6. ÓRA

Függvények

A függvény a jól szervezett program lelke, mert a programot könnyen olvashatóvá és újrahasznosíthatóvá teszi. Függvények nélkül a nagy programok kezelhetetlenek lennének. Ebben az órában a függvényeket tanulmányozzuk és mutatunk rá néhány példát, hogyan kímélhetnek meg minket az ismétlődésekből adódó pluszmunkától. Az órában a következőket tanuljuk meg:

- Hogyan hozhatunk létre függvényeket?
- Hogyan adjunk át a függvényeinknek értékeket és hogyan kapjuk meg tőlük az eredményt?
- Hogyan hívjunk meg függvényeket dinamikusan, változóban tárolt karakterlánc segítségével?
- Hogyan érjük el a függvényekből a globális változókat?
- Hogyan érjük el, hogy függvényeinknek „emlékezete” legyen?
- Hogyan adjunk át a függvényeknek hivatkozásokat?

Mit nevezünk függvénynek?

A függvényt egy gépnek tekinthetjük. A gép a bele töltött nyersanyagokkal addig dolgozik, amíg a kívánt terméket elő nem állítja vagy el nem éri kitűzött célját. A függvény értékeket vesz át tőlünk, feldolgozza azokat és végez velük valamit (például kiírja az eredményt a böngészőbe) vagy visszaad egy értéket, esetleg mindkettőt.

Ha a kedves Olvasónak egy süteményt kell csinálnia, maga süti meg. De ha több ezret, akkor esetleg készít vagy beszerez egy süteménysütő gépezetet. Hasonlóképp, amikor elhatározzuk, hogy függvényt írunk, a legfontosabb szempont, amit mérlegelnünk kell, az, hogy az ismétlődések csökkentésével akkorává zsugorodike a program, hogy rövidebb lesz a függvény használatával.

A függvény valójában egy zárt, önálló kódrészlet, melyet programunkból meghívhatunk. Amikor meghívjuk, a függvény törzse lefut. A függvénynek feldolgozás céljából értékeket adhatunk át. Amikor a függvény véget ér, a hívónak egy értéket ad vissza.

ÚJDONSÁG A függvény olyan kódrészlet, amely nem közvetlenül hajtódik végre, hanem a programból hívhatjuk meg: onnan, ahol épp szükség van rá. A függvények lehetnek beépítettek vagy felhasználó által megadottak. Működésükhöz szükségük lehet információkra és többnyire értéket adnak vissza.

Függvények hívása

Kétféle függvény létezik: a nyelvbe beépített függvény és az általunk létrehozott függvény. A PHP 4-ben rengeteg beépített függvény van. A könyv legelső PHP oldala egyetlen függvényhívásból állt:

```
print ("Hello Web!");
```

A print abból a szempontból nem jellegzetes függvény, hogy paramétereit nem kell zárójelbe tenni. A

```
print ("Hello Web!");
```

és

```
print "Hello Web!";
```

egyaránt helyes megoldások. Ez egy különleges függvény. Szinte az összes többi függvénynél kötelező a zárójel; akár kell paramétert átadnunk, akár nem.

A fenti példában a `print()` függvényt a `"Hello Web!"` szövegparaméterrel hívtuk meg. A program a karakterlánc kiírását hajtja végre. A függvényhívás egy függvénynévből (ebben az esetben ez a `print`) és az utána tett zárójelekből áll. Ha a függvénynek információt szeretnénk átadni, azt a függvény utáni zárójelbe tesszük. Az információt, amit ily módon adunk át a függvénynek, paraméternek hívjuk. Néhány függvénynek több paramétert kell átadni. A paramétereket vesszővel választjuk el.

ÚJDONSÁG A paraméter a függvénynek átadott érték. A paramétereket a függvényhívás zárójelén belülre kell írnunk. Ha több paramétert kell átadnunk, az egyes paramétereket vesszővel kell elválasztanunk egymástól. A paraméterek a függvényeken belül helyi (lokális) változóként érhetők el.

```
valamilyen_fuggveny ( $elso_parameter, $masodik_parameter );
```

A `print()` abból a szempontból tipikus függvény, hogy van visszatérési értéke. A legtöbb függvény, ha nincs „értelmes” visszatérési értéke, információt ad arról, hogy munkáját sikeresen befejezte-e. A `print()` visszatérési értéke logikai típusú (`true`, ha sikeres volt).

Az `abs()` függvény például egy szám típusú paramétert vár és a paraméter abszolútértékét adja vissza. Próbáljuk is ki a 6.1. példaprogrammal!

6.1. program A beépített abs() függvény használata

```
 1: <html>
 2: <head>
 3: <title>6.1. program A beépített abs() függvény
használata</title>
 4: </head>
 5: <body>
 6: <?php
 7: $szam = -321;
 8: $ujszam = abs( $szam );
 9: print $ujszam; // kiírja, hogy "321"
10: ?>
11: </body>
12: </html>
```

Ebben a példában a `$szam` nevű változóhoz a -321-es értéket rendeltük, majd átadtuk az `abs()` függvénynek, amely elvégzi a szükséges számításokat és visszaadja az eredményt. Ezt az új értéket rendeljük az `$ujszam` nevű változóhoz és kiírjuk az eredményt. A feladatot megoldhattuk volna segédváltozók segítsége nélkül is, az `abs()` függvényt adva közvetlenül a `print()` paramétereként:

```
print( abs( -321 ) );
```

A felhasználó által megírt függvényeket teljesen hasonló módon kell meghívnunk.

Függvények létrehozása

Függvényt a `function` kulcsszó segítségével hozhatunk létre:

```
funcion valamilyen_fuggveny( $parameter1, $parameter2 )
     {
     // itt van a függvény törzse
     }
```

A `function` kulcsszót a függvény neve követi, majd egy zárójelpár. Ha a függvény paramétereket igényel, a zárójelbe vesszővel elválasztott változókat kell tenni. A változók értékei a függvényhíváskor átadott paraméterek lesznek. A zárójelpárt akkor is ki kell írni, ha a függvény nem vesz át paramétereket.

A 6.2. példaprogram egy függvényt hoz létre.

6.2. program Függvény létrehozása

```
 1: <html>
 2: <head>
 3: <title>6.2. program Függvény létrehozása</title>
 4: </head>
 5: <body>
 6: <?php
 7: function nagyHello()
 8:      {
 9:      print "<h1>HELLO!</h1>";
10:      }
11: nagyHello();
12: ?>
13: </body>
14: </html>
```

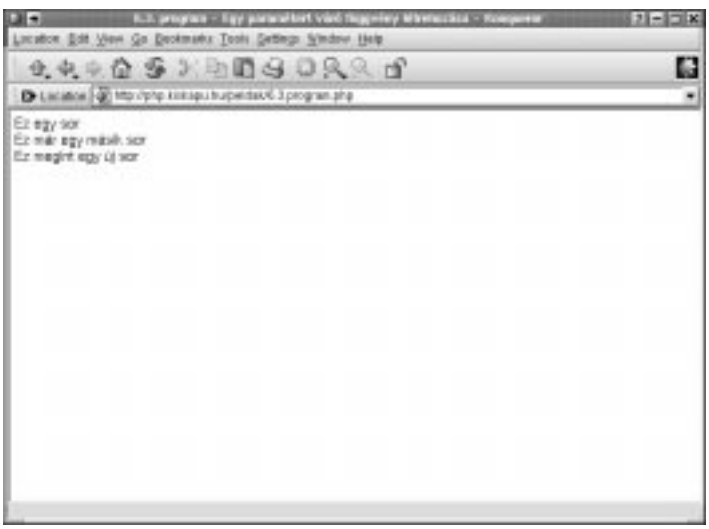
A fenti program egyszerűen kiírja a `"HELLO"` szöveget egy `<H1>` HTML elemben. A programban egy olyan `nagyHello()` nevű függvényt hoztunk létre, amely nem vesz át paramétereket. Ezért hagytuk üresen a zárójelpárt. Bár a `nagyHello()` működő függvény, mégsem túl hasznos. A 6.3. példaprogramban olyan függvényt láthatunk, amely egy paramétert fogad és valami hasznosat csinál vele.

6.3. program Egy paramétert váró függvény létrehozása

```
 1: <html>
 2: <head>
 3: <title>6.3. program Egy paramétert váró függvény
          létrehozása</title>
 4: </head>
 5: <body>
 6: <?php
 7: function sorKiir( $sor )
 8:     {
 9:     print ("$sor<br>\n");
10:     }
11: sorKiir("Ez egy sor");
12: sorKiir("Ez már egy másik sor");
13: sorKiir("Ez megint egy új sor");
14: ?>
15: </body>
16: </html>
```

6.1. ábra

*Függvény, amely egy karakterláncot, majd egy
 HTML elemet ír ki.*



A 6.3. példaprogram kimenetét a 6.1. ábrán láthatjuk. A `sorKiir()` függvény egy karakterláncot vár, ezért tettük a `$sor` nevű változót a zárójelbe. Amit a `sorKiir()` függvény zárójelei közé teszünk, az kerül a `$sor` nevű változóba. A függvény törzsében kiírjuk a `$sor` nevű változót, majd egy `<br>` elemet és egy újsor karaktert (hogy a HTML kód is olvasható legyen, ne csak a kimenet).

Ha most ki szeretnénk írni egy sort a böngészőbe, akkor meghívhatjuk a `sorKiir()` függvényt, ahelyett, hogy a `print()` függvénnyel oldanánk meg a problémát, így nem kell a sorvégi jeleket minden sor kiírásakor begépelnünk.

Függvények visszatérési értéke

A függvényeknek visszatérési értéke is lehet, ehhez a `return` utasításra van szükség. A `return` befejezi a függvény futtatását és az utána írt kifejezést küldi vissza a hívónak.

A 6.4. példaprogramban létrehozunk egy függvényt, amely két szám összegével tér vissza.

6.4. program Visszatérési értékkel rendelkező függvény

```
 1: <html>
 2: <head>
 3: <title>6.4. program Visszatérési értékkel rendelkező
    függvény</title>
 4: </head>
 5: <body>
 6: <?php
 7: function osszead( $elsoszam, $masodikszam )
 8:     {
 9:     $eredmeny = $elsoszam + $masodikszam;
10:     return $eredmeny;
11:     }
12: print osszead(3,5); // kiírja, hogy "8"
13: ?>
14: </body>
15: </html>
```

A 6.4. példaprogram a 8-as számot írja ki. Az `osszead()` függvényt két paraméterrel kell meghívni (itt a két paraméter a 3 és az 5 volt). Ezek az `$elsoszam` és a `$masodikszam` nevű változókban tárolódnak. Várhatóan az `osszead()` függvény e két változó eredményét adja össze és tárolja az `$eredmeny` nevű változóban. Az `$eredmeny` nevű segédváltozó használatát kiküszöbölhetjük:

```
function osszead( $elsoszam, $masodikszam )
     {
     return ( $elsoszam + $masodikszam );
     }
```

A `return` segítségével értéket és objektumot is visszaadhatunk, vagy esetleg „semmit”. Ha semmit sem adunk meg a `return` után, az csak a függvény futtatásának befejezését eredményezi. A visszatérési érték átadásának módja többféle lehet. Az érték lehet előre „beégetett”

```
return 4;
```

de lehet kifejezés eredménye

```
return ( $a / $b );
```

vagy akár egy másik függvény visszatérési értéke is:

```
return ( masik_fuggveny( $parameter ) );
```

Dinamikus függvényhívások

Lehetőségünk van rá, hogy karakterláncba tegyük egy függvény nevét és ezt a változót pontosan úgy tekintsük, mint ha maga a függvény neve lenne. Ezt a 6.5. példaprogramon keresztül próbálhatjuk ki.

6.5. program Dinamikus függvényhívás

```
1: <html>
2: <head>
3: <title>6.5. program Dinamikus függvényhívás</title>
4: </head>
5: <body>
6: <?php
```

6.5. program (folytatás)

```
 7: function koszon()
 8:  {
 9:  print "Jó napot!<br>";
10:  }
11: $fuggveny_tarolo = "koszon";
12: $fuggveny_tarolo();
13: ?>
14: </body>
15: </html>
```

Itt a `$fuggveny_tarolo` változóhoz a `koszon()` függvény nevével azonos karakterláncot rendeltünk. Ha ezt megtettük, a változót arra használhatjuk, hogy meghívja nekünk a `koszon()` függvényt, csupán zárójeleket kell tennünk a változó neve után.

Miért jó ez? A fenti példában csak még több munkát csináltunk magunknak azzal, hogy a `"koszon"` karakterláncot rendeltük a `$fuggveny_tarolo` nevű változóhoz. A dinamikus függvényhívás akkor igazán hasznos, ha a program folyását bizonyos körülményektől függően változtatni szeretnénk. Például szeretnénk mást és mást csinálni egy URL-kérés paraméterétől függően. Ezt a paramétert feldolgozhatjuk és értékétől függően más és más függvényt hívhatunk meg.

A PHP beépített függvényei még hasznosabbá teszik ezt a lehetőséget. Az `array_walk()` függvény például egy karakterláncot használ arra, hogy a tömb minden elemére meghívja a függvényt. Az `array_walk()` alkalmazására a tizenhatodik órában láthatunk példát.

Változók hatóköre

A függvényben használt változók az adott függvényre nézve helyiek maradnak. Más szavakkal: a változó a függvényen kívülről vagy más függvényekből nem lesz elérhető. Nagyobb projektek esetén ez megóvhat minket attól, hogy véletlenül két függvény felülírja azonos nevű változóik tartalmát.

A 6.6. példaprogramban létrehozunk egy változót a függvényen belül, majd megpróbáljuk kiíratni a függvényen kívül.

6.6. program Változók hatóköre: A függvényen belül létrehozott változó a függvényen kívülről nem elérhető

```
 1: <html>
 2: <head>
 3: <title>6.6. program Változók hatóköre: A függvényen
    belül létrehozott változó a függvényen kívülről
    nem elérhető</title>
 4: </head>
 5: <body>
 6: <?php
 7: function proba()
 8:      {
 9:      $probavaltozo = "Ez egy proba valtozo";
10:      }
11: print "proba valtozo: $probavaltozo<br>";
12: ?>
13: </body>
14: </html>
```

6.2 ábra

Kísérlet függvényen belüli változóra hivatkozásra.

A 6.6. példaprogram kimenetét a 6.2. ábrán láthatjuk. A `$probavaltozo` értéke nem íródik ki. Ez annak a következménye, hogy ilyen nevű változó a `proba()` nevű függvényen kívül nem létezik. Figyeljük meg, hogy a nem létező változóra történő hivatkozás nem eredményez hibát.

Hasonlóképp, a függvényen kívül meghatározott változó nem érhető el automatikusan a függvényen belülről.

Hozzáférés változókhoz a global kulcsszó segítségével

Alapértelmezés szerint a függvényeken belülről nem érhetjük el a máshol meghatározott változókat. Ha mégis megpróbáljuk ezeket használni, akkor helyi változót fogunk létrehozni vagy elérni. Próbáljuk ki ezt a 6.7. példaprogrammal:

6.7. program A függvényen kívül meghatározott változók a függvényen belül alapértelmezés szerint nem elérhetők

```
 1: <html>
 2: <head>
 3: <title>6.7. program A függvényen kívül meghatározott
    változók a függvényen belül alapértelmezés szerint
    nem elérhetők</title>
 5: </head>
 6: <body>
 7: <?php
 8: $elet = 42;
 9: function eletErtelme()
10:      {
11:      print "Az élet értelme: $elet<br>";
12:      }
13: eletErtelme();
14: ?>
15: </body>
16: </html>
```

6.3. ábra

Globális változó elérési kísérlete függvényen belülről

A 6.7. példaprogram kimenetét a 6.3. ábrán láthatjuk. Amint azt vártuk, az `eletErtelme()` függvény nem tudta elérni az `$elet` változót; így az `$elet` a függvényen belül üres. Ezt láthatjuk, amikor a függvény kiírja a változó értékét. Mindent figyelembe véve ez egy jó dolog. Megmenekültünk az azonos nevű változók ütközésétől és a függvény paramétert igényelhet, ha meg szeretne tudni valamit a „külvilág"-ról. Esetenként azonban egy-egy fontos globális (függvényen kívüli) változót anélkül szeretnénk elérni, hogy azt paraméterként át kellene adnunk. Ez az a helyzet, ahol a `global` kulcsszónak létjogosultsága van. A 6.8. példaprogram a `global` kulcsszóval állítja vissza a világ rendjét.

6.8. program Globális változó elérése a global kulcsszó segítségével

```
 1: <html>
 2: <head>
 3: <title>6.8. program Globális változó elérése
    a global kulcsszó segítségével</title>
 4: </head>
 5: <body>
 6: <?php
 7: $elet=42;
 8: function eletErtelme()
 9:  {
10:  global $elet;
11:  print "Az élet értelme: $elet<br>";
12:  }
13: eletErtelme();
14: ?>
15: </body>
16: </html>
```


6.4. ábra

Globális változó függvényből történő sikeres elérése a global kulcsszó segítségével

A 6.8. példaprogram kimenetét a 6.4. ábrán láthatjuk. Miután az `eletErtelme()` függvényben az `$elet` változó elé a `global` kulcsszót tesszük, a függvényen belül a külső, globális `$elet` változót érhetjük el.
Minden függvényben, amely globális változót szeretne elérni, használnunk kell a `global` kulcsszót.

Legyünk óvatosak! Ha most az `$elet` nevű változót a függvényen belül megváltoztatjuk, annak a program egészére hatása lesz. A függvénynek átadott paraméter általában valamilyen érték másolata; a paraméter megváltoztatásának a függvény vége után semmilyen hatása nincs. Ha azonban egy globális változó értékét változtatjuk meg egy függvényen belül, akkor az eredeti változót változtattuk meg és nem egy másolatot. Ezért a `global` kulcsszót lehetőleg minél kevesebbszer használjuk.

Állapot megőrzése a függvényhívások között a static kulcsszó segítségével

A függvényen belüli változóknak egészében véve rövid, de boldog életük van. Létrejönnek például akkor, amikor a `szamozottCimsor()` függvényt meghívjuk és eltűnnek, amikor a függvény futása befejeződik. Tulajdonképpen ennek így is kell lennie. A lehető legjobb úgy felépíteni egy programot, hogy az független és kis tudású függvényekből álljon. Esetenként azonban jól jönne, ha egy függvénynek lehetne valamilyen kezdetleges memóriája.

Tegyük fel például, hogy tudni szeretnénk, hányszor fut le egy adott függvény. Miért? Példánkban a függvény célja számozott címsor készítése, ami egy dinamikusan létrejövő dokumentációt eredményez.

Itt persze használhatnánk a `global` kulcsszóról újonnan szerzett tudásunkat. Ennek alapján a 6.9. példaprogramhoz hasonlót írhatnánk.

6.9. program Változó értékének függvényhívások közti megőrzése a global kulcsszó segítségével

```
1: <html>
2: <head>
3: <title>6.9. program Változó értékének függvényhívások
   közti megőrzése a global kulcsszó
   segítségével</title>
4: </head>
5: <body>
6: <?php
```

6.9. program (folytatás)

```
 7: $fvHivasokSzama = 0;
 8: function szamozottCimsor( $cimszoveg )
 9:       {
10:       global $fvHivasokSzama;
11:       $fvHivasokSzama++;
12:       print "<h1>$fvHivasokSzama. $cimszoveg</h1>";
13:       }
14: szamozottCimsor("Alkatrészek");
15: print("Az alkatrészek széles skáláját gyártjuk<p>");
16: szamozottCimsor("Mütyürök");
17: print("A legjobbak a világon<p>");
18: ?>
19: </body>
20: </html>
```

6.5. ábra

A függvényhívások számának nyomon követése a global kulcsszó használatával

Ez úgy működik, ahogy azt vártuk. Létrehoztuk a `$fvHivasokSzama` nevű változót a `szamozottCimsor()` függvényen kívül. A változót a függvény számára a `global` kulcsszó segítségével tettük elérhetővé. A 6.9. példaprogram kimenetét a 6.5. ábrán láthatjuk.

Valahányszor meghívjuk az `szamozottCimsor()` nevű függvényt, a `$fvHivasokSzama` változó értéke eggyel nő, majd a megnövelt értékkel a címsort teljessé tesszük.

Ez nem igazán elegáns megoldás. A `global` kulcsszót használó függvények nem függetlenek, nem tekinthetők önálló programrésznek. Ha a kódot olvassuk vagy újrahasznosítjuk, figyelnünk kell az érintett globális változókra. Ez az a pont, ahol a `static` kulcsszó hasznos lehet. Ha egy függvényen belül egy változót

a `static` kulcsszóval hozunk létre, a változó az adott függvényre nézve helyi marad. Másrészt a függvény hívásról hívásra „emlékszik” a változó értékére. A 6.10. példaprogram a 6.9. példaprogram `static` kulcsszót használó változata.

6.10. program Függvényhívások közötti állapot megőrzése a static kulcsszó használatával

```
 1: <html>
 2: <head>
 3: <title>6.10. program Függvényhívások közötti állapot
           megőrzése a static kulcsszó
           használatával</title>
 4: </head>
 5: <body>
 6: <?php
 7: function szamozottCimsor( $cimszoveg )
 8:      {
 9:      static $fvHivasokSzama = 0;
10:      $fvHivasokSzama++;
11:      print "<h1>$fvHivasokSzama. $cimszoveg</h1>";
12:      }
13: szamozottCimsor("Alkatrészek");
14: print("Az alkatrészek széles skáláját gyártjuk<p>");
15: szamozottCimsor("Mütyürök");
16: print("A legjobbak a világon<p>");
17: ?>
18: </body>
19: </html>
```

A `szamozottCimsor()` függvény így teljesen független lett. Amikor a függvényt először hívjuk meg, a `$fvHivasokSzama` nevű változó kezdeti értéket is kap. Ezt a hozzárendelést a függvény további meghívásainál figyelmen kívül hagyja a PHP, ehelyett az előző futtatáskor megőrzött értéket kapjuk vissza. Így a `szamozottCimsor()` függvényt már beilleszthetjük más programokba és nem kell aggódnunk a globális változók miatt. Bár a 6.10. példaprogram kimenete teljesen megegyezik a 6.9. példaprograméval, a programot elegánsabbá és hordozhatóbbá tettük. Gondoljunk bele például, mi történne akkor, ha a függvény 6.9. példaprogrambeli változatát egy olyan programba illesztenénk bele, amelyben van egy `$fvHivasokSzama` nevű változó.

Paraméterek további tulajdonságai

Már láttuk, hogyan kell függvényeknek paramétereket átadni, de van még néhány hasznos dolog, amiről még nem beszéltünk. Ebben a részben megtanuljuk, miként lehet a függvényparamétereknek alapértelmezett értéket adni és megtanuljuk azt is, hogyan kell változóinkra mutató hivatkozást átadni a változó értékének másolata helyett.

Paraméterek alapértelmezett értéke

A PHP nyelv remek lehetőséget biztosít számunkra rugalmas függvények kialakításához. Eddig azt mondtuk, hogy a függvények többsége paramétert igényel. Néhány paramétert elhagyhatóvá téve függvényeink rugalmasabbá válnak.

A 6.11. példaprogramban létrehozunk egy hasznos kis függvényt, amely egy karakterláncot egy `HTML font` elembe zár. A függvény használójának megadjuk a lehetőséget, hogy megváltoztathassa a `font` elem `size` tulajdonságát, ezért a karakterlánc mellé kell egy `$meret` nevű paraméter is.

6.11. program Két paramétert igénylő függvény

```
 1: <html>
 2: <head>
 3: <title>6.11. program Két paramétert igénylő
    függvény</title>
 4: </head>
 5: <body>
 6: <?php
 7: function meretez( $szoveg, $meret )
 8:  {
 9:  print "<font size=\"$meret\"
face=\"Helvetica,Arial,Sans-Serif\">$szoveg</font>";
10: }
11: meretez("Egy címsor<br>",5);
12: meretez("szöveg<br>",3);
13: meretez("újabb szöveg<BR>",3);
14: meretez("még több szöveg<BR>",3);
15: ?>
16: </body>
17: </html>
```

6.6. ábra

Függvény, amely formázott karakterláncokat ír ki a böngészőbe

A 6.11. példaprogram kimenetét a 6.6. ábrán láthatjuk. Függvényünk ugyan hasznos, de valójában a `$meret` nevű paraméter majdnem mindig 3. Ha kezdőértéket rendelünk ehhez a paraméterhez, akkor az elhagyható lesz. Így ha a függvényhívásban nem adunk értéket ennek a paraméternek, akkor az általunk meghatározott értéket fogja felvenni. A 6.12. példaprogram ennek a módszernek a segítségével teszi a `$meret` paramétert elhagyhatóvá.

6.12. program Függvény elhagyható paraméterrel

```
 1: <html>
 2: <head>
 3: <title>6.12. program Függvény elhagyható
    paraméterrel</title>
 4: </head>
 5: <body>
 6: <?php
 7: function meretez( $szoveg, $meret=3 )
 8:      {
 9:      print "<font size=\"$meret\" face=\"Helvetica,
              Arial,Sans-Serif\">$szoveg</font>";
10:      }
11: meretez("Egy címsor<br>",5);
12: meretez("szöveg<br>");
13: meretez("újabb szöveg<BR>");
14: meretez("még több szöveg<BR>");
15: ?>
16: </body>
17: </html>
```

Ha a `meretez()` függvénynek van második paramétere, akkor a `$meret` nevű változót ez fogja meghatározni. Ha a függvényhíváskor nem adunk meg második paramétert, akkor a függvény a 3 értéket feltételezi. Annyi elhagyható paramétert adhatunk meg, ahányat csak akarunk, de arra vigyáznunk kell, hogy az elhagyható paramétereknek a paraméterlista végén kell szerepelniük.

Ezt a korlátozást kikerülhetjük, sőt még a paraméterek sorrendjét sem kell fejben tartanunk, ha a paramétereket egyetlen asszociatív tömbben adjuk át. (Erről bővebben a hetedik, tömbökről szóló fejezetben lesz szó).

Hivatkozás típusú paraméterek

Amikor a függvényeknek paramétereket adunk át, a helyi paraméter-változókba a paraméterek értékének másolata kerül. Az e változókon végzett műveleteknek a függvényhívás befejeződése után nincs hatásuk az átadott paraméterekre. Ennek igazolására futtassuk le a 6.13. példaprogramot.

6.13. program Függvényparaméter érték szerinti átadása

```
 1: <html>
 2: <head>
 3: <title>6.13. program Függvényparaméter érték szerinti
           átadása</title>
 4: </head>
 5: <body>
 6: <?php
 7: function ottelTobb( $szam )
 8:      {
 9:      $szam += 5;
10:      }
11: $regiSzam = 10;
12: ottelTobb( $regiSzam );
13: print( $regiSzam );
14: ?>
15: </body>
16: </html>
```

Az `ottelTobb()` függvény egyetlen számot vár, majd ötöt ad hozzá. Visszatérési értéke nincs. A főprogramban a `$regiSzam` nevű változónak értéket adunk, majd ezzel a változóval meghívjuk az `ottelTobb()` függvényt. A `$regiSzam` változó

értékének másolata kerül a `$szam` nevű változóba. Amikor kiíratjuk a függvényhívás után a `$regiSzam` változó értékét, azt találjuk, hogy az érték még mindig 10. Alapértelmezés szerint a függvények paraméterei érték szerint adódnak át. Más szavakkal, a változók értékéről helyi másolat készül.

Lehetőségünk van arra is, hogy ne a változó értékét, hanem arra mutató hivatkozást adjunk át. (Ezt cím szerinti paraméterátadásnak is hívják.) Ez azt jelenti, hogy a változóra mutató hivatkozással dolgozunk a függvényben és nem a változó értékének másolatával. A paraméteren végzett bármilyen művelet megváltoztatja az eredeti változó értékét is. Hivatkozásokat függvényeknek úgy adhatunk át, hogy az átadandó változó vagy a paraméter neve elé egy `&` jelet teszünk. A 6.14. és a 6.15. példaprogram az előző problémára mutatja be a két előbb említett módszert.

6.14. program Cím szerinti paraméterátadás változóra mutatkozó hivatkozás segítségével

```
 1: <html>
 2: <head>
 3: <title>6.14. program Cím szerinti paraméterátadás
    változóra mutató hivatkozás segítségével</title>
 4: </head>
 5: <body>
 6: <?php
 7: function ottelTobb( $szam )
 8:  {
 9:  $szam += 5;
10:  }
11: $regiSzam = 10;
12: ottelTobb( &$regiSzam );
13: print( $regiSzam );
14: ?>
15: </body>
16: </html>
```

6.15. program Cím szerinti paraméterátadás a függvénydeklaráció módosításával

```
 1: <html>
 2: <head>
 3: <title>6.15. program Cím szerinti paraméterátadás
    a függvénydeklaráció módosításával</title>
 4: </head>
 5: <body>
 6: <?php
 7: function ottelTobb( &$szam )
 8:     {
 9:     $szam += 5;
10:     }
11: $regiSzam = 10;
12: ottelTobb( $regiSzam );
13: print( $regiSzam );
14: ?>
15: </body>
16: </html>
```

Ha az átadandó paramétert alakítjuk hivatkozássá, akkor nem fogjuk elfelejteni, hogy az átadott paraméter értéke megváltozhat, hiszen mi tettük hivatkozássá. Ennek a változatnak viszont az a veszélye, hogy elfelejtjük kitenni az `&` jelet. (C programozók már biztosan tapasztalták...) Ha a második megoldást választjuk, akkor lehet, hogy elfelejtjük, hogy a paraméter értéke megváltozhat, viszont nem felejtjük el kitenni az `&` jelet, mivel nem is kell kitennünk. Mindent egybevéve talán több értelme van annak, hogy a függvénydeklarációban írunk `&` jelet a paraméter neve elé. Így biztosak lehetünk benne, hogy az egyes függvényhívások azonos módon viselkednek.

Összefoglalás

Ebben az órában megtanultuk, mik azok a függvények és hogyan kell őket használni. Megtanultuk, hogyan kell létrehozni és paramétereket átadni nekik. Elsajátítottuk a `global` és a `static` kulcsszavak használatát. Megtanultuk, hogyan kell a függvényeknek hivatkozásokat átadni és a paramétereknek alapértelmezett értéket adni.

Kérdések és válaszok

A global kulcsszón kívül van más mód arra, hogy egy függvény hozzá tudjon férni vagy módosítani tudjon egy globális változót?
A globális változókat a programon belül bárhonnan elérhetjük a `$GLOBALS` nevű asszociatív tömb segítségével. A `$proba` nevű globális változót például `$GLOBALS["proba"]`-ként érhetjük el. Az asszociatív tömbökről a következő órában tanulunk.

A globális változót a függvényen belül akkor is módosíthatjuk, ha a függvény paraméterként egy arra mutató hivatkozást kapott.

Kérdés: Lehet-e a függvényhívásokat a változókhoz hasonlóan karakterláncba ágyazni?
Nem. A függvényhívásoknak az idézőjeleken kívül kell lenniük.

Műhely

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

Kvíz

1. Igaz-e, hogy ha egy függvénynek nem kell paramétert átadni, akkor a függvény neve után nem szükséges kitenni a zárójelpárt?

2. Hogyan lehet függvényből értéket visszaadni?

3. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer()
     {
     $szam = $szam * 10;
     }

tizszer();
print $szam;
```

4. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer()
     {
     global $szam;
     $szam = $szam * 10;
     }

tizszer();
print $szam;
```

5. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer($sz)
     {
     $sz = $sz * 10;
     }

tizszer($szam);
print $szam;
```

6. Mit ír ki az alábbi kódrészlet?

```
$szam = 50;

function tizszer(&$sz)
     {
     $sz = $sz * 10;
     }

$tizszer($szam);
print $szam;
```


Feladatok

1. Írjunk függvényt, amely négy karakterlánc típusú értéket fogad és olyan karakterlánccal tér vissza, amely paramétereit HTML cellaelemekbe (TD) zárja!