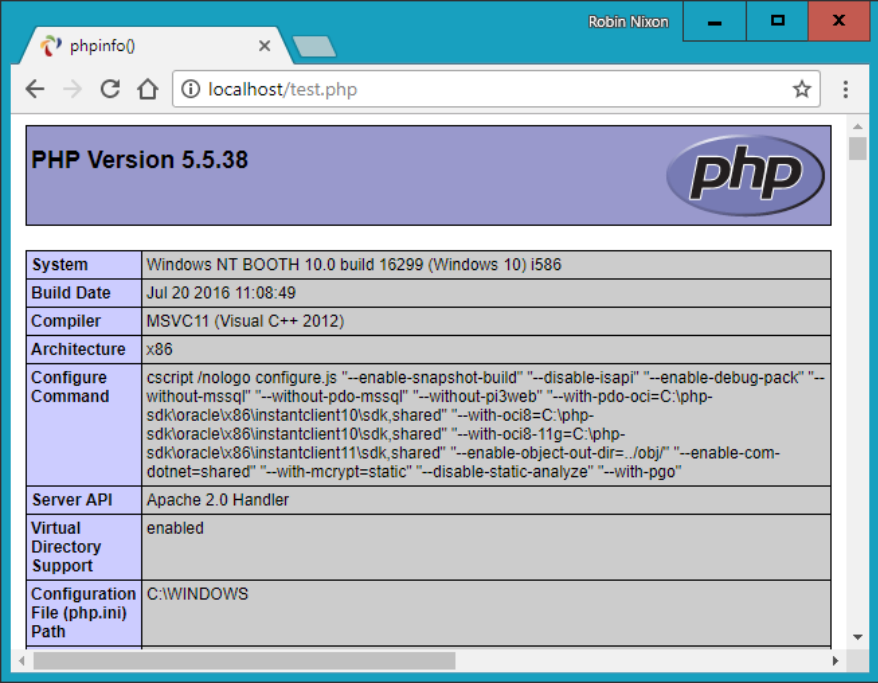
1. **PHP Funkciók**

Babos Zsombor-László, Gazdasági informatika

2021

A PHP több száz kész, beépített függvényt tartalmaz, így nagyon gazdag

nyelvvé téve azt. Hogy használjon egy funkciót, például láthatja a dátum funkciót itt működés közben:  
echo date (“l”); // Displays the day of the week  
A zárójelek jelzik a PHP-nek, hogy egy függvényre hivatkozol. Különben azt hiszi konstansra utalsz.  
A függvények tetszőleges számú argumentumot vehetnek fel, beleértve a nullát is. Például phpinfo, ahogy az alábbiakban látható, sok információt jelenít meg a PHP jelenlegi telepítéséről és nem igényel argumentumokat. Az ennek a funkciónak a meghívásának az eredménye az 5-1. ábrán látható:

  
*5-1. ábra. A PHP beépített phpinfo funkciójának a kiiratása*

A phpinfo funkció rendkívül hasznos információszerzéshez a jelenlegi PHP-telepítésről, de ez az információ a potenciális hackerek számára is nagyon hasznos lehet. Ezért soha ne hagyja meg ezt a funkciót bármely web-kész kódban.

Egyes beépített függvények, amelyek egy vagy több argumentumot használnak, megjelennek az 5-1. példában.

*5-1. példa. 3 string funkicó*

<?php  
 echo strrev(“ .dlrow olleH”); // Reverse string  
 echo str\_repeat(“Hip “, 2); // Repeat String  
 echo strtoupper(“hooray!”); // String to uppercase  
?>

Ez a példa három karakterlánc-függvényt használ a következő szöveg kimenetére:  
 **Hello world. Hip Hip HOORAY!**Mint ahogy látható a sttrev funció megfordította a karaktereket a string-ben, az str\_repeat megismételte a “Hip “ sting-et kétszer, és astrtoupper átkonertálta a “hooray!” nagybetűsre.

* 1. **Függvény meghatározása**

A függvény általános szintaxisa a következő:

function function\_name([parameter [, ...]])  
{  
 // Statements  
}

A szintaxis első sora a következőket jelzi:

• A definíció a függvény szóval kezdődik.

• Egy név következik, amelynek betűvel vagy aláhúzásjellel kell kezdődnie, majd betűk, számok vagy aláhúzásjelekel kell folytatódnia.

• A zárójelek megadása kötelező.

• Egy vagy több, vesszővel elválasztott paraméter nem kötelező.

A függvénynevek nem tesznek különbséget a kis- és nagybetűk között, így a következő karakterláncok mindegyike hivatkozhat a nyomtatási funkcióra : PRINT, Print és Print.  
A nyitó kapcsos zárójel elindítja azokat az utasításokat, amelyek végrehajtásra kerülnek, amikor meghívja a funkciót; hozzáillő merevítőnek kell zárnia. Ezek az állítások tartalmazhatnak egy vagy több return utasítást, amelyek arra kényszerítik a függvényt, hogy leállítsa a végrehajtást, és visszatérjen a hívókódhoz. Ha a return utasításhoz érték van csatolva, akkor a hívó kód visszaveheti, ahogy a továbbiakban látni fogjuk.

* 1. **Értékek visszaadása**

Vessünk egy pillantást egy egyszerű függvényre, amellyel egy személy teljes nevét kisbetűssé alakíthatja át majd az egyes névrészek első betűjét írjuk nagybetűvel. Láttunk már példát a PHP beépített strtoupper funkciójára az 5-1. példában. Jelenlegi funkciónkhoz az ellentéttjét, a strtolower-t használjuk:

$lowered = strtolower("aNY # of Letters and Punctuation you WANT"); echo $lowered;

A kimenet a következő:

**Any # of letters and punctuation you want**

Most megtehetjük a programtervezés első részét: hogy egy szót a nagy kezdőbetűvel kaphassunk, először a strtolower-t, majd az ucfirst-t hívjuk meg. A teendők módja ez a strtolower hívás beágyazása az ucfirstbe. Lássuk miért, mert fontos megérteni a kód kiértékelésének sorrendjét.

Tegyük fel, hogy egyszerűen meghívja a nyomtatási funkciót:

print((5-8));  
Először az 5-8 kifejezés kerül kiértékelésre, a kimenet pedig –3. (Ahogyan az előző fejezetben láthatta a PHP karakterlánccá alakítja az eredményt, hogy megjelenítse azt.) Ha a kifejezés tartalmaz függvényt, akkor az a függvény először kerül kiértékelésre:  
 print(abs(5-8));

A PHP számos dolgot csinál ennek a rövid utasításnak a végrehajtása során:

1. Kiértékeli 5-8-at és –3-at termel.

2. Használja az abs funkciót, hogy a –3-at 3-ra alakítsa.

3. Átalakítja az eredményt karakterlánccá, és kiadja a nyomtatási funkció segítségével.  
Mindez azért működik, mert a PHP minden elemet belülről kifelé értékel. Ugyanez az eljárás működik, amikor a következőket hívjuk:

ucfirst(strtolower("aNY # of Letters and Punctuation you WANT"))

A PHP átadja a karakterláncunkat a strtolower-nek, majd az ucfirst-nek, létrehozva (mint már láthattuk, amikor a funkciókkal külön játszottunk):

**Any # of letters and punctuation you want**

Most határozzunk meg egy függvényt (lásd 5-2. példa), amely három nevet vesz fel és mindegyiket kisbetűvé alakítja, nagy kezdőbetűvel.

*5-2. példa. Teljes név kitakarítása*

<?php  
 echo fix\_names(“WILLIAM”, “henry”, “gatES”);

function fix\_names($n1, $n2, $n3)  
 {

$n1 = ucifirst(strtolower($n1));  
 $n2 = ucifirst(strtolower($n2));

$n3 = ucifirst(strtolower($n3));

return $n1 . “ ” . $n2 . “ ” . $n3;  
 }  
?>

Könnyen előfordulhat, hogy ilyen típusú kódot ír, mert a felhasználók gyakran bekapcsolva hagyják a Caps Lock billentyűt, véletlenül rossz helyre írják be a nagybetűket, és még a nagybetűket is elfelejtik. A példa kimenete itt látható:  
 **William Henry Gates**

* 1. **Tömbb visszatérítése**

Most láttunk egy függvényt, amely egyetlen értéket térít vissza. Vannak módok arra is, hogy egy függvényből több értéket kapjunk. Az első módszer az, hogy visszaadja őket egy tömbön belül. Ahogy a 3. fejezetben láthatta, egy tömb olyan, mint egy csomó változó, amelyek sorban besűrítve A 5-3. példa bemutatja, hogyan lehet függvényértékek visszaadásához tömböt használni.

*5-3 példa. Több érték visszaadása egy tömbben*

<?php

$names = fix\_names("WILLIAM", "henry", "gatES");

echo $names[0] . " " . $names[1] . " " . $names[2];

function fix\_names($n1, $n2, $n3)

{

$n1 = ucfirst(strtolower($n1));

$n2 = ucfirst(strtolower($n2));

$n3 = ucfirst(strtolower($n3));

return array($n1, $n2, $n3);

}

?>

Ennek a módszernek az az előnye, hogy mindhárom nevet külön tartja, ahelyett, hogy egyetlen karakterláncba fűzné össze őket, így bármely felhasználóra hivatkozhat egyszerűen az első vagy az utolsó névvel anélkül, hogy bármelyik nevet ki kellene vonnia a visszaadott karakterláncból.

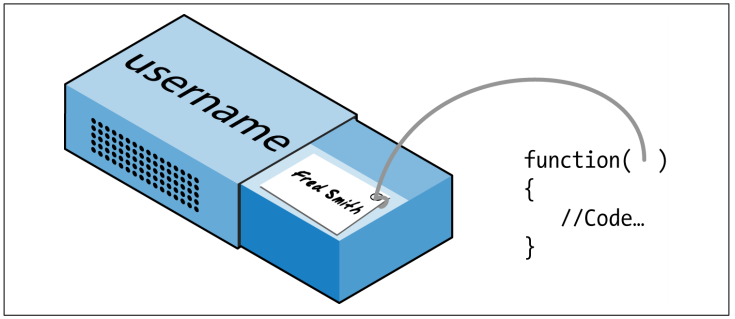
* 1. **Argumentumok átadása hivatkozással**

Az 5.3 előtti PHP-verziókban képes volt egy változót a & szimbólum egy függvény meghívásakor (például increment(&$myvar);) az elemző, hogy a változóra való hivatkozást adja át, nem pedig a változó értékét. Ez egy függvény hozzáférést biztosít a változóhoz (különböző értékek visszaírását lehetővé téve).

A hívásidő-hivatkozás elavult a PHP 5.3-ban és eltávolítva a PHP 5.4-ben. Ezért ne használja ezt a funkciót csak a régi webhelyeken, és még ott is ajánlott átírni a hivatkozással áthaladó kódot, mert a PHP újabb verzióinál végzetes hibával leáll.

Egy függvénydefiníción belül azonban továbbra is hivatkozással érheti el az argumentumokat. Ezt a koncepciót nehéz lehet felfogni, ezért térjünk vissza a 3. fejezet gyufásdoboz-metaforájához.

Képzelje el, hogy ahelyett, hogy kivenne egy papírt a gyufásdobozból, elolvassa, lemásolja mi van rajta egy másik papírra, visszahelyezve az eredetit, és átadva a függvénybe másolni, egyszerűen csatolhat egy cérnadarabot az eredeti darab papírhoz, és adja át az egyik végét a funkciónak (lásd 5-2. ábra).



*5-2. ábra. A hivatkozást egy változóhoz csatolt szálként képzeljük el*

Most a függvény követheti a szálat, hogy megtalálja a hozzáférni kívánt adatokat. Ezzel elkerülhető minden olyan többletköltség, amely a változó csak a függvény használatára való másolatának létrehozásához szükséges. Sőt, a függvény mostantól módosíthatja a változó értékét.  
Ez azt jelenti, hogy átírhatja az 5-3. példát, hogy átadja a hivatkozásokat az összes paraméterek, és akkor a függvény közvetlenül módosíthatja ezeket (lásd 5-4. példa).  
  
*5-4. példa. Értékek átadása egy függvénynek hivatkozással*

<?php

$a1 = "WILLIAM";

$a2 = "henry";

$a3 = "gatES";

echo $a1 . " " . $a2 . " " . $a3 . "<br>";

fix\_names($a1, $a2, $a3);

echo $a1 . " " . $a2 . " " . $a3;

function fix\_names(&$n1, &$n2, &$n3)

{

$n1 = ucfirst(strtolower($n1));

$n2 = ucfirst(strtolower($n2));

$n3 = ucfirst(strtolower($n3));

}

?>

Ahelyett, hogy a karakterláncokat közvetlenül a függvénynek adná át, először változókhoz rendelje őket és nyomtassa ki őket, hogy lássa az „előtt” értékeit. Ezután meghívja a függvényt, mint korábban, de a függvénydefiníción belül minden referenciaként átadandó paraméter elé egy & szimbólumot teszünk.

Most a $n1, $n2 és $n3 változók olyan „szálakhoz” kapcsolódnak, amelyek a $a1, $a2 és $a3 értékekhez vezetnek. Más szóval, van egy értékcsoport, de két változó név-készlet is hozzáférhet hozzájuk.  
Ezért a fix\_names függvénynek csak új értékeket kell hozzárendelnie a $n1, $n2 és $n3 elemekhez, hogy frissítse a $a1, $a2 és $a3 értékeket. A kód kimenete a következő:

**WILLIAM henry gatES   
William Henry Gates**

Amint látja, mindkét echo utasítás csak az $a1, $a2 és $a3 értékeket használja.

* 1. **Globális változók visszatérítése**

A legjobb módja annak, hogy egy függvény hozzáférést biztosítson egy külsőleg létrehozott változóhoz, ha úgy deklaráljuk, hogy globális hozzáféréssel rendelkezzen a függvényen belülről. A globális kulcsszó következett a változónévvel a kód minden részéhez teljes hozzáférést biztosít (lásd az 5-5. példát).

*5-5. példa. Visszatérő értékek a globális változókban*

<?php

$a1 = "WILLIAM";

$a2 = "henry";

$a3 = "gatES";

echo $a1 . " " . $a2 . " " . $a3 . "<br>";

fix\_names();

echo $a1 . " " . $a2 . " " . $a3;

function fix\_names()

{

global $a1; $a1 = ucfirst(strtolower($a1));

global $a2; $a2 = ucfirst(strtolower($a2));

global $a3; $a3 = ucfirst(strtolower($a3));

}

?>

Most már nem kell paramétereket átadnia a függvénynek, és nem kell elfogadnia őket. A deklarációt követően ezek a változók megtartják a globális hozzáférést, és elérhetők a programjának többi részén is, beleértve a funkcióit is.

* 1. **Összefoglaló a változó hatóköréről**

Egy gyors emlékeztető arról, amit a 3. fejezetből tudni kell:

• A helyi változók csak a kód azon részéből érhetők el, ahol Ön meghatározza őket. Ha egy függvényen kívül esnek, akkor minden kóddal elérhetők a függvényeken, osztályokon és így tovább. Ha egy változó egy függvényen belül van, akkor csak az a függvény hozzáférhet a változóhoz, és az értéke elveszik, amikor a függvény visszatér.

• A globális változók a kód minden részéből elérhetők.

• A statikus változók csak azon függvényen belül érhetők el, amelyik deklarálta őket, detöbb hívás során is megőrzik értéküket.

* 1. **Fájlok befoglalása és igénylése**

Ahogy fejlődik a PHP programozásának használatéban, valószínűleg elkezdi építeni a saját függvénykönyvtárát, amelyre úgy gondolja, hogy újra szüksége lesz. És valószínűleg elkezd más programozók által létrehozott könyvtárakat használni.

Nem szükséges ezeket a függvényeket a kódba másolni és beilleszteni. Elmentheti őket külön fájlokban, és parancsokkal húzza be őket. Két paranccsal hajthatja végre ezt a műveletet: az include és a require használatával.

* 1. **Az include állítás**

Az include használatával megmondhatja a PHP-nek, hogy kérjen le egy adott fájlt, és töltse be annak teljes tartalmát. Ez olyan mintha a beillesztett fájlt beillesztette volna az aktuális fájlba a beillesztési pontnál. Az 5-6. példa bemutatja, hogyan kell beilleszteni a *library.php* nevű fájlt.

*5-6. példa. Egy PHP fájl beillesztése*

<?php

include "library.php";

// Your code goes here

?>

1. **Include\_once használata**

Minden alkalommal, amikor kiadja az include direktívát, az újra tartalmazza a kért fájlt, még akkor is, ha már beillesztette. Tegyük fel például, hogy a *library.php* nagyon sok hasznos funkciót tartalmaz, ezért beleveszi a fájlba, de egy másik könyvtárat is tartalmazza a *library.php*-t. Az egymásba ágyazás révén véletlenül bevette a *library.php* fájlt kétszer. Ez hibaüzeneteket produkál, mert ugyanazt az állandót vagy függvényt többször próbálja definiálni. Tehát inkább az *include\_once*-t kell használnia (lásd

5-7. példa).

*5-7. példa. PHP fájl egyszeri beágyazása*

<?php

include\_once "library.php";

// Your code goes here

?>

Ezután minden további kísérlet ugyanannak a fájlnak a felvételére (az include vagy include\_once funkcióval) figyelmen kívül lesz hagyva. Annak megállapításához, hogy a kért fájl végrehajtásra került-e már, az abszolút fájl elérési útja megegyezik az összes relatív elérési út feloldása és a fájl elérése után megtalálható az include útvonalában.

  
Általánosságban elmondható, hogy a legjobb az include\_once mellett maradni és figyelmen kívül hagyni az alapvető include utasítás. Így soha nem lesz meg a a fájlok többszöri felvételének problémája.

* 1. **Require és require\_once használata**

Az include és include\_once lehetséges problémája az, hogy a PHP csak megkísérli

tartalmazni a kért fájlt. A program végrehajtása akkor is folytatódik, ha a fájl nem található.

Ha feltétlenül szükséges egy fájl hozzáadása, használj a require-t. Ugyanezen okokból én az include\_once használatát javaslom. Javaslom, hogy általánosságban ragaszkodjon ehhez a require\_once függvényhez amikor fájlt kell kérnie (lásd az 5-8. példát).

*5-8. példa. PHP fájl egyszeri igénylése*

<?php

require\_once "library.php";

// Your code goes here

?>

1. **Gyakorlati PHP**

Az előző fejezetek a PHP nyelv elemeit járták át. Ez a fejezet új programozási készségeire épít, hogy megtanítsa néhány gyakori de fontos gyakorlati feladatok végrehajtására. Megtanulja a húrok rendezett kezelésének legjobb módjait világos és tömör kód elérése érdekében, amely pontosan úgy jelenik meg a böngészőben, ahogyan szeretné, beleértve a fejlett dátum- és időkezelést. Azt is megtudhatja, hogyan hozhat létre és más módon módosíthat fájlokat, beleértve a felhasználók által feltöltötteket is.

* 1. **pintf használata**

Látta már a nyomtatás és visszhang funkciókat, amelyek egyszerűen szöveget adnak ki a böngészőben. De egy sokkal erősebb funkció, a printf, úgy szabályozza a kimenet formátumát, hogy lehetővé teszi speciális formázási karakterek karakterláncba helyezését. Minden formázáshoz karaktert, a printf elvárja, hogy átadjon egy argumentumot, amelyet ebben a formátumban fog megjeleníteni. Például a következő példa a %d konverzióspecifikátort használja a megjelenítéshez a 3-as érték tizedesjegyben:

printf("There are %d items in your basket", 3);

Ha a %d-t %b-re cseréli, a 3-as érték binárisan (11) jelenik meg. 7-1. táblázat

mutatja a támogatott konverziós specifikációkat.

*7-1. tábla. A printf konverziós specifikációk*

|  |  |  |  |
| --- | --- | --- | --- |
| Specifikáló | Konverziós művelet argumentum arg-ra | Példa (arg 123-ra) |  |
| % | % karakter megjelenítése (arg nem szükséges) | % |  |
| b | az arg megjelenítése bináris egész számként | 1111011 |  |
| c | ASCII karakter megjelenítése az arg-hoz | { |  |
| d | az arg megjelenítése előjeles decimális számént | 123 |  |
| e | az arg megjelenítése tudományos jelöléssel | 1.23000e+2 |  |
| f | az arg megjelenítése lebegőpontosként | 123.000000 |  |
| o | az arg megjelenítése oktális egész számlént | 173 |  |
| s | az arg megjelenítése karakterlácként | 123 |  |
| u | az arg megjelenítése előjel nélküli tizedesjegyként | 123 |  |
| x | az arg megjelenítése kisbetűs hexadecimális formában | 7b |  |
| X | az arg megjelenítése hexadecimális nagybetűvel | 7b |  |

A printf függvényben tetszőleges számú specifikációja lehet, mindaddig, amíg átadja a

egyező számú argumentumot, és mindaddig, amíg minden specifikáció előtt egy % szimbólum szerepel. Ezért a következő kód érvényes, és a következőt írja ki: "A nevem Simon. I'm

33 éves, ami hexadecimálisan 21":

printf("My name is %s. I'm %d years old, which is %X in hexadecimal", 'Simon', 33, 33);

Ha kihagy néhány érvet, akkor egy elemzési hibaüzenetet kap, amely tájékoztatja, hogy a jobb zárójel, ), váratlanul találkozott.  
A printf gyakorlatiasabb példája a színeket a HTML-ben decimális értékek segítségével állítja be. Mert Tegyük fel például, hogy tudja, hogy olyan színt szeretne, amelynek hármasértéke 65 vörös, 127 zöld és 245 kék, de nem szeretnéd ezt hexadecimálisra alakítani. Itt van egy egyszerű megoldás:

printf("<span style='color:#%X%X%X'>Hello</span>", 65, 127, 245);

Gondosan ellenőrizze az aposztrófok ('') közötti színmeghatározás formátumát. Először a font vagy hash jel (#), amelyet a színspecifikáció vár. Azután jöjjön három %X formátum-specifikáció, minden számhoz egy-egy. Az így kapott kimenet ebből a parancsból a következő:

**<span style='color:#417FF5'>Hello</span>**

Általában kényelmesnek találja a változókat vagy kifejezéseket argumentumként használni

printf. Például, ha a színek értékeit tárolta a három változóban: $r, $g, és $b, sötétebb színt hozhat létre ezzel:

printf("<span style='color:#%X%X%X'>Hello</span>", $r-20, $g-20, $b-20);

* 1. **Precíziós beállítás**

Nemcsak megadhat egy konverziótípust, hanem beállíthatja a pontosságát is a

megjelenített eredménynek. Például a valutaösszegek általában csak két számjegyű pontossággal jelennek meg. Számítás után azonban egy érték pontosabb lehet, mint 123,42 / 12, ami 10,285-öt eredményez. Annak biztosítása érdekében, hogy az ilyen értékek helyesen tárolódnak belül, de csak két számjegy pontossággal jelennjenek meg illessze be a ".2" karakterláncot a % szimbólum és a konverziómeghatározó közé:

printf("The result is: $%.2f", 123.42 / 12);

A kódnak a kimenete a következő:

**The result is $10.29**

De valójában még ennél is több irányításod van, mert megadhatsz olyat is hogy a kimenetet nullákkal vagy szóközzel írja-e be úgy, hogy a specifikátor elé bizonyos értékeket ír. A 7-1. példa négy lehetséges kombinációt mutat be.

*7-1.példa. Precíziós beállítás*

<?php

echo "<pre>"; // Enables viewing of the spaces

// Pad to 15 spaces

printf("The result is $%15f\n", 123.42 / 12);

// Pad to 15 spaces, fill with zeros

printf("The result is $%015f\n", 123.42 / 12);

// Pad to 15 spaces, 2 decimal places precision

printf("The result is $%15.2f\n", 123.42 / 12);

// Pad to 15 spaces, 2 decimal places precision, fill with zeros

printf("The result is $%015.2f\n", 123.42 / 12);

// Pad to 15 spaces, 2 decimal places precision, fill with # symbol

printf("The result is $%'#15.2f\n", 123.42 / 12);

?>

A kódnak a kimenete a következő:

**The result is $ 10.285000**

**The result is $00000010.285000**

**The result is $ 10.29**

**The result is $000000000010.29**

**The result is $##########10.29**

A működése egyszerű, ha jobbról balra halad (lásd 7-2. táblázat).

• A jobb szélső karakter a konverziós specifikáció: ebben az esetben az f a lebegő pont.

• Közvetlenül a konverziómeghatározó előtt, ha egy pont és egy szám van együtt,

akkor a kimenet pontosságát adjuk meg a szám értékeként.

• Függetlenül attól, hogy van-e precíziós megadó, ha van szám, akkor az azoknak a karaktereknek a számát jelöli, amelyekre a kimenetet ki kell tölteni. Az előző példában, ez 15 karakter. Ha a kimenet már egyenlő a vagy nagyobb, mint a kitöltési hossz, akkor ez az argumentum figyelmen kívül marad.

• A % szimbólum után a bal szélső megengedett paraméter egy 0, amelyet figyelmen kívül hagyunk, hacsak nem a kitöltési érték lett beállítva, ebben az esetben a kimenet nullákkal van kitöltve szóközök helyett. Ha nullától vagy szóköztől eltérő pad karakterre van szükség, akkor tetszés szerint bármelyiket használhatja, feltéve, hogy egyetlen idézetet ír elő jelölje meg, így: „#.

• A bal oldalon található a % szimbólum, amely elindítja az átalakítást.

*7-2. tábla. Konverzióspecifikációs összetevők*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Konverziók megkezdése | Pad karakter | Pad karakterek száma | Precíziók mutatása | Specifikus konverziók | Példa |
| % |  | 15 |  | f | 10.285000 |
| % | 0 | 15 | .2 | f | 000000000010.29 |
| % | ‘# | 15 | .4 | f | ########10.2850 |

* 1. **String Padding**

A karakterláncokat is kitöltheti a kívánt hosszúságra (ahogyan a számok esetében is teheti), kiválaszthat különböző kitöltési karaktereket, és még a bal és jobb oldali igazítás között is választhat.

A 7-2. példa különböző példákat mutat be.

*7-2. példa. String Padding*

<?php

echo "<pre>"; // Enables viewing of the spaces

$h = 'Rasmus';

printf("[%s]\n", $h); // Standard string output

printf("[%12s]\n", $h); // Right justify with spaces to width 12

printf("[%-12s]\n", $h); // Left justify with spaces

printf("[%012s]\n", $h); // Pad with zeros

printf("[%'#12s]\n\n", $h); // Use the custom padding character '#'

$d = 'Rasmus Lerdorf'; // The original creator of PHP

printf("[%12.8s]\n", $d); // Right justify, cutoff of 8 characters

printf("[%-12.12s]\n", $d); // Left justify, cutoff of 12 characters

printf("[%-'@12.10s]\n", $d); // Left justify, pad with '@', cutoff 10 chars

?>

Vegye figyelembe, hogy a weboldal elrendezéséhez a <pre> HTML címkét használtam, hogy megőrizzem minden megjelenítendő sor után az összes szóközt és az \n újsor karaktert minden sor után, hogy így jelenítse meg. A példa kimenete a következő:

**[Rasmus]**

**[ Rasmus]**

**[Rasmus ]**

**[000000Rasmus]**

**[######Rasmus]**

**[ Rasmus L]**

**[Rasmus Lerdo]**

**[Rasmus Ler@@]**

Ha kitöltési értéket ad meg, akkor ezzel egyenlő vagy nagyobb hosszúságú karakterláncok

értéke figyelmen kívül lesz hagyva, hacsak nem adunk meg olyan cutoff értéket, amely visszarövidíti a karakterláncokat ami kisebb, mint a padding értéke.

A 7-3. táblázat a karakterlánc-konverziós specifikátorok számára elérhető összetevőket mutatja be.

*7-3. táblázat. Karakterlánc-konverzió-specifikációs összetevők*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Konverziók megkezdése | job/bal indokolás | Pad karakter | Pad karakterek száma | Precíziók mutatása | Specifikus konverziók | Példa |
| % |  |  |  |  | s | [Rasmus] |
| % | - |  | 10 |  | s | [Rasmus ] |
| % |  | ‘# | 8 | .4 | s | [####Rasm] |

* 1. **printf használata**

Gyakran előfordul, hogy nem szeretné kiadni az átalakítás eredményét, de máshol kell használni a kódjában. Itt jön be a sprintf függvény. Ezzel elküldheti a böngésző helyett, hogy egy másik változóba adja ki.

Használhatja konverzió végrehajtására, mint a következő példában, amely visszaadja a

hexadecimális karakterlánc értéke a 65, 127, 245 RGB színcsoporthoz a $hexstringben:

$hexstring = sprintf("%X%X%X", 65, 127, 245);

Vagy tárolhatja a kimenetet, amely készen áll a későbbi megjelenítésre:

$out = sprintf("The result is: $%.2f", 123.42 / 12);

echo $out;

* 1. **Idő és dátum funkciók**

A dátum és idő nyomon követésére a PHP szabványos Unix időbélyegeket használ, amelyek egyszerűen az 1970. január 1-je óta eltelt másodpercek száma. Annak meghatározásához, hogy az aktuális időbélyegző, használhatja az idő függvényt:

echo time();

Mivel az értéket a rendszer másodpercben tárolja, az időbélyegző lekéréséhez legközelebbi

héten a következőt használja, amely hozzáad 7 nap × 24 óra × 60 perc × 60 másodpercet a visszaadott értékig:

echo time() + 7 \* 24 \* 60 \* 60;

Ha egy adott dátumhoz szeretne időbélyeget létrehozni, használhatja az mktime függvényt.

A kimenete a 946684800 időbélyeg, az első perc első másodpercében. 2000. év első napjának első órája:

echo mktime(0, 0, 0, 1, 1, 2000);

Az átadandó paraméterek balról jobbra haladva a következők:

• Az óra száma (0-23)

• A perc száma (0-59)

• A másodpercek száma (0–59)

• A hónap száma (1-12)

• A nap száma (1-31)

• Az év (1970–2038 vagy 1901–2038 PHP 5.1.0+ verzióval 32 bites aláírt rendszereken)

Felteheti a kérdést, hogy miért korlátozódik az 1970 és 2038 közötti évekre. Nos, ez azért van, mert a Unix eredeti fejlesztői az 1970-es évet választották az alapdátumnak, mivrl egyetlen programozónak sem kellene szüksége legyen ennél visszább menni! Szerencsére az 5.1.0-s verziótól a PHP támogatja az aláírt rendszereket 32 bites egész szám az időbélyeg, és az 1901 és 2038 közötti dátumok megengedettrk. Ez azonban még súlyosabb problémát jelent mint az eredeti, mert a Unix tervezői is így döntöttek senki sem használ Unixot körülbelül 70 év után, és ezért úgy gondolta, hogy megúszhatják az időbélyeg tárolását 32 bites értékként – ami 2038. január 19-én fog lejárni!

Ez létrehozza az Y2K38-as hibát (hasonlóan a millenniumi bughoz, amit az évek tárolása okozott

kétjegyű értékekként, és amelyeket szintén rögzíteni kellett). A PHP az 5.2-es verzióban bevezette a DateTime osztályt a probléma megoldására, de csak 64 bites architektúrán fog működni, ami a legtöbb számítógépen működik manapság (de használat előtt azért ellenőrizze).

A dátum megjelenítéséhez használja a dátum funkciót, amely számos formázási opciót támogat, amelyek lehetővé teszik a dátum tetszőleges módon történő megjelenítését. A formátum a következő:

date($format, $timestamp);  
  
A $format paraméternek olyan karakterláncnak kell lennie, amely a részletezett formázási specifikációkat tartalmazza a 7-4. táblázatban, és a $timestamp Unix időbélyegnek kell lennie. A teljes listához specifikációkat, lásd a [dokumentációt](https://www.php.net/manual/en/function.date.php). A következő parancs az aktuális dátumot és időt „2017. július 6., csütörtök – 13:38” formátumban adja ki:

echo date("l F jS, Y - g:ia", time());

*7-4 táblázat. A fő dátumfüggvény-formátum-specifikációk*

|  |  |  |
| --- | --- | --- |
| Formátumok | Leírás | Visszaadott érték |
| **Nap specifikációk** | | |
| d | a hónap napja két számjeggyel, kezdő 0-ákkal | 01-től 31-ig |
| D | a hét napja 3 betűvel | Mon to Sun |
| j | a hónap napja nem 0-ával | 1-től 31-ig |
| l | hét napjai teljes névvel | hétfőtől szombatig |
| N | hét napjai számokkal (hétfőtől vasárnapig) | 1-től 7-ig |
| S | a hónap egy napjának a rövidítése | st, nd, rd, or th |
| w | a hét napjai számokkal (vasárnaptól szombatig) | 0-tól 6-ig |
| z | az év egy napja | 0-tól 365-ig |
| **Hét specifikációk** | | |
| W | hét száma az évben | 01-től 52-ig |
| **Hónap specifikációk** | | |
| F | hónap neve | Januártól Decemberig |
| m | hónap száma kezdő 0-ával | 01-től 12-ig |
| M | hónap neve 3 betűvel | Jan.-tól Dec.-ig |
| n | hónap száma nem 0-ával | 1-től 12-ig |
| t | napok száma egy bizonyos hónapban | 28-tól 31-ig |
| **Év specifikációk** | | |
| L | szökőév | 1=igen, 0=nem |
| y | év 2 számjeggyel | 00-tól 99-ig |
| Y | év 4 számjeggyel | 0000-tól 9999-ig |
| **Idő specifikációk** | | |
| a | nappal előtt vagy után kisbetűvel | am vagy pm |
| A | nappal előtt vagy után nagybetűvel | AM vagy PM |
| g | a nap egy órája 12 órás időformátum nem 0 | 1-től 12-ig |
| G | a nap egy órája 24 órás időformátum nem 0 | 0-tól 23-ig |
| h | a nap egy órája 12 órás időformátum, kezdő 0 | 01-től 12-ig |
| H | a nap egy órája 24 órás időformátum, kezdő 0 | 00-tól 23-ig |
| i | percek kezdő 0-ával | 00-tól 59-ig |
| s | másodpercek kezdő 0-ával | 00-tól 59-ig |

* 1. **Dátum állandók**

Számos hasznos konstans használható ami dátum paranccsal adott formátumban adja vissza a dátumot. Például a dátum(DATE\_RSS) az aktuális értéket adja vissza dátumot és időt az RSS-hírcsatorna érvényes formátumában. Néhány a gyakrabban használt az állandók a következők:

DATE\_ATOM

Ez az Atom feedek formátuma. A PHP formátuma "Y-m-d\TH:i:sP" és példa kimenet: "2022-10-22T12:00:00+00:00".

DATE\_COOKIE

Ez a webszerverről vagy JavaScriptről beállított cookie-k formátuma. A PHP formátuma "l, d-M-y H:i:s T", a példa kimenete pedig "szerda, október 26-22.

12:00:00 UTC"

DATE\_RSS

Ez az RSS-hírcsatornák formátuma. A PHP formátuma "D, d M Y H:i:s O" és

példa kimenet: "Sze, 2022. október 26. 12:00:00 UTC".

DATE\_W3C

Ez a World Wide Web Consortium formátuma. A PHP formátuma "Y m-d\TH:i:sP", a példa kimenete pedig "2022-10-26T12:00:00+00:00".

A teljes lista a [dokumentációban](https://www.php.net/manual/en/class.datetime.php) található.

* 1. **Dátumellenőrzés használata**

Látta, hogyan jeleníthet meg egy érvényes dátumot számos formátumban. De hogyan lehet ellenőrizni hogy egy felhasználó megadott-e érvényes dátumot a programodban? A válasz átadni a hónap, nap és év a checkdate függvényhez, amely IGAZ értéket ad vissza, ha a dátum érvényes, vagy HAMIS, ha nem.

Például, ha bármely év szeptember 31-ét adja meg, az mindig érvénytelen dátum lesz. A 7-3. példa bemutatja a kódot, amelyet erre használhat. Így meg fogja találni ha az adott dátum érvénytelen.

*7-3. példa.**Egy dátum érvényességének ellenőrzése*

<?php

$month = 9; // September (only has 30 days)

$day = 31; // 31st

$year = 2022; // 2022

if (checkdate($month, $day, $year)) echo "Date is valid";

else echo "Date is invalid";

?>