Irányított strukturák

és függvények

Lazsádi Laura, Gazdasági informatika

2022

# **Irányított strukturák**

Eddig a fájlokat mindig soronként hajtottuk végre. Ezért kaptunk értesítéseket egyes

forgatókönyvekről, például amikor a tömb nem tartalmazza azt, amit keresünk. Nem lenne jobb, ha kitudnánk választani, hogy melyik sor hajtodjon végre? Az irányított strukturák ebben segítenek nekünk.

Az irányított strukturák olyanok, mint a forgalomelterelő táblák. Néhány előre

meghatározott feltételtől függ a végrehajtási folyamat. Külömböző irányított srukturák léteznek, de leginkább ezeket a következő kategoriákba soroljuk: feltételes és ciklusos. A feltételes strukura lehetővé teszi számunkra, hogy eldöntsük végrehajtsuk-e az utasítást vagy nem. A ciklusos struktura annyiszor hajt végre egy utasítást, ahányszor csak szükséges.

# **Feltételes**

A feltételes struktura logikai kifejezést értékel ki, amely visszatérít egy értéket. Ha a

kifejezés igaz, akkor minden műveletet végrehajt ami abban a kódrészben található. A kód részeket csoportosíthatjuk, ezeket {} zárojel közé tesszük. Lássuk, hogyan működik:

<?php

echo ”Before the conditional.”;

if (4 > 3) {

echo ”Inside the conditional.”;

}

if (3 > 4) {

echo ”This will not be printed.”;

}

echo ”After the conditional.”;

Az előző kódrészletben 2 feltételt használtunk. A feltétel az **if** kulcsszóval van definiálva,

ezt követi a feltétel zárojelbe téve, majd a a kódrészlet. Ha a kifejezés igaz végrehajtja a kódrészt, különben kihagyja.

Növelheti a feltételek erejét, ha az **else** kulcszót használja. Ez arra utasítja a PHP-t, hogy

hajtson végre valamilyen kódrészletet ha a feltétel nem teljesül. Nézzünk is egy példát:

if ( 2 > 3 ) {

echo “Inside the conditional.”;

} else {

Echo “Inside the else.”;

}

Az előző példa az **else**-ben levő kódrészt hajtja végre, mivel az **if** nem teljesült.

Végül lehet használni az **elseif** kulcsszót, amit egy újabb feltétel és kódrész követ, hogy

további feltételeket kérjen a PHP-től. Annyi **elseif**-et használhat az **if** után amennyire szükség van. Ha egy **else**-t hozzáadunk, akkor ez kell legyen a feltétel utolsó láncszeme. Ne felejtsük el azt sem, hogyha a PHP talál egy olyan feltételt ami igaz, akkor a többi feltételt nem veszi figyelemben.

<?php

if ( 4> 5 ) {

echo “Not printed”;

}elseif ( 4 > 4){

echo “Not printed”;

}**elseif ( 4 == 4) {**

**Echo “Printed.”;**

}elseif (4 > 2 ){

echo ”Not evaluated.”;

}else {

echo “ Not evaluated.”;

}

if ( 4 == 4){

echo “Printed.”;

}

Az utolsó példában a kiemelt feltétel bizonyult igaznak. Ezt követően a PHP nem értékel

ki több feltételt, amíg egy új **if** el nem indul.

Ezeket tudva próbáljuk meg elrendezni az alkalmazásunkat, hogy az utasítások csak

szükség esetén hajtodjanak végre. A következő kódrészt másoljuk be az index.php fájlba.

<?!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<title> Bookstores</title>

</head>

<body>

<p>

**<?php**

**if ( isset ($\_COOKIE[‘usernam’])) {**

**echo “You are ” . $\_COOKIE[‘username’];**

**} else {**

**echo ”You are not authenticated.”;**

**}**

**?>**

</p>

**<?php**

**if ( isset ($\_GET[‘title’]) && isset ($\_GET[‘author’])) {**

**?>**

**<p> The book you are looking for is </p>**

**<ul>**

**<li><b>Title</b>: <?php echo $\_GET[‘title’]; ?> </li>**

**<li><b>Author</b>: <?php echo $\_GET[‘author’]; ?> </li>**

**</ul>**

**<?php**

**} else {**

**?>**

**<p> You are not looking for a book? </p>**

**<?php**

**}**

**?>**

</body>

</html>

Ebben az új kódrészben két különböző módon kevertük össze a feltételeket és a HTML

kódot. Az első PHP tag megnyítása után használtunk egy **if…else**-t, amely kiirja az **echo** segítségével ha hitelesek vagyunk vagy nem. A HTML nincs összevonva a feltételekkel, ez teljesen egyértemű.

A második opció - a második kiemelt kódrész – egy csúnyább megoldást mutat, de néha

ez is szükséges. Ha sok HTML kódot szeretne kiíratni, akkor az **echo** nem a legpraktikusabb dolog, jobb ha bezárja a PHP tag-et, kiíratja a szükséges HTML-t majd ismét megnyítja a PHP tag-et. Ezt akkár egy **if**-en belül is megteheti, ahogy a kódban is látható.

**PHP és HTML keverése**

Ha úgy érzi, hogy a legutóbbi szerkesztett fájl csúnyán néz ki, akkor igaza van.

A PHP és a HTML keverése eléggé zavaró és ezért kell ezt kerülni. A 6 fejezetben található MVC-hez való alkalmazkodás cimű tanulmányban meglátja, hogyan kell helyesen csinálni.

Szerkesszük az **authenticate.php** fájlt is, mivel ez a fájl megpróbál hozzáférni azokhoz a

$\_POST bejegyzésekhez, amelyek esetleg nincsenek ott. A fájl új tartalma a következő:

<?php

$submitted = isset ($\_POST['username']) && isset($\_POST['password']);

if ($submitted) {

setcookie('username', $\_POST['username']);

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Bookstore</title>

</head>

<body>

**<?php if ($submitted): ?>**

<p>Your login info is</p>

<ul>

<li><b>username</b>: <?php echo $\_POST['username']; ?></li>

<li><b>password</b>: <?php echo $\_POST['password']; ?></li>

</ul>

**<?php else: ?>**

<p>You did not submit anything.</p>

**<?php endif; ?>**

</body>

</html>

Ez a kódrész tartalmazza a feltételes feltételeket, amiket már ismerünk. Beállítunk egy

változót, hogy majd tudjuk ellenőrizni, ha elküldte-e a bejelenkezést vagy nem, ha elküldte akkor állítsa be a sütiket. A kiemelt sorok a HTML-be való feltételes feltételek beépítésének új módját mutatják be. Ha HTML-el dolgozik ez a módszer olvashatóbbá teszi a kódot, kerülni kell a {} használatát, helyette használja a **:** és az **endif**-et. Mindkét szintaxis helyes, de mindig azt kell használni amelyik jobban olvasható.

# **Switch...case**

A **switch...case** egy másik irányított struktura, hasonló az **if...else**-hez. Ez a struktura

csak egy kifejezést értékel ki és ennek értékétől függöen hajtja végre a kódrészt. Lássunk egy példát.

<?php

switch ( $title ) {

case ’Harry Potter’ :

echo ”Nice story, a bit to long.” ;

break;

case ’Lord of the Rings’ ;

echo ”A classic!”;

break;

default:

echo ”Dunno that one.”;

break;

}

A **switch** egy kifejezést vesz fel, ebben az esetben egy vátozót, majd meghatározza az

eseteket. Ha az eset(case) megegyezik a kifejezés aktuális értékével, akkor a PHP végrehajtja a benne levő kódot. Amint a PHP talál egy **break** utasítást kilép a **switch...case-**ből. Ha egyik eset sem felel meg a kifejezésnek a PHP végrehajtja a **default** parancsot, ha van ilyen mivel ez nem kötelező.

Tudnunk kell, hogy a **break** kötelező, ha ki szeretnénk lépni a a **switch...case**

utasításból. Ha nem adjuk meg, akkor a PHP továbbra is végrehajtja az utasításokat, még akkor is ha új case-el találkozik. Lássunk egy hasonló példát, de break használat nelkül.

<?php

$title = 'Twilight';

switch ($title) {

case 'Harry Potter':

echo "Nice story, a bit too long.";

case 'Twilight':

echo 'Uh...';

case 'Lord of the Rings':

echo "A classic!";

default:

echo "Dunno that one.";

}

Ha a kódot teszteljük a böngészőben, akkor látni fogjuk, hogy a kiírt szöveg a következő:

**Uh…A classic! Dunno that one.** A PHP megállapította, hogy a második eset érvényes, ezért végrehajtotta az ott található kódrészt. De mivel nem használtunk break-et végig fut. Néha lehet ez a kívánt hatás, de azért legyünk óvatosak amikor használjuk.

# **Ciklusok**

A ciklusok olyan irányított strukturák amelyek lehetővé teszik bizonyos utasítások

többszöri végrehajtását, ahányszor csak szükséges. Többféle módon használhatjuk őket, de a leggyakoribb a tömbök bejárása. Például képzelje el, hogy van egy tömbje elemekkel, de nem tudja, hogy mi van benne. Mindegyik elemet kiszeretné íratni, ezért mindegyiken végig kell menie.

Négyféle ciklus létezik. Mindegyiknek megvannak a saját használati esetei, de általában

az egyik típusú ciklust átlehet alakítani másik tjpusúvá. Nézük meg őket alaposan.

# **While**

A while ciklus a legegyszerűbb a ciklusok közül. Egy kódrészt addig hajt végre, amíg az

értékelendő kifejzés hamis nem lesz. Lássunk egy példát.

<?php

$i = 1;

while ($i < 4) {

echo $i . " ";

$i++;

}

Az előző példában definiáltunk egy változót melynek értéke 1. Ezután van egy **while**

ciklusunk, amelyben a feltétel **$i < 4**. A ciklus addig hajtja végre a kódrészt, amíg a feltétel hamis nem lesz. Mint látható a cikluson belül minden egyes alkalommal 1-gyel nőveljük az **$i** változó értékét, ezért a ciklus 4 alkalom után véget ér. Ellenőrizze a kód kimenetét, “0 1 2 3” üzenetet fogja látni. Az utolsó kiírt érték a 3, tehát akkor a **$i** értéke 3 volt. Eztután az értéket növeltük 4-re, így amikor a **while** feltételhez értünk az eredmény hamis.

**While és végtelen ciklusok**

A while ciklus leggyakoribb problémája, hogy végtelen ciklus keletkezik. Ha nem ad hozzá kódot a **while** ciklus belsejébe, hogy frissítse a változókat, akkor a PHP sosem fog kilépni a ciklusból

# **Do…while**

A **do…while** nagyon hasonlít a **while** ciklusra, abban az értelemben, hogy minden

alkalomkor egy feltételnek felel meg, és addig hajtja végre a kódrészt, amíg a kifejezés hamis lesz. Az egyetlen különbség az, hogy a while ciklus a feltételre rákérdez a kódrész végrehajtása előtt, így néha előfordulhat, hogy nem is lépünk be a ciklusba, ha a kifejezés hamis már az első alkalomkor. A do…while a feltételre csak a kódrész végrehajtása után kérdez rá, tehát ha a kifejezés hamis már az elején, akkor is a ciklus legalább egyszer lefut.

<?php

echo "with while: ";

$i = 1;

while ($i < 0) {

echo $i . " ";

$i++;

}

echo "with do-while: ";

$i = 1;

**do {**

**echo $i . " ";**

**$i++;**

**} while ($i < 0);**

Az előző kódrészben két ciklust határoztunk meg ugyanazzal a feltételel és kódrésszel, de

ha végrehajtja őket látni fogja, hogy csak a **do…while** cikluson belüli kód fog végrehajtodni. A feltételek már az elejétől fogva hamisak, így a **while** ciklusba nem is lép be, míg a **do…while**-ba egyszer belép.

# **For**

A **for** ciklus a legösszetettebb a 4 ciklus közül. Meghatározz egy kezdeti értéket, egy

kilépési feltételt és a végén egy léptetési feltételt. Amikor a PHP először találkozik a ciklussal, végrehajtja az inicializálási feltételt. Majd kiértékeli a kilépési feltételt, és ha ez **igaz** akkor belép a ciklusba. Miután mindent végrahjtott a cikluson belül, végrehajtha a léptetési/növelési kifejezést is. Miután ezek megtörténtek megismétli, amíg a hamis értéket nem kapja. Most is egy példa fogja ezt bemutatni:

<?php

for ($i = 1; $i < 10; $i++) {

echo $i . " ";

}

Az inicializáló kifejezés a **$i = 1,** és ez csak első alkalomkor kerül végrehajtásra. A

kilépési feltétel **$i < 10,** ez minden alkalomkor végrehajtódik. A **$i++** is minden egyes alkalom végén végrehajtódik. Ez a példa 1-től 9-ig kiírja a számokat. A for ciklus másik leggyakoribb használata a tömöknél történik.

<?php

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

for ($i = 0; $i < count($names); $i++) {

echo $names[$i] . " ";

}

Ebben a példában egy neveket tartalmazó tömböt láthatunk. Mivel úgy van definiálva

mint egy lista, kulcsai 0, 1 és 2. A ciklus az **$i** változónak kezdetben a 0 kapja, és ez az érték nm lehet nagyobb mint a tömb elemeinek száma, azaz 3. Az első iterációban a **$i** 0, a másodikban 1, a harmadikban pedig 2. Ha a **$i** 3, akkor nemfog belépni a ciklusba, mivel a kilépési feltétel hamis.

Minden végrehajtás után kiírjuk a tömb **$i** poziciójának tartalmát, ezért a kód eredmánye

a tömben szereplő mindhárom név.

**Legyen óvatós a kilépési feltételekkel**

Nagyon gyakori, hogy olyan kilépési feltételeket állítunk be, amire nem igazán van szükségünk, különösen tömbök esetén. Ne feledje, hogy a tömbök 0-tól kezdődnek, ha listák, tehát egy három elemből álló tömb idexelése 0, 1 és 2 lesznek. Ha a kilépési feltételt **$i <= count($array)**-ként határozzuk meg, az hibát fog okozni, mivel ha **$i** 3, akkor az is teljesíti a kilépési feltételt, és megpróbálja elérni a 3-as értéket, amely nem létezik.

# **Foreach**

Az utolsó ciklus típus a **foreach**. Ez a ciklus kizárólag a tömbökre vonatkozik, és

lehetővé teszi egy tömb teljes bejárását, még akkor is, ha nem ismeri a kulcsait. A szintaxisnak két lehetősége van, amint a következő példában látható:

<?php

$name = { ‘Harry’, ‘Ron’, ‘Hermione’} ;

foreach ($name as $name) {

echo $name . “ ”;

}

foreach ($name as $key => $name) {

echo $key . “ -> “ . $name . “ ”;

}

A foreach ciklus elfogat egy tömböt – ebben az esetben a $name – és megad egy

változót, amely a tömb bejegyzésének értékét fogja tartalmazni. Láthatjuk, hogy nem kell kimeneti feltételt megadnunk, mivel a PHP tudni fogja, ha a tömb iterációra került. Opcionálisan megadhat egy változót, amely tartalmazza az egyes iterációk kulcsát, mint a második ciklusban.

A **foreach** ciklusok a térképeknél is hasznosak, ahol a kulcsok nem feltétlenül

numerikusak. A PHP tömb indexelési sorrendje megegyezik a tömb tartalmának beszúrásának sorrendjével.

Használjunk néhány ciklust a mi alkalmazásunkban is. Az elérhető könyveket szeretnénk

Megjeleníteni a kezdőoldalon. A könyvek listája egy tömbben található, így mindegyiket bekell járjuk a foreach ciklussal, mindegyikből kikell írni néhány információt. Illessze a következő kódot a body tag-be az index.php fájlba.

<?php endif;

$books = [

[

'title' => 'To Kill A Mockingbird',

'author' => 'Harper Lee',

'available' => true,

'pages' => 336,

'isbn' => 9780061120084

],

[

'title' => '1984',

'author' => 'George Orwell',

'available' => true,

'pages' => 267,

'isbn' => 9780547249643

],

[

'title' => 'One Hundred Years Of Solitude',

'author' => 'Gabriel Garcia Marquez',

'available' => false,

'pages' => 457,

'isbn' => 9785267006323

],

];

?>

<ul>

**<?php foreach ($books as $book): ?>**

**<li>**

**<i><?php echo** **$book['title']; ?></i>**

**- <?php echo $book['author']; ?>**

**<?php if (!$book['available']): ?>**

**<b>Not available</b>**

**<?php endif; ?>**

**</li>**

**<?php endforeach; ?>**

**</ul>**

A kiemelt kód egy **foreach** ciklust mutat a **:** jelölés használatával, ami jobb, ha

HTML-lel keverjük a PHP-t. Indexeli az összes **$book** tömböt, és minden könyvhöz kiír bizonyos információt a HTML listába. Figyeljük meg azt is, hogy van egy feltételes ciklusunk, ami teljesen rendben van. Természetesen ez a feltételes feltétel a tömb minden egyes bejegyzésére lefut, ezért érdemes a ciklusok kódrészét a lehető legegyszerűbben megírni.

# **Függvények**

A függvény egy újrafelhasználható kódrész, amely adott bemenet esetén végrehajt

bizonyos műveleteket, és valamilyen eredményt ad vissza. Számos előre definiált függvényt már ismer, például: empty, in\_array vagy var\_dump. Ezeket a függvényeket a PHP tartalmazza, így nem kell újra feltalálnia és megírni, de nagyon egyszerűen sajátot is létrehozhatunk.

Függvényeket akkor használhat, amikor egy adott kódot többszőr használ, vagy bizonyos funkciók betöltésére.

# **Függvények deklarálása**

A függvény deklarálása azt jeleni, hogy megkell írjuk a függvényt előre, hogy később

tudjuk majd használni. A függvénynek van neve, argumentumai, és van egy kódrésze. Opcionálisan meghatározhatja a vissztatéritési értéket is. A függvény nevének ugyanazokat a szabályokat kell követnie, mint a változóneveknél, azaz betűvel vagy aláhúzással kell kezdődnie, és betűket, számokat vagy aláhúzást tartalmazhat. Nem lehett foglalt szó. Lássunk egy egyszerű példát.

function addNumbers($a, $b) {

$sum = $a + $b;

return $sum;

}

**$result = addNumbers(2, 3);**

Az előző függvény neve **addNumbers**, és két argumentumra van szüksége **$a** és **$b**. A kódrész definiál egy új **$sum** változót, amely mindkét argumentum összege, majd a tartalmát **return**-nal vissza adja. A függvény használátához, csak a nevét hasznáva és a kívánt paraméterekkel kell meghívni, ahogy a kiemelt sorban is látható.

A PHP nem támogatja a túlterhelt függvényeket. A túlterhelés azt jelenti, hogy két vagy

több függvény ugyanazon névvel szerepel de mások a paraméterek. Amit látja a paramétereket análkül deklarálhatja, hogy tudná annak típusát, így a PHP nem tudja eldönteni, hogy melyik függvényt használja.

Egy másik fontos dolog amit megkell jegyezni az a változók hatóköre. A **$sum** változót a

függvényen belül deklaráljuk, ezáltal a függvény lejárta után a változó többet nem lesz elerhető. Ez azt jelenti, hogy a függvényen belül deklarált változok csak a függveny belsejében használhatóak. Továbbá, ha a függvényen kivül deklarálum egy **$sum** változót azt nem érhetjuk el a függvényen belül, csak akkor ha megdajuk a függvenynek mint paraméter.

# **Függvények paraméterei**

A függvények a paramétereken keresztül kapnak infomációt kivülről. Bármennyi

paramétert megadhat egy függvénynek – beleértve 0 (egyet sem). A függvény paramétereinek névre van szükségük, hogy tudjuk őket használni a függvényen belül; két paraméternek nem lehet azonos a neve. A függvény meghívásakor aparamétereket a deklarált sorrendben kell elküldeni.

A függvények tartalmazhatnak opcionlális paramétereket, ezeknek nem kell értéket

megadni. A függvény deklarálásakor meg kell adni egy alapételmezett értéket a paramétereknek. Tehát, ha a felhasználó nem ad meg értéket, akkor a függvény az alapértelmezett értéket fogja használni.

**function addNumbers($a, $b, $printResult = false) {**

$sum = $a + $b;

if ($printResult) {

echo 'The result is ' . $sum;

}

return $sum;

}

$sum1 = addNumbers(1, 2);

$sum1 = addNumbers(3, 4, false);

$sum1 = addNumbers(5, 6, true); // it will print the result

Ebben a példában láthatjuk, hogy az utolsó függvényhíváskor két kötelező és egy

opciónális paramétert használunk. Az opcionális paraméter alapértelmezett értéke **hamis**, és ezt követően a függvényen belül normálisan használjuk. A függvény kiírja az összeg eredményét, ha a felhasználó harmadik paraméterként igazat ad meg, ami csak a függvény harmadik hívásánál történik meg. Az első két alkalomkor a **$printResult** értéke hamis.

A függvényben használt paraméterek azok csak a felhasználó által megadott paraméterek

másolatai. Ez azt jelenti, hogy ha a függvényen belül módosítjuk a paramétereket, azok nem lesznek hatással az eredeti értékekre. Ez a funkció arról ismert, hogy az értékek alapján küldi el a paramétereket. Lássunk egy példát:

function modify($a) {

$a = 3;

}

$a = 2;

modify($a);

var\_dump($a); // prints 2

Deklarálunk egy **$a** változót,melynek **2** értéket adunk, majd meghívjuk a **modify**

metódust, amely elküldi ezt az **$a**-t. A **modify** metódus módosítja az **$a** paramétert, értékét **3**-ra állítva, de ez nem befolyásolja az **$a** valtozó eredeti értékét, amely 2 marad, ahogy a **var\_dump**-ból is látható.

Ha ténylegesen módosítani szeretné a meghívásban használt eredeti változó értékét,

akkor a paramétert referenciával kell átadnia. Ehez a függvény deklarálásakor a parameter elé adjon hozza egy **&** jelet.

Function modify(&$a) {

$a = 3;

}

Most a **modify** függvény meghívásakor a **$a** értéke mindig **3** lesz.

**Paraméterek értékekkel és referenciaval**

A PHP lehetővé teszi ezt, sőt a PHP a nativ függvényeknél használ referenciás paramétereket. Emlékszel a tömbrendező függvényekre? Nem a rendezett tömböt adták vissza, hanem a tömböt rendezték. De a referenciával használt paraméterek a fejlesztők megzavarásának egy módja. Általában, amikor valaki egy függvényt használ, akkor egy eredményt vár, és nem azt akarja, hogy az általa megadott paraméterek módosuljanak. Tehát próbáld elkerülni ezt; az emberek hálásak lesznek!

# **A visszatérítési érték**

A függvényn belül tetszőleges számú **return** utasítás szerepelhet, de a PHP azonal kilép a

függvényből, hogy találkozik egy ilyennel. Tehát ez azt jelenti, hogy ha van két egymás követő **retunr** utaítas, a második sosem fog végrehajtodni. Ennek ellenére hasznos lehet több **return** utasítás használata, ha ezek feltételek belsejében talalhatóak. Adja hozzá ezt a függvényt a functions.php fájlhoz:

function loginMessage() {

if (isset($\_COOKIE['username'])) {

return "You are " . $\_COOKIE['username'];

} else {

return "You are not authenticated.";

}

}

Használjuk az utolsó példát az index.php fájlban, kicserélve a kiemelt kódrészt(azt a

kódrészt amit nem változott azt //...-re cseréltem):

//...

<body>

**<p><?php echo loginMessage(); ?></p>**

<?php if (isset($\_GET['title']) && isset($\_GET['author'])): ?>

//...

Ezenkívül elhagyhatja a **return** utasítást, ha nem szeretné, hogy a függvény bármit is

vissza adjon. Ebben az esetben a függvény véget ér, ha eléri a kódrész végét.

# **Type hinting és visszatérítési érték típusok**

A PHP7 megjelenésével, a nyelv lehetővé teszi a fejlesztők számára, hogy megtudják

határozni mily függvények kapnak értéket és melyek térítenek vissza értéket. Megadhatja- mindig opcionális-, hogy milyen típusú paraméterre van szüksége a függvénynek ( **type hinting**) és a függvény által visszaadott eredmény típusát (**visszatérítési érték típus**) is. Lássunk előszőr egy példát:

<?php

**declare(strict\_types=1);**

**function addNumbers(int $a, int $b, bool $printSum): int {**

$sum = $a + $b;

if ($printSum) {

echo 'The sum is ' . $sum;

}

return $sum;

}

addNumbers(1, 2, true);

addNumbers(1, '2', true); // it fails when strict\_types is 1

addNumbers(1, 'something', true); // it always fails

A fenti függvényben látható, hogy a paraméterek egész és logikai értékűek kell legyenek

és az eredmény egész. Mostmár tudja, hogy a PHP rendelkezik típuszsonglőrrel, általában egy típus az értékét képes átalakítani egy másik típus megfelelő értékére, például a "2" karakterlánc használható egész 2-ként. Ahhoz, hogy a PHP ne használjon típuszsonglőrködést a függvények paramétereivel és eredményeivel, deklarálhatja a **strict\_types** direktívát az első kiemelt sorban látható módon. Ezt az utasítást minden olyan fájl tetején kell deklarálni, ahol ezt a viselkedést érvényesíteni kívánja.

A három meghívás a következőképpen történik:

* Az első hívás két egész számot és egy logikai értéket küld, amit a függvény vár, így a strict\_types értékétől függetlenül mindig működni fog.
* A második hívás egy egész számot, egy karakterláncot és egy logikai értéket küld. A karakterlánc érvényes egész értékkel rendelkezik, így ha a PHP-nek engedélyezték volna a típuszsonglőrködést, a hívás a szokásos módon oldódna meg. Ebben a példában azonban a fájl tetején lévő deklaráció miatt nem lehetséges.
* A harmadik hívás mindig sikertelen lesz, mivel a **" something "** karakterlánc nem alakítható át érvényes egész számmá.

Probáljunk meg használni egy függvényt a projektünkön belül. Az **index.php** fájlban van

egy foreach ciklus, amely bejárja a könyveket, és kinyomtatja azokat. A cikluson belüli kódot nehéz megérteni, mivel a HTML és a PHP keverve van és van egy feltétel is. Probáljuk meg összefoglani a logikát egy cikluson belül. Először hozzon létre egy új **functions.php** nevű fájlt a következő tartalomma:

<?php

function printableTitle(array $book): string {

$result = '<i>' . $book['title'] . '</i> - ' . $book['author'];

if (!$book['available']) {

$result .= ' <b>Not available</b>';

}

return $result;

}

Ez a fájl tartalmazz a majd a függvényeket. Az első a **printableTitle** , egy könyvet

tartalmazó tömböt kap paraméternek és egy karakterláncot épít fel, hogy a HTML-ben szépen nézzen ki. A kód ugyanaz, mint korábban, csak most egy függvénybe van beépítve.

Most az **index.php**-nek tartalmaznia kell a **functions.php** fájlt, majd a cikluson belüli

függvényt kell használnia. Lássuk hogyan:

<?php require\_once 'functions.php' ?>

<!DOCTYPE html>

<html lang="en">

//...

?>

<ul>

<?php foreach ($books as $book): ?>

**<li><?php echo printableTitle($book); ?> </li>**

<?php endforeach; ?>

</ul>

//...

Nos, mostmár a ciklus sokkal tisztábbnak tűnik, igaz? Illetve, ha a könyv címét máshol szeretnénk kiírni a kód többszöri megjrása helyett használjuk a függvényt.