Php „ismétlés”

Bevezetés - történelem

PHP, azaz Hypertext Preprocessor. Tulajdonképpen kiszolgálóoldali programozási nyelv, amit jellemzően HTML oldalakon használnak. Segítségével aktív, dinamikus weboldalakat készíthetünk. A hagyományos HTML lapokkal ellentétben azonban a kiszolgáló a PHP parancsokat nem küldi el az ügyfélnek, azokat a kiszolgáló oldalán a PHP értelmező dolgozza fel.

A PHP modult az Apache webszerverbe illesztve lehet használni és a PHP kódot közvetlenül a HTML lap forrásszövegébe írhatjuk. Ráadásul a programjainkban lévő HTML elemek érintetlenül maradnak. Ennek óriási előnye, hogy az oldalak dizájnja és lefutási logikája a lapon belül világosan elkülöníthető egymástól. Így külön ember avagy emberek foglalkozhatnak az oldal felépítésével, látványával, míg mások a programot írják, anélkül, hogy egymás munkájába komolyabban beavatkoznának. A PHP lehetőséget ad elválasztani a kódolási, tervezési és az összeállítási szakaszt.

A kódok végezhetnek adatbázis-lekérdezést, dinamikusan létrehozhatnak képeket, fájlokat olvashatnak, írhatnak, kapcsolatot létesíthetnek távoli kiszolgálókkal, stb.

Végül pedig a PHP kódok kimenete a megadott HTML elemekkel együtt kerül az ügyfélhez, s számára a PHP kód rejtett marad.

A PHP előnyei:

- Nyílt forráskódú
- Hordozható: UNIX, Linux, Windows rendszereken egyaránt használható
- Rengeteg hasznos függvényt tartalmaz, továbbiak szerezhetők hozzá az Internetről.
- Apache és IIS alatt egyaránt működik
- CGI-ként és modulként is használható.

Munkakönyvtár - környezet

A PHP nyelv

A HTML lap forráskódjában az Apache felismeri a PHP kilépő szekvenciát <? és attól kezdve a PHP kódot értelmezi, a lezáró szekvenciáig ?>. Mindezt azon kiterjesztésű fájlok esetén teszi, melyeket az Apache httpd.conf fájljában megadtunk:

```
AddType          application /x-httpd-php  .php
```

Elnevezés	Kezdőelem (kilépő szekvencia)	Záróelem (lezáró szekvencia)
Hagyományos	<?php	?>
Rövid	<?	?>
ASP stílusú	<%	%>
Script elem	<SCRIPT LANGUAGE=”PHP”>	</SCRIPT>

```
<html>
<body gbcolor=white>
<?
  print ’Hello!<p>’;
  print ’Ezt a PHP fordító írja ide.’;
?>
```

```
</body>
</html>
```

A futtatás eredménye:

Hello

Ezt a PHP fordító írja ide.

Az Apache észlelte a <? jelet, ettől kezdve végrehajtotta a PHP nyelvű utasításokat, ennek eredményét küldi el a webszerver a kliens gép böngészőjének:

```
<html>
<body gbcolor=white>
Hello<p>Ezt a PHP fordító írja ide.</body>
</html>
```

A szerkesztett dokumentum állhat csupán PHP kódból is:

```
<?
  print  '<body gbcolor=white>
 print 'Hello!<p>';
 print 'Ezt a PHP fordító írja ide.';
 print '</body>';
?>
```

Megjegyzések:
vagy // : egy soros megjegyzés
/* több soros megjegyzés */

Változó deklarálás:
Az egyszerű skalár változók többféle típust tartalmazhatnak, ezeket a program használatuk során automatikusan konvertálja a megfelelő típusra. Változó bevezetése a $ jel segítségével történik, ezt követi a változót azonosító név.
Példák:

Számok:

```
$a = 1234;          # Egész, tízes számrendszerben
$b = -1234;         # Egész, tízes számrendszerben
$c = 0123;          # Nyolcas (octal) számrendszerbeli szám
$d = 0x12;          # Tizenhatos (hexa) számrendszerbeli szám
$e = 12.123;        # Lebegőpontos (double) szám
$f = 123.1e2;       # Lebegőpontos (double) szám
```

Karakteres típus:

```
$s1='Szöveg';
$s2="Szöveg";
$s3="Szöveg benne 'ezt' használhatom";
$s4='Szöveg benne "ezt" használhatom';
```

Logikai típus (boolean):

```
$L=true;            # true vagy false
```

A backslash karakterek a C-hez hasonlóan használhatók.

Példa

```
<?
  $a=1;
  $b=2;
  $c=$a+$b;
  $s1='$a + $b = $c';     # 'esetén a $ jeleket szövegnek értelmezi
  $s2="$a + $b = $c";     # "esetén a változók értékeit behelyettesíti.
  print" Első: $s1 <br>";
  print"Második: $s2 <br>";
?>
```

Eredmény:

Első: $a + $b = $c Második: 1 + 2 = 3

Szöveg összefűzés (konkatenáció)
. (pont) operátorral. Használatakor a változók érékeit behelyettesítve fűzi a szöveghez.

Pl.:

```
<?
  $a=1;
  $a=2;
  $s="Összegük: ";
  $eredmeny = $a.' + '.$b.' = '.$c;
  print ' Az eredeti számok: a = '.$a', b= '.$b"<p>\n";
  print $s.$eredmeny;
>?
```

Eredmény:

Az eredeti számok: a=1 b=2 Összegük: 1 + 2 = 3

Típus lekérdezése és módosítása:
Típus lekérdezése a gettype() függvény segítségével történik:

```
print gettype($a);
```

Típus módosítása pedig a settype() függvénnyel.

```
settype ($a,'string');
```

Explicit típuskonverzió alkalmazásakor a a változó neve elé zárójelbe írt adattípus segítségével a változó értékének általunk meghatározott típusúvá alakított másolatát kapjuk.

```
$tarolo = (double)$a;
```

Példa:

```
</head>
<body>
<?
  $a=3;
```

```
  print gettype($a);
  settype ($a,'string');
  print"<br>";
  print gettype($a);
  print"<br>";
  print ("$a+4");
  //Explicit típuskonverzió
  $tarolo = (double)$a;
  print "<br>";
  print gettype($tarolo);
  print"<br>";
  print($nev = "alma");
  $nev="Enni ".$nev."!";
  print"<br>";
  print($nev);
  print"<br>";
?>
</body>
</html>
```

Eredmény:

integer
string
3+4
double
alma
Enni alma!

Tömbök és asszociatív tömbök:

Tömbök:

Egyszerű számokkal indexelt tömb. Jelzésére a [] karaktereket használjuk a változó neve után.

Tömbök létrehozása:

```
$t[0]='alma';
$t[1]='körte';
$t[]=5;             # integerként jön létre, az utolsó helyen.
$t[]='citrom';      # stringként jön létre, az utolsó helyen.
$m[0][0] = '1';     # többdimenziós tömb
$m[0][1] = '2';
```

Asszociatív tömb:

Valamilyen hivatkozással indexelt tömb, pl.: stringgel, változóval.

```
$s['alma']=1;
$s['citrom']=2;        #$s[$citrom] – ként is hivatkozhatunk rá.
$s[$szilva]=3;         #$s['szilva'] – ként is hivatkozhatunk rá.
```

Operátorok, kifejezések:

Egyszerű aritmetika:

Operátor	Példa	Művelet
+	$a + $b	Összeadás
-	$a - $b	Kivonás

*	$a * $b	Szorzás
/	$a / $b	Osztás
%	$a % $b	Maradékképzés

Bitműveletek:

Operátor	Példa	Művelet
&	$a & $b	Bitenkénti logikai ÉS
|	$a | $b	Bitenkénti logikai VAGY
^	$a ^ $b	Bitenkénti logikai KIZÁRÓ VAGY
~	~ $a	Bitenkénti logikai NEM
<	$a < $b	Bitrotáció (shift) balra
>	$a > $b	Bitrotáció (shift) jobbra

Értékadás:

Operátor	Példa	Művelet
=	$a=5	Egyszerű értékadás
= (.. = ..)	$a=($b=5)+2	Többszörös értékadás ($a=7, $b=5)
+=, -=, *=m /=, %=	$a+=$b	Hozzárendelés: $a=$a+$b
&=, |=, ^=, ~=	$a&=$b	Hozzárend.: $a=$a&$b
<=	%a<=2	$a=$a<2
>=	$a>3	$a=$a>3
.=	$a.=$b	$a=$a.$b: szöveg típusú változóra vonatkozik, illetve eredményeként szöveget kapunk.

Logikai (összehasonlító) operátorok:

Operátor	Példa	Művelet
==	$a==$b	Egyenlősség
===	$a===$b	Azonosság (azonos, ha értéke és típusa is megegyezik)
!=	$a!=$b	Egyenlőtlenség
!==	$a!==$b	Nem azonos
<	$a<$b	Kisebb
>	$a>$b	Nagyobb
<=	$a<=$b	Kisebb vagy egyenlő
>=	$a>=$b	Nagyobb vagy egyenlő
(E1)?(E2):(E3)	($a=2)?($b=2):($c=2)	Értékadás kiértékelés alapján

Logikai operátorok:

Operátor	Példa	Művelet
&& (and)	$a && $b, $a and $b	És kapcsolat
|| (or)	$a || $b	Vagy kapcsolat

xor	$a xor $b	Kizáró vagy kapcsolat
!	! $a	Tagadás

Növelés, csökkentés:

Operátor	Példa	Művelet
++	++$a	Kifejezés=$a, majd $a=$a+1
--	--$a	Kif=$a, majd $a=$a-1
++	$a++	$a=$a+1, majd kif=$a
--	$a--	$a=$a-1, majd kif=$a

Vezérlési szerkezetek:

IF – ELSEIF – ELSE

```
if ($a==$b) {
  print ”A és B egyenlőek<br>”;
} elseif ($a<$b) {
  print ”A kisebb,mint B<br>”;
} else {
  print ”A nagyobb, mint B<br>”;
}
```

WHILE – elöltesztelő ciklus

```
$i=0;
while ($i<5) {
  print ”$i <br>”;
  $i++;
}
```

{ } jelek használata helyett használható az endwhile utasítás is, a } jel helyén.

DO – WHILE – hátultesztelős ciklus

```
$i=0;
do {
  print ”$i <br>”;
  $i++;
} while ($i<5);
```

FOR

```
for ($i=0; $i<5; $i++) {
  $j=$i*$j;
  print ”$i”;
}
```

FOREACH

Használata elsősorban tömbök esetén indolkolt, amikor valamilyen műveletsort a tömb minden egyes elemével el akarunk végezni. Egész számokkal indexelt tömbök esetén a ($tömb as $érték) alakot használjuk, ahol a foreach a $tömb elemeit sorban behelyettesíti az $érték változóba, és ezt használhatjuk a ciklusmagban.

```
$t[]=’Egy’;
$t[]=’Kettő’;
```

```
$t[]='Három';
foreach ($t as $x) print "$x";
```

Asszociatív tömböknél a tömb indexét (key) és az adott elem értékét (value) is megkaphatjuk a ($tömb as $kulcs => $érték) kifejezéssel.

```
foreach ($GLOBALS as $kulcs =>$érték) {
  print "$kulcs => $érték <br>\n";
}
```

SWITCH-CASE – számokkal, stringekkel egyaránt.

```
switch ($a) {
  case 0:
     print "0<br>";
     break;
  case 1:
     print "1<br>";
     break;
  default:
     print "semmi <br>";
     break;
}
```

BREAK, CONTINUE

- Break: megszakítja az aktuális blokkot, kilép belőle
- Continue: a ciklus hátralevő részét átugorva a következő iterációt indítja.

Függvények:

```
function függvénynév (paraméterlista) {
…
return visszatérési érték;
}
```

Paraméterek, visszatérési érték (return) nem kötelező.
A függvényeken belül használt változókat a PHP lokálisnak tekinti, ezért, ha globális változót szeretnénk használni, azt kétféle módon tehetjük meg:

- globals kulcsszót felhasználva deklaráljuk a változókat: global $a,$b;
- $GLOBALS[] tömböt felhasználva a függvényen kívüli változókra nevükkel hivatkozhatunk. Például:

```
function fuggv ($a) {
  $ip=$GLOBALS['REMOTE_ADR'];
  print "A kliens gép IP címe: $ip<br>";
}
```

Paraméter értékének átadása:

- Érték szerinti átadás: $a: A megadott változónévvel létrejött egy lokális, átmeneti példány a változóból, a függvény ezt használja.
- Hivatkozás vagy referencia szerinti átadás: &$a: ez esetben nem keletkezik lokális példány, hanem a globális hivatkozással dolgozunk.

Az include és require utasítások:
Segítségükkel PHP kódunkba külső fájlokat szúrhatunk be. A külső fájl egyaránt lehet egyszerű szövegfájl (HTML) vagy PHP forráskód.

- include: beolvassa és értelmezi (HTML esetén egyszerűen kiírja) a beszúrandó fájlt. Pl.: szabányos fejléc alkalmazásakor: include(”header.html”);
- require: hozzáfűzi az adott PHP állományhoz a megadott fájlt a megadott helyen. Pl.: saját függvényeinket célszerű külső fájlokba szervezni, ilyenkor a require a jobb választás: require(”functions.php”);

Környezeti változók:
A PHP a saját változóink mellett előre definiált változókat és konstans értékeket is szolgáltat, ezek segítségével főleg a program futási környezetéről, az operációs rendszerről, a kliensről és egyéb rendszerspecifikus adatokról kaphatunk képet.
Egy HTML form GET-tel vagy POST-tal történő elküldésekor az input mezőket a PHP automatikusan PHP változókká alakítja.
Néhány példa:

- $GLOBALS[]: asszociatív tömb, mely az összes PHP változót tartalmazza. Elsősorban saját függvényeinken belül lehet jól használni, ahol a változók alapvetően lokálisak. A tömb segítségével bármelyik globális változót elérhetjük.
- $HTTP_GET_VARS[]: A GET metódussal átadott paramétereket tartalmazó asszociatív tömb.
- $HTTP_POST_VARS[]: A POST metódussal átadott paramétereket tartalmazó asszociatív tömb.
- $QUERY_STRING: A GET metódusnál megismert QUERY_STRING környezeti változó.
- $PHP_AUTH_USER: Jelszavas védelem esetén a felhasználó által begépelt loginnév.
- $PHP_AUTH_PW: Jelszavas védelem esetén a felhasználó által begépelt jelszó.
- $REMOTE_ADDR: a kliens IP címe
- $REMOTE_USER: ha van **à** login neve
- $DOCUMENT_ROOT: A szerver fő dokumentum könyvtárának elérési útvonala.
- $HTTP_USER_AGENT: A kliensgép böngészője
- $PATH: PATH környezeti változó, ahol a rendszer a futtatható prg-ket keresi.
- $SERVER_SOFTWARE: A http szerver neve, verzió
- $SERVER_PROTOCOL: A szerver által használt protokoll és verziószáma. pl.: HTTP/1.0
- $SERVER_ADDR: Szerver IP címe.
- $SERVER_NAME, SERVER_ADMIN: Teljes host neve, admin e-mail címe.
- $GATEWAY_INTERFACE: A szerver által támogatott CGI protokoll és verziószám. pl.: CGI/1.1

```
<html>
<head>
<title>Űrlap</title>
</head>
<?php
$f1="alma";
$f2="banán";
$f3="citrom";
?>
<body>
<?php
foreach ($GLOBALS as $kulcs=>$ertek)
```

```
  {
    print "\$GLOBALS[\"$kulcs\"] ==$ertek<br>";
  }
?>
</body>
```

A PHP környezetéről, az Apache webszerverről, támogatott alrendszerekről (modulokról) és ezek állapotairól a phpinfo() függvénnyel kaphatunk tájékoztatást.

```
<?
  phpinfo();
?>
```

Néhány hasznos függvény:

- isset($változó): Ha a változóhoz már rendeltünk értéket, a függvény visszatérési értéke: true, egyébként false lesz.

Űrlapokról származó információk feldolgozása:

Form.html állomány:

```
<html>
<head>
<title>Ürlap</title>
</head>
<body>
<form action="urlap.php" method="post">
Név: <br>
<input type="text" name="f">
<br>
Cím:<br>
<textarea name ="cim" rows="5" cols="40"></textarea>
<br>
<input type="submit" value="OK!">
<input type="reset" value="Mégsem">
</form>
</body>
</html>
```

Az urlap.php fájl tartalma:

```
<html>
```

```
<head>
<title>Űrlap eredmény</title>
</head>
<body>
<?php
print $_POST[f];
print $_REQUEST[f];
print "Név: <b>$HTTP_POST_VARS[f]</b><br>";
//print "A címed: <P>\n\n<b>$_POST[cim]</b>";
print "A címed: <b>$cim</b>";
?>
</body>
</html>
```

Eredménye:

Név: **Ka Pál**
A címed: **Otthon**

Fájlkezelés:

- file_exists():

```
if (file_exists(”proba.txt”))
  print ”Létezik!”;
```

- is_file():

```
if ( is_file (”proba.txt”) );
  print ”fájl”;
```

- is_dir();

```
if ( is_dir (”proba”) );
  print ”könyvtár”;
```

- is_readable()
- is_writeable()
- is_executable()
- filesize()

print filesize(”proba.txt”);

- fileatime(); - utolsó megnyitás dátuma

```
$atime = filetime(”proba.txt”);
```

print date(”Y.m.d. H:i”,$atime); # 2005.02.27. 21:56

- filemtile(); - utolsó módosítás dátuma
- filectime(); - utolsó változás dátuma
- touch(); - fájl létrehozása, ha nem létezik. Ha már létezik, tartalma nem változik meg, de a módosítás dátuma a függvény végrehajtási idejére módosul.
  pl.: `touch(”\alma\file.txt”);`
- unlink(); - fájl törlése.

Fájlt törölni, írni, olvasni, módosítani csak akkor lehetséges, ha egy fájlon a megfelelő jogosultságokkal rendelkezünk.

- fopen() – file megnyitása. Két paramétere van: a fájl elérési útja, a megnyitási mód. ’r’ – olvasás, ’w’ – írás, ’a’ – hozzáfűzés: visszatérési értéke egy egész szám, ún. fájlazonosító. Ezt változóként tárolhatjuk, majd ezzel hivatkozhatunk a fájlra a későbbiekben.

```
$fa = fopen (”proba.txt”,’r’);
```

pl. 2.:

```
if ($fa = fopen (”proba.txt”,’w’) )
  {
  # $fa műveletek
  }
```
pl. 3.:
```
($fa = fopen(”proba.txt”, ’w’) ) or die
  (”A fájl nem nyitható meg!”);
```
Ha az fopen() true értékkel tér vissza, a die() nem hajtódik végre, kül. a kif. jobb oldalán a die() kiírja a paraméterben szereplő karakterláncot és megszakítja a program futását.

- fclose($fa): file lezárása.
- fgets(): Fájl tartalmának beolvasása, sorról-sorra. Két paramétere van, a fájl azonosító, egy egész szám. Utóbbi meghatározza, hogy legfeljebb hány bájtot olvasson ki a PHP, amíg sorvége vagy fájlvége jelet nem talál. Addig olvas, amíg újsor karakterhez (”\n”) nem ér, a megadott bájtnyi adatot ki nem olvassa, vagy a fájl végét el nem éri.

```
$sor = fgets( $fa, 1024);
```

- feof(): Fájl végére értünk-e. Visszatérési értéke true, ha a fájl végére értünk, különben false.

```
while ( ! feof($fa) )
  {
  $sor = fgets($fa,1024);
  print ”$sor<br>”;
  }
```

- fread(): tetszőleges mennyiségű, előre meghatározott méretű adat olvasása fájlból. Visszatérési értéke a megadott mennyiségű adat lesz, kivéve, ha elérte a fájl végét.

```
$reszlet = fread ( $fa, 16);
```

- fseek(): az olvasás helyének beállítása.

```
fseek( $fa, 64);
```

- fgetc($fa); - karakterenkénti olvasás.
- fwrite(); fputs(); - fájlba írás.

```
fwrite( $fa, ”Szia”);
```

- flock(); fájl zárolása. A második paraméter: 1- megosztott: más folyamatok olvashatják a fájlt, de nem írhatják. 2 – kizáró: más folyamatok nem olvashatják, nem írhatják. 3 – felszabadítás.

```
flock( $fa, 1);
```

- mkdir(): könyvtár létrehozása.

```
mkdir (”proba_konyvtar”,0777) # teljes jog
```

- rmdir (”proba_konyvtar”); # törlés
- opendir(); könyvtár megnyitása olvasásra.

```
$kvt = opendir (”konyvtar”);
```

- readdir(): könyvtár tartalmának olvasása.

```
$kvtnev = ”proba_dir”;
$kvt = opendir($kvtnev);
while (gettype ($fajl = readdir($kvt)) !=boolean)
  {
  if (is_dir(”$kvtnev/$fajl”))
     print ”(D)”;
  print ”$fajl<br>”
  }
closedir($kvt);
```

Objektumok:

```
class elso_osztaly
  {
   var $nev = ”alma”;
   function koszon()
     {
      print ”Üdv!”;
     }
  }
$obj1 = new elso_osztaly();
$obj ->koszon();
```