogy megadunk egy kulcsot, amit majd felül lehet írni. Listaként kezelni azt jelenti, hogy hozzá fűzni egy elemet a tömb végére.

<?php

$names = ['Harry', 'Ron', 'Hermione'];

$status = [

'name' => 'James Potter',

'status' => 'dead'

];

**$names[] = 'Neville';**

**$status['age'] = 32;**

print\_r($names, $status);

Az első kijelölt sorban a nevet hozzáfűzi a nevek listához. A másodikban egy új kulcs-érték párt ad hozzá. Az eredményt meg lehet tekinteni a böngészőben, használva a print\_r -t. Hasonló dolgot csinál, mint a var\_dump, csak minden érték típusa és mérete nélkül.

**print\_r és var\_dump a böngészőben**

Mikor kiíratunk valamit a böngészőbe jó, hogy lássunk soronként egy kulcs-érték párt, de ha megnézzük, akkor az egész tömböt egy sorba fogja kiíratni. Ez azért történik, mert a böngésző HTML-ként akartja megjeleníteni a kódot, és figyelmen kívül hagyja az új sorokat és az üres helyeket. Ha a ezt PHP-ként akarjuk megtekinteni, akkor az oldal forráskódját kell megnézni -ezt megtehetjük, úgy hogy a jobb klikket katintunk az oldalon.

Hogy kitöröljünk egy elemet a tömbből, ahelyett, hogy hozzáadnánk vagy frissítenénk a tömböt, használjuk a unset funkciót:

<?php

$status = [

'name' => 'James Potter',

'status' => 'dead'

];

**unset($status['status']);**

print\_r ($status);

Az új $status tömb csak a kulcs nevét tartalmazza.

**Tömbökhöz való hozzáférés**

A tömbökhöz való hozzáférés egyszerű, mint mikor frissítjük őket. Ehhez szükséges a listák működésének az ismerete. Mint tudjuk, a listák úgy vannak belülről kezelve, mint a térképek különböző sorbarendezett numerikus kulcsokkal. Az első kulcs mindig 0; szóval egy n elemű listában az értékek mindig 0-tól n-ig lesznek:

Egy adott tömbhöz akármilyen kulcsot hozzá leet adni, még akkor is, ha az előtte numerikus érték volt. A probléma ott kezdődik, mikor egy numerikus kulcsot megadva később, megpróbálunk hozzáfűzni egy elemet a tömbhöz. Vajon mi történik?

<?php

$names = ['Harry', 'Ron', 'Hermione'];

$names['badguy'] = 'Voldemort';

$names[8] = 'Snape';

$names[] = 'McGonagall';

print\_r($names);

Az utolsó kódrészlet eredménye a következő:

Array

(

[0] => Harry

[1] => Ron

[2] => Hermione

[badguy] => Voldemort

[8] => Snape

[9] => McGonagall

)

Mikor egy értéket próbálunk hozzáfűzni, a PHP az utolsó numerikus kulcs után fogja azt betenni, ami esetünkben a 8.

Talán már észrevettük, hogy egy tömb akármelyik részét ki tudjuk iratni úgy, hogy megadjuk annak a kulcsát:

<?php

$names = ['Harry', 'Ron', 'Hermione'];

**print\_r($names[1]); // prints 'Ron'**

Ha egy olyan kulcshoz próbálunk hozzáférni, ami nem létezik, akkor az null-t ad vissza és egy megjegyzést, mert a PHP úgy érzékeli, hogy valamit hiba van a kódban.

<?php

$names = ['Harry', 'Ron', 'Hermione'];

var\_dump($names[4]); // null and a PHP notice

**Az üres és az isset függvények**

Két hasznos függvény van ahhoz, hogy egy tömb tartalmát megvizsgáljuk. Ha meg akarjuk kérdezni, hogy ha egyáltalán tartalmaz e valamit a tömb, akkor használhatjuk az empty függvényt. Ez műkődik stringekkel is, meg tudja mondani ha üres vagy sem (’’). Az isset függvény megnézi egy tömb helyzetét és igazat vagy hamisat ad vissza eredményként:

<?php

$string = '';

$array = [];

$names = ['Harry', 'Ron', 'Hermione'];

var\_dump(empty($string)); // true

var\_dump(empty($array)); // true

var\_dump(empty($names)); // false

var\_dump(isset($names[2])); // true

var\_dump(isset($names[3])); // false

**Keresés egy tömbben**

Talán a leghasználtabb függvény a tömböknél az in\_array.Ez két értéket fogad be, az értéket amit keresel, és a tömböt. Ez a függvény igazat vagy hamisat ad vissza eredményként. Ez nagyon hasznos, mert legtöbb esetben úgy dolgozunk egy tömbbel, hogy megnézzük ha egy adott elem benne van e.

Ennél talán hasznosabb az array\_search. Ez ugyanúgy működik, annyi különbséggel, hogy boolean helyett az elem kulcsát adja meg, hogy az hol található. Nézzük meg mindkét függvényt:

<?php

$names = ['Harry', 'Ron', 'Hermione'];

$containsHermione = in\_array('Hermione', $names);

var\_dump($containsHermione); // true

$containsSnape = in\_array('Snape', $names);

var\_dump($containsSnape); // false

$wheresRon = array\_search('Ron', $names);

var\_dump($wheresRon); // 1

$wheresVoldemort = array\_search('Voldemort', $names);

var\_dump($wheresVoldemort); // false

Tömbök rendezése

Egy tömb többféle képpen is elrendezhető, így nagy az esélye annak, hogy az a sorren, ami nekünk kell, az különbözik az eredetitől. Alapértelmezetten a tömb olyan sorrendbe van rakva, amilyen sorrendbe az elemek hozzá lettek adva. Tömbrendezésnél lehet választani, hogy vagy a kulcsot megadva rendezzük azt, vagy pedig új listába rakjuk őket, új kulcsokkal.

Itt van egy teljes lista az összes ilyen függvénnyel http://php.net/manual/en/array.sorting.php,de itt le vannak írva a legfontosabbak:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Sorts by** | **Maintains key association** | **Order of sort** |
| sort | Value | No | Low to high |
| rsort | Value | No | High to low |
| asort | Value | Yes | Low to high |
| arsort | Value | Yes | High to low |
| ksort | Key | Yes | Low to high |
| krsort | Key | Yes | High to low |

Ezek a függvények mindig csak egy paramétert fogadnak el, a tömböt, és nem térítenek vissza semmit. E helyett, ők rögtön elrendezik a tömböt, amit paraméterként adtunk meg. Itt van egy pár példa:

<?php

$properties = [

'firstname' => 'Tom',

'surname' => 'Riddle',

'house' => 'Slytherin'

];

$properties1 = $properties2 = $properties3 = $properties;

sort($properties1);

var\_dump($properties1);

asort($properties3);

var\_dump($properties3);

ksort($properties2);

var\_dump($properties2);

Először inicializálunk egy tömböt néhány kulcs-érték párral, majd hozzárendeljük a $properties-hez. Majd létrehozunk három változót, amik az eredeti tömb másolatai. Miért csináljuk ezt? Mert ha az eredeti tömböt rendezzük el, akkor az már nem lesz ugyanaz, mint volt. Ebben a példában, csak azt szemléltetjük, hogy mi történik a tömbbel, ha különböző képpen rendezzük azt el. Végül ezeket a külőnböző elrendezéseket kiíratjuk. A böngésző valami ilyesmit kell mutasson:

array(3) {

[0]=>

string(6) "Riddle"

[1]=>

string(9) "Slytherin"

[2]=>

string(3) "Tom"

}

array(3) {

["surname"]=>

string(6) "Riddle"

["house"]=>

string(9) "Slytherin"

["firstname"]=>

string(3) "Tom"

}

array(3) {

["firstname"]=>

string(3) "Tom"

["house"]=>

string(9) "Slytherin"

["surname"]=>

string(6) "Riddle"

}

Az első függvény, a sort, abc sorrendbe rendezi az értékeket. És ha megnézzük a kulcsokat, akkor lássuk, hogy a kulcsok meg vannak számozva, az eredeti kulcsokkal ellentétben. Az asort ugyanúgy rendezi az értékeket, csak meghagyva az eredeti kulcs-érték párokat. A ksort a kulcsok szerint abc sorrendbe rendezi az értékeket.

**Hogy emlékezzünk ilyen sok függvény névre**

A PHP-nak sok előre megírt függvénye van, amik segítenek nekünk, anélkül, hogy megtanulnánk mindegyiket kívűlről. A függvény segítők egy külön dokumentumba jelennek meg. Persze könyebb úgy programozni, hogy nem váltunk egyik dokumentumról át a másikra. Szóval itt van néhány tipp, hogy könnyebben megjegyezzük őket:

* Egy a a névben asszociatív jellegű, így megőrzi a kulcs-érték asszociációt.
* Egy r a névben fordítottat jelent, tehát a sorrend megváltozik
* A k kulcsot jelent, szóval a kulcs alapján fog történni a rendezés

**Egyéb tömbös függvények**

Több, mint 80 különböző függvény van, ami tömbökkel kapcsolatos. Így olyanos is lehetnek, amiről més sosem hallottunk, mivel nagyon konkrét céljaik vannak. A teljes lista itt megtalálható: http://php.net/manual/en/book.array.php.

Egy listát kaphatunk a kulcsokról az array\_keys , és egy listát az értékekről a array\_values függvénnyel:

<?php

$properties = [

'firstname' => 'Tom',

'surname' => 'Riddle',

'house' => 'Slytherin'

];

**$keys = array\_keys($properties);**

var\_dump($keys);

**$values = array\_values($properties);**

var\_dump($values);

Az elemek számát kaphatjuk vissza a count függvénnyel:

<?php

$names = ['Harry', 'Ron', 'Hermione'];

**$size = count($names);**

var\_dump($size); // 3

És az array\_merge függvénnyel két tömböt fűzhetünk össze egy tömbbe:

<?php

$good = ['Harry', 'Ron', 'Hermione'];

$bad = ['Dudley', 'Vernon', 'Petunia'];

**$all = array\_merge($good, $bad);**

var\_dump($all);

Ez az utolsó példa ezt fogja kiírni:

array(6) {

[0]=>

string(5) "Harry"

[1]=>

string(3) "Ron"

[2]=>

string(8) "Hermione"

[3]=>

string(6) "Dudley"

[4]=>

string(6) "Vernon"

[5]=>

string(7) "Petunia"

}

Mint láthatjuk, a második tömb kulcsai már mások lettek, mert a két eredeti tömb kulcsai ugyanazok voltak, és egy tömb nem tartalmazhat két értéket egy kulcshoz.