PHP ELÁGAZÁSOK

Elágazások nélkül az értelmező lépésről, lépésre végrehajtaná a PHP kódot, de nem tudná kezelni a felhasználók egyedi kéréseit. A végeredmény mindig ugyanaz lenne.

Az IF szerkezet

Az if kulcsszót követő zárójelben megadott feltétel teljesülése esetén, végrehajtásra kerül az azt követő kapcsos zárójelben elhelyezett kódblokk.

Az if utasítás szintaxisa:

Példa:
A kódblokk akkor kerül végrehajtásra, ha a feltétel igaz.

```
<body>
  <h2>IF utasítás példa 01</h2>
  <?php
    $v=2;
    if ($v>0) {echo $v; $v=0;}
    if ($v>0) {echo $v;}
  ?>
</body>
```

Eredmény:

IF utasítás példa 01

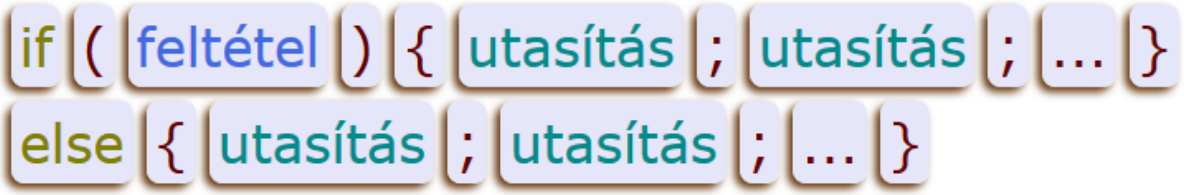
2

Az else záradék

Az if utasításban szereplő feltétel esetén végrehajtandó kódblokk mögé az else kulcsszót, és azt követően kapcsos zárójelek között egy újabb kódblokkot írhatunk.

A második kódblokk akkor lesz végrehajtva, ha a feltétel nem teljesül.

Az if else vezérlési szerkezet szintaxisa:



Példa:

Az első kódblokk, akkor kerül végrehajtásra, ha a feltétel igaz.
A második kódblokk pedig akkor, ha a feltétel hamis.

```
<body>
  <h2>Else záradék példa 01</h2>
  <?php
    $v=0;
    if    ($v>0) {echo $v; $v=0;}
    else {echo $v; $v=1;}
    echo ", ".$v;
  ?>
</body>
```

Eredmény:

Else záradék példa 01

0, 1

Az elseif záradék

Ha több feltétel közül egy-egy teljesülése esetén, külön kódot szeretnénk végrehajtani, akkor használhatjuk az elseif szerkezetet.

Az elseif vezérlési szerkezet szintaxisa:

```
if ( feltétel ) { utasítás ; utasítás ; ... }
elseif ( feltétel ) { utasítás ; utasítás ; ... }
else { utasítás ; utasítás ; ... }
```

Az első kódblokk, akkor kerül végrehajtásra, ha az 1. feltétel igaz.

A második kódblokk, akkor kerül végrehajtásra, ha az 1. feltétel nem igaz, de a 1. feltétel igaz.

A harmadik kódblok, akkor kerül végrehajtásra, ha egyik feltétel sem igaz.

Példa:

```
<body>
  <h2>Elseif záradék példa 01</h2>
  <?php
    $a=3; $b=1;
    if    ($a==$b) {echo "a=b";}
    elseif ($a>$b) {echo "a>b";}
    else  {echo "a<b";}
  ?>
</body>
```

Eredmény:

Elseif záradék példa 01

a>b

A switch szerkezet

Ha egy változó lehetséges értékeihez más-más PHP kódot kívánunk rendelni, akkor a switch utasítást célszerű használni. A break utasítás hatására a program kilép a switch szerkezetből.

A switch vezérlési szerkezet szintaxisa:

```
switch ( változó ) {
case 1. érték: { utasítás ; utasítás ; ... break ; }
case 2. érték: { utasítás ; utasítás ; ... break ; }
case n. érték: { utasítás ; utasítás ; ... break ; }
default { utasítás ; utasítás ; ... }
}
```

Az első kódblokk, akkor kerül végrehajtásra, ha a változó = 1.érték.

A második kódblokk, akkor kerül végrehajtásra, ha a változó = 2.érték.

Az n-edik kódblokk, akkor kerül végrehajtásra, ha a változó = n.érték.

A default kódblokk, akkor kerül végrehajtásra, ha a változó értéke nem egyezett meg egyetlen vizsgált értékkel sem, vagy a programozó elhagyta a break utasítást.

```php
<body>
  <h2>Switch szerkezet példa 01</h2>
  <?php
  $v = 3;
  switch ($v) {
   case 1:
     echo "Egy";
     break;
   case 2:
     echo "Kettő";
     break;
   case 3:
     echo "Három";
     break;
   default:
     echo "A változó értéke nem 1, 2 vagy 3";
  }
  ?>
</body>
```

Eredmény:

Switch szerkezet példa 01

Három

Alternatív szintaxis

A PHP nyelvben egyes vezérlési szerkezetek esetén (if, while, for, foreach, és switch) alternatív szintaxist is használhatunk. Az alternatív szintaxisnál a nyitó kapcsos zárójelet kettőspontot (:) helyettesíti. A záró zárójelet pedig a vezérlési szerkezetnek megfelelő endif;, endwhile;, endfor;, endforeach;, vagy endswitch;.

Az alternatív szintaxis megkönnyíti a HTML kód beillesztését.

A vezérlési szerkezetek szintaxisa:

```
<?php if ( feltétel ) : ?>
HTML kód
<?php endif ; ?>
```

Példa:

A naptárat tartalmazó DIV csak akkor kerül megjelenítésre, ha a $NaptarKell változó értéke igaz.

```
<body>
  <?php $NaptarKell = true; ?>
  <h2>Switch szerkezet alternatív példa </h2>
  <?php if ($NaptarKell == true): ?>
    <div id="naptar">Ide jön egy naptár</div>
  <?php endif; ?>
</body>
```

Eredmény:

Switch szerkezet alternatív példa

Ide jön egy naptár

PHP CIKLUSOK

Ha egy-egy kódblokkot többször egymás után végre kell hajtani, akkor elég csupán egyszer leírni. Megadjuk, hogy meddig kell ismételni és a többi az értelmező program dolga.

For ciklus

Akkor használjuk, ha előre tudjuk, hányszor kell a kódblokkot végrehajtani.

A for ciklus szintaxisa:

1. kifejezés

   Az első kifejezés a ciklusváltozónak ad kezdeti értékét.

2. kifejezés

   A második egy feltétel.

3. kifejezés

   A harmadik a ciklusváltozót növeli vagy csökkenti.

Ciklusmag

   PHP kódblokk, amelynek a végrehajtása mindaddig ismétlődik, amíg a második kifejezésben meghatározott feltétel teljesül.

A for ciklus elől tesztel. Ha a feltétel már kezdetben sem igaz, akkor a ciklusmag egyszer sem lesz végrehajtva.

Példa:

```
<body>
  <h2>For ciklus példa 1</h2>
  <?php
   for ($i=1; $i<=3; $i++) {
     echo ' $i='. $i;    // $i=1 $i=2 $i=3
   }
 ?>

  <h2>For ciklus példa 2 </h2>
  <?php
    for ($i=1; $i<3; $i++) {
    echo ' $i='. $i;     // $i=1 $i=2
  }
 ?>

  <h2>For ciklus példa 3 </h2>
  <?php
    for ($i=6; $i>3; $i--) {
    echo  ' $i='. $i;    // $i=6 $i=5 $i=4
  }
 ?>

</body>
```

Eredmény:

For ciklus példa 1

$i=1 $i=2 $i=3

For ciklus példa 2

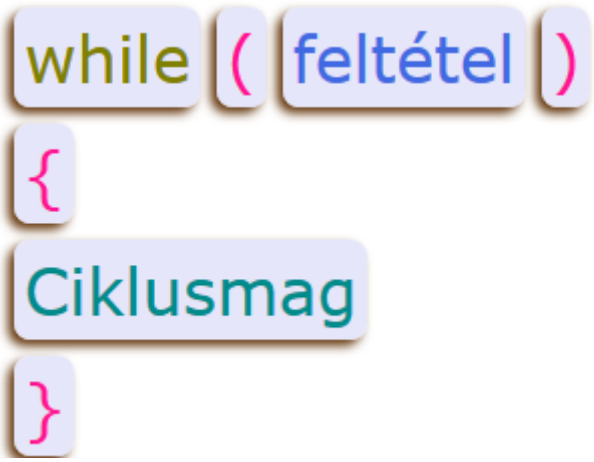
$i=1 $i=2

For ciklus példa 3

$i=6 $i=5 $i=4

While ciklus

A while ciklus szintaxisa:



Amíg a feltétel igaz, addig a ciklusmag (kódblokk) újra és újra végrehajtásra kerül.

Elöl tesztelő ciklus. Ha a feltétel már kezdetben sem teljesül, akkor a programkód egyszer sem lesz végrehajtva.

Ha a programkód sohasem biztosítja, hogy a feltétel ne teljesüljön, akkor végtelen ciklushoz jutunk.

Példa:

```
<body>
  <h2>While ciklus példa 1</h2>
  <?php
    $v = 1;
    while ($v < 5){
      echo $v;
      echo ", ";
      $v = $v + 1;       //1, 2, 3, 4,
    }
  ?>
  <h2>While ciklus példa 2 </h2>
  <?php
    $v = 1;
    while ($v < 5){
       echo ++$v;
       echo ", ";        //2, 3, 4, 5,
    }
  ?>
```

```
  <h2>While ciklus példa 3 </h2>
  <?php
    $v = 1;
    while ($v < 5){
      echo $v++;
      echo ", ";          //1, 2, 3, 4,
   }
   ?>
  <h2>While ciklus példa 4 </h2>
  <?php
     $v = 6;
     while ($v < 5){
        echo $v;
        echo ", ";
        $v = $v + 1;     //
    }
 ?>
</body>
```

Eredmény:

While ciklus példa 1

1, 2, 3, 4,

While ciklus példa 2

2, 3, 4, 5,

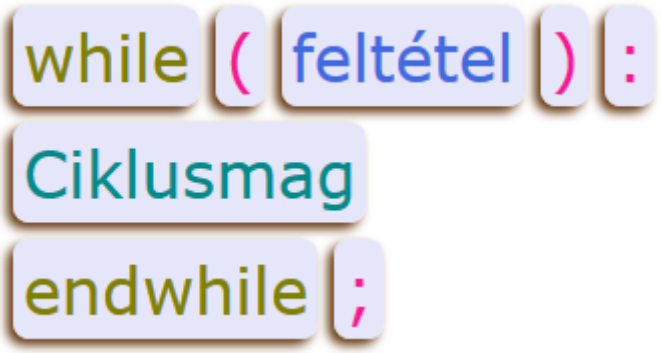
While ciklus példa 3

1, 2, 3, 4,

While ciklus példa 4

While alternatív megadási módja

While alternatív szintaxisa:



While alternatív szintaxisánál a nyitó utasítászárójelet ({) kettőspont, a zárót (}) pedig endwhile kúlcsszó helyettesíti.

Példa:

```
<body>
  <h2>While ciklus példa A1</h2>
  <?php
     $v = 1;
     while ($v < 5):
       echo $v;
       $v++;
     endwhile;
  ?>
</body>
```

Eredmény:

While ciklus példa A1

1234

Do-while ciklus

Do-while ciklus szintaxisa:

Amíg a feltétel igaz, addig a kódblokk újra és újra végrehajtásra kerül.

Hátul tesztelő ciklus. A ciklusmag egyszer mindig végrehajtásra kerül.

Ha a programkód sohasem biztosítja, hogy a feltétel ne teljesüljön, akkor végtelen ciklushoz jutunk.

Példa:

```
<body>
  <h2>Do-while ciklus példa 1</h2>
  <?php
    $v = 1;
    do {
       echo $v;
       echo ", ";
       $v = $v + 1;       //1, 2, 3, 4,
    } while ($v < 5)
  ?>
  <h2>Do-while ciklus példa 2</h2>
  <?php
    $v = 1;
    do {
       echo ++$v;
       echo ", ";         //2, 3, 4, 5,
  } while ($v < 5)
   ?>
  <h2>Do-while ciklus példa 3</h2>
  <?php
     $v = 1;
     do {
        echo $v++;
        echo ", ";        //1, 2, 3, 4,
    } while ($v < 5)
  ?>
```

```
  <h2>Do-while ciklus példa 4</h2>
  <?php
    $v = 6;
    do {
      echo $v;
      echo ", ";
      $v = $v + 1;              // 6,
    } while ($v < 5)
   ?>
</body>
```

Eredmény:

Do-while ciklus példa 1

1, 2, 3, 4,

Do-while ciklus példa 2

2, 3, 4, 5,

Do-while ciklus példa 3

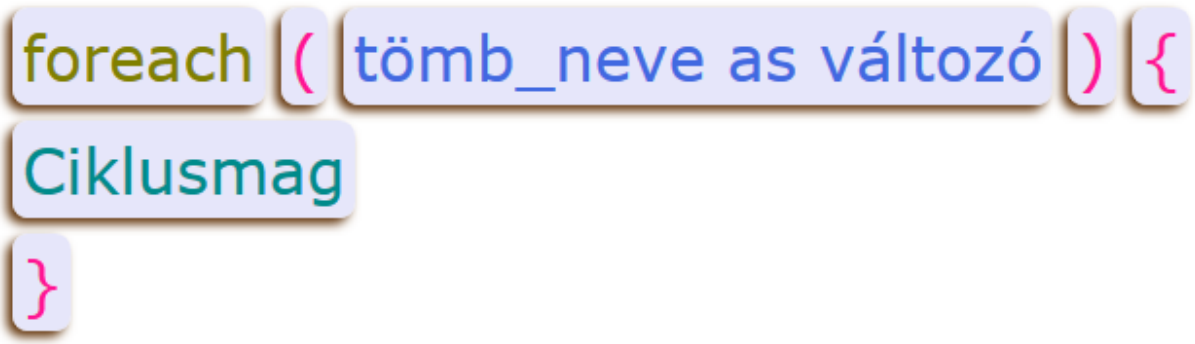
1, 2, 3, 4,

Do-while ciklus példa 4

6,

Foreach ciklus

Foreach ciklus szintaxisa:



Végigmegy a tömbön, és az egymást követő tömbelemeket teszi a ciklusmagon belül elérhetővé.

Először az első tömbelem értéke lesz a kódblokkon belül elérhető a változóban, majd a második… végül az utolsó.

Nem kel ismernünk a tömbelemek számát, és nem fordulhat elő, hogy nem létező tömbelemmel végzünk műveletet.

A leggyorsabb megoldást kínálja ismeretlen méretű tömbök kezelésére.

Példák:

```
<body>
  <h2>Foreach ciklus példa 1</h2>
  <?php
    $napok =array("hétfő","kedd","szerda","csütörtök",
                                "péntek","szombat","vasárnap");

    foreach ($napok as $v) {
      echo $v . ", ";
    }
  ?>
</body>
```

Eredmény:

Foreach ciklus példa 1

hétfő, kedd, szerda, csütörtök, péntek, szombat,
vasárnap,

```
<body>
  <h2>Foreach ciklus példa 2</h2>
  <?php
    $arr = array(1, 2, 3, 4);

    foreach ($arr as &$v) {      // & jel révén a $v változó címére hivatkozunk (referencia változóként)
      $v = $v + 1;                        // így a blokkban megadott értéke megmarad azon kívül is
    }
    echo '<br>$arr[0]: ' .$arr[0];
    echo '<br>$arr[1]: ' .$arr[1];
    echo '<br>$arr[2]: ' .$arr[2];
    echo '<br>$arr[3]: ' .$arr[3];
  ?>
</body>
```

Eredmény:

Foreach ciklus példa 2

$arr[0]: 2
$arr[1]: 3
$arr[2]: 4
$arr[3]: 5

Kilépés ciklusból

A break utasítás segítségével kiléphetünk az aktuális for, foreach, while, do-while ciklusból vagy switch szerkezetből.

A break opcionális paramétereként megadható, hogy hány szintet kell elhagyni. (Az egymásba ágyazott kódstruktúrában hány szintet lépünk fel.)

Példa:

```
<body>
  <h2>Kilépés ciklusból - break példa 1</h2>
  <?php
     for ($i=1; $i<=10; $i++) {
        echo ' $i='. $i;    // $i=1 $i=2 $i=3
        if ($i==3) {break;}
     }
  ?>

  <h2>Kilépés ciklusból - break példa 2</h2>
  <?php
     $i = 0;
     while (++$i) {
       switch ($i) {
         case 1:
           echo "Első kódblokk helye";
           break 1;  /* kilépés a switch-ből */
         case 2:
           echo "Második kódblokk helye";
           break 2;    /* kilépés a switch-ből és a while-ból is*/
         case 3:
           echo "Harmadik kódblokk helye";/*Ide már nem jut el.*/
           break;
         default:
           break;
       }
     }
   ?>
</body>
```

Eredmény:

Kilépés ciklusból - break példa 1

$i=1 $i=2 $i=3

Kilépés ciklusból - break példa 2

Első kódblokk helyeMásodik kódblokk helye

A ciklusmag átugrása

A ciklusmag hátra lévő részének átugrása.

A continue segítségével átugorhatjuk a ciklusmag continue után álló utasításait. A kód végrehajtása a következő iterációval (a ciklusmag következő végrehajtásával) folytatódik, a ciklusfeltétel teljesülése esetén.

Példa:

```
<body>
  <h2>Continue példa 1</h2>
  <?php

  for ($v = 1; $v <= 4; ++$v) {
      if ($v == 2) {continue;}
      echo $v. ", ";  // 1,3,4,
  }

 ?>
</body>
```

Eredmény:

Continue példa 1

1, 3, 4,

PHP FÜGGVÉNYEK

Ha egy-egy kódblokk végrehajtására a program több pontján is szükség lehet, akkor célszerű függvényként megadni.

A függvények

A függvények olyan kódrészletek, amelyek akkor kerülnek végrehajtásra, ha a programkódból meghívjuk őket.

Lehetővé teszik, hogy a programon belül többször (akár több ezerszer) végrehajtásra kerülő kódot csak egyszer kelljen elkészíteni. Ezt követően a program bármely részéről meghívható.

A függvények használatával rövidebb, áttekinthetőbb és így könnyebben karbantartható kód készíthető.

Egy függvény meghívásához csupán a nevét kel elhelyezni a kódban. Ha a végrehajtásához szükség van paraméterek átadására, akkor azok nevét a függvénynév után zárójelben, vesszővel elválasztva soroljuk fel. Ha nincs ilyen paraméter, akkor a zárójel üres marad.

Példa:

megj. A strtolower() függvény a magyar ékezetes karaktereket hibásan kezeli.

```
<body>

  <?php
    $str="NAGYBETŰS VOLTAM";
    $str=strtolower( $str );
    echo $str;
  ?>

</body>
```

Eredmény:

nagybetŰs voltam

Függvények készítése

A függvényeket a function kulcsszóval vezetjük be. Egy függvényt a neve alapján azonosít az értelmező.

A függvény neve:

Az angol ABC betűiből és számjegyekből állhat, de nem kezdődhet számjeggyel!

Egy szóból kell álljon (több szó esetén a szóköz helyett használhatunk aláhúzás karaktert)!

A PHP a függvények esetén nem tesz különbséget kis és nagybetűk között

Használjunk beszélő neveket (a név utaljon a funkcióra)! Saját életünket könnyítjük meg így.

Paraméterek:

A függvények számára, a függvénynevet követő zárójelben átadhatunk paramétereket. Ha egy függvény több paramétert igényel azokat vesszővel elválasztva soroljuk fel. Ha egyet sem, akkor a zárójel üresen marad.

Alapesetben az átadott változók lokális változókként használhatók. A függvény lefutását követően a futás során kapott új értékük elveszik. Ezt nevezzük érték szerinti paraméter átadásnak.

A kódblokk tartalmazza a függvények meghívásakor végrehajtásra kerülő kódot.

Függvények szintaxisa:

```
function fuggveny_neve ( $parameter1, $parameter1, ... )
{
kódblokk
}
```

Példa:

```
<body>

  <?php
    function osszegki( $a, $b)   //  függvény feje
    {
      echo   $a + $b;     //  függvény törzse
    }
    osszegki(1, 1);    // 2
  ?>

</body>
```

Eredmény: 2

A függvény visszatérési értéke

A PHP nyelvben minden függvény rendelkezik visszatéréssel, amely segítségével meghatározható, hogy a függvény rendben lefutott, vagy hiba történt, illetve a visszatérési értékkel kaphatjuk meg a függvény által végzett műveletek eredményét.

Saját készítésű függvények esetén a visszatérési érték a return kulcsszóval adható meg.

Ha nem adunk meg visszatérési értéket, akkor explicit módon, az alapértelmezett NULL értékkel tér vissza a függvény.

Függvények szintaxisa:

```
function fuggveny_neve ( $parameter1, $parameter1, ... )
{
kódblokk
return érték;
}
```

Példák:

```
<body>

  <?php

    function osszeg( $a, $b)
    {
      return   $a + $b;     //  függvény törzse
    }
    echo osszeg(1, 1);    // 2
  ?>

</body>
```

Eredmény: 2

```
<body>

  <?php
    function osszeg( $a, $b)
    {
       $a = $a + $b;    // a függvény törzse
    }
    $vissza = osszeg(1, 1);
    var_dump($vissza);     // NULL
  ?>

</body>
```

Eredmény: NULL

A változók hatóköre (scope)

A változók hatóköre megadja, hogy értékük hol érhető el.

A globális változók, amelyeket a PHP kódunk bármely pontján, de nem egy felhasználói függvényen belül hoztunk létre, a létrehozást követően bárhonnan közvetlenül elérhetők. hatáskörük kiterjed az include vagy require segítségével használt fájlokra is. Függvényeinken belül azonban nem elérhetők.

A felhasználói függvényen belül deklarált lokális változókkal fordított a helyzet. Ezek csak az adott függvényen belül használhatók.

Példa:

```
<body>

  <?php
    $v="globális változó";
    function Teszt() {
      $v="lokális változó";
      echo $v;  // A lokális változó írja ki!
    }
    Teszt();//  => lokális változó
    echo "<br> $v";  //  => globális változó
  ?>

</body>
```

Eredmény:

lokális változó
globális változó

A global kulcsszó

Ha egy függvényben globális változókat szeretnénk használni global kulcsszóval kell deklarálni őket.

Példák:

```
<body>

  <?php
    $v1=1;
    $v2=2;
    function Teszt() {
      global $v1;
      echo $v1;          // => 1
      echo $v2;          // =>
    }
    Teszt();  // $v2 nem elérhető a függvényen belül!
  ?>

</body>
```

Eredmény: 1

```
<body>

  <?php
    $v1=1;
    $v2=2;
    function Teszt() {
      global $v1, $v2;
      $v1 = $v1 + $v2;
      echo $v1;  // => 3
    }
    Teszt();
    echo "<br> $v1";  // => 3
  ?>

</body>
```

Eredmény:

3
3

GLOBALS tömb használata

A PHP által definiált speciális $GLOBALS tömb használatával szintén elérhetők a globális változók.

Példa:

```
<body>

  <?php
    $v1=1;
    $v2=2;
    function Teszt() {
      $GLOBALS["v1"] = $GLOBALS["v2"] + $GLOBALS["v1"];
    }
    Teszt();
    echo $v1;   // => 3
  ?>

</body>
```

Eredmény: 3

A static kulcsszó

A függvények futása után rendszerint a lokális változók törlődnek. Az értelmező felszabadítja a számukra lefoglalt helyet.

A static kulcsszóval deklarált változók nem törlődnek és értéküket is megőrzik két függvényhívás között.

Példa:

```
<body>

  <?php
    function Teszt() {
      static $v=1;
      echo "<br> $v";
      $v = $v  + 1;
    }
    Teszt(); // => 1
    Teszt(); // => 2
    Teszt(); // => 3
  ?>

</body>
```

Eredmény:

1
2
3

Paraméterátadás

A függvénynek átadott paramétereket alapértelmezetten lokális változóként lehet használni.

Példa:

```
<body>

  <?php
    function Teszt($v) {
      $v = $v + 1;
      echo $v; // => 11
    }
    $v = 10;
    Teszt($v);
    echo '<br> $v:'.$v;
  ?>

</body>
```

Eredmény:

11
$v:10

PHP MATEMATIKAI FÜGGVÉNYEI

Megismertük a változók és operátorok használatát. A matematikai műveletek megoldásához a PHP függvényeket is kínál.

Abszolút érték

A abs() függvény a paraméterként kapott számérték abszolút értékével tér vissza.

A abs() függvény szintaxisa:

Paraméter: float vagy integer.

A visszatérési érték: float vagy integer a paraméter típusától függően.

Példa:

```
<body>

<?php

var_dump(abs(12.3)); echo  "<br>";
var_dump(abs(-123)); echo  "<br><br>";

$v = 123;
settype ( $v,  "integer" );
var_dump(abs($v));echo  "<br>";
settype ( $v,  "float" );
var_dump(abs($v));

?>

</body>
```

Eredmény:

```
float(12.3)
int(123)

int(123)
float(123)
```

Kerekítés

A ceil() függvény a paraméterként kapott számértéket felfelé kerekíti a legközelebbi egész számra.

A floor() függvény a paraméterként kapott számértéket lefelé kerekíti a legközelebbi egész számra.

A round() függvény a paraméterként kapott számértéket megadott pontossággal kerekíti.

A ceil() függvény szintaxisa:

A floor() függvény szintaxisa:

A round() függvény szintaxisa:

Paraméter: float vagy integer.

Paraméter (a round esetén): integer a pontosságot határozza meg.

A visszatérési érték: float vagy integer

Példa:

```
<body>

<?php
$v1 = 123.456789;
$v2 = 9.87654321;

echo "ceil($v1) = ".ceil($v1) . "<br>";
echo "ceil($v2) = ".ceil($v2) . "<br>";
echo "floor($v1) = ".floor($v1) . "<br>";
echo "floor($v2) = ".floor($v2) . "<br>";
echo "round($v1) = ".round($v1) . "<br>";
echo "round($v2) = ".round($v2) . "<br>";
echo "round($v1,2) = ".round($v1,2) . "<br>";
echo "round($v2,2) = ".round($v2,2) . "<br>";
?>

</body>
```

Eredmény:

ceil(123.456789) = 124
ceil(9.87654321) = 10
floor(123.456789) = 123
floor(9.87654321) = 9
round(123.456789) = 123
round(9.87654321) = 10
round(123.456789,2) = 123.46
round(9.87654321,2) = 9.88

Hatványozás

A pow() függvény az első paraméterben megadott számot a második paraméterben megadott kitevőre emeli.

A pow() függvény szintaxisa:

```
integer/float = pow ( paraméter , paraméter ) ;
```

1. paraméter: alapszám.

2. paraméter: kitevő.

A visszatérési érték: float vagy integer

Példa:

```
<body>

<?php
echo ' <br><b> pow(10,2): </b><br>';
var_dump(pow(10,2));
echo ' <br><b> pow(2,8): </b><br>';
var_dump(pow(2,8));
echo ' <br><b> pow(3.333,8): </b><br>';
var_dump(pow(3.333,8));
echo ' <br><b> pow(-2,8): </b><br>';
var_dump(pow(-2,8));
echo ' <br><b> pow(2,-8): </b><br>';
var_dump(pow(2,-8));
echo "<br>";
?>

</body>
```

Eredmény:

pow(10,2):
int(100)
pow(2,8):
int(256)
pow(3.333,8):
float(15229.390031154)
pow(-2,8):
int(256)
pow(2,-8):
float(0.00390625)

Gyökvonás

A sqrt() függvény a paraméterben megadott szám négyzetgyökével tér vissza.

A sqrt() függvény szintaxisa:

Paraméter: poztitív szám.

A visszatérési érték: float

Példa:

```
<body>

<?php
echo ' <br><b> sqrt(8): </b><br>';
var_dump(sqrt(8));
echo ' <br><b> sqrt(64): </b><br>';
var_dump(sqrt(64));
echo ' <br><b> sqrt(0.64): </b><br>';
var_dump(sqrt(0.64));
echo ' <br><b> sqrt(-64): </b><br>';
var_dump(sqrt(-64));
?>

</body>
```

Eredmény:

sqrt(8):
float(2.8284271247462)
sqrt(64):
float(8)
sqrt(0.64):
float(0.8)
sqrt(-64):
float(NAN)

Véletlen számok

A srand() függvény beállítja a véletlenszám-generátor kezdőértékét.

A rand() függvény egy véletlen számmal tér vissza.

A getrandmax() függvény visszaadja a véletlenszám-generátor által előállítható legnagyobb értéket.

A srand() függvény szintaxisa:

Csak egyszer kell meghívni!

Paraméter (opcionális): pozitív szám.

A visszatérési érték: nincs (void)

A rand() függvény szintaxisa:

1. paraméter (opcionális): véletlenszám legkisebb értéke (minimum)

2. paraméter (opcionális): véletlenszám legnagyobb értéke (maximum)

A visszatérési érték: Integer véletlen számérték. Ha a minimum és maximum értéke meg lett adva, akkor a két érték közé esik.

A getrandmax() függvény szintaxisa:

Paraméter: nincs

A visszatérési érték: Integer: a véletlenszám-generátor által előállítható legnagyobb érték.

Példa: <body>

```
<?php
srand(); //véletlenszám generátor kezdőértékének beállítása
echo "Véletlen szám maximális értéke: ".getrandmax()."<br>" ;
echo "Véletlen szám: ".rand()."<br>" ;

echo "Véletlen szám: ".rand(10,55)."<br>" ;
echo "Véletlen szám: ".rand(5,10)."<br>" ;
?>

</body>
```

Eredmény:

Véletlen szám maximális értéke: 2147483647
Véletlen szám: 1369615746
Véletlen szám: 52
Véletlen szám: 7

Maximum kiválasztása

A max() függvény a paraméterekben kapott értékek közül kiválasztja a legnagyobbat.

A max() függvény szintaxisa:

```
integer/float = max ( 1. paraméter , 2. paraméter , ... ) ;
```

1. paraméter: szám

2. paraméter: szám

3. paraméter (opcionális): szám

n. paraméter (opcionális): szám

A visszatérési érték: integer/float a legnagyobb számérték.

A max() függvény nem csak számokkal használható.

Példa:

```
<body>

<?php
echo  '<b>max(3,4)</b><br>';
var_dump(max(3,4)); echo  "<br>";
echo  '<b>max(3,-4)</b><br>';
var_dump(max(3,-4)); echo  "<br>";
echo  '<b>max(3.333,3.334)</b><br>';
var_dump(max(3.333,3.334)); echo  "<br>";
echo  '<b>max(3,4,6,9,1,0.2)</b><br>';
var_dump(max(3,4,6,9,1,0.2)); echo  "<br>";
?>

</body>
```

Eredmény:

max(3,4)
int(4)
max(3,-4)
int(3)
max(3.333,3.334)
float(3.334)
max(3,4,6,9,1,0.2)
int(9)

Minimum kiválasztása

A min() függvény a paraméterekben kapott értékek közül kiválasztja a legkisebbet.

A min() függvény szintaxisa:

```
integer/float = min ( 1. paraméter , 2. paraméter , ... ) ;
```

1. paraméter: szám

2. paraméter: szám

3. paraméter (opcionális): szám

n. paraméter (opcionális): szám

A visszatérési érték: integer/float a legkisebb számérték.

A min() függvény nem csak számokkal használható.

Példa:

```
<body>

<?php
echo  '<b>min(3,4)</b><br>';
var_dump(min(3,4)); echo  "<br>";
echo  '<b>min(3,-4)</b><br>';
var_dump(min(3,-4)); echo  "<br>";
echo  '<b>min(3.333,3.334)</b><br>';
var_dump(min(3.333,3.334)); echo  "<br>";
echo  '<b>min(3,4,6,9,1,0.2)</b><br>';
var_dump(min(3,4,6,9,1,0.2)); echo  "<br>";
?>

</body>
```

Eredmény:

min(3,4)
int(3)
min(3,-4)
int(-4)
min(3.333,3.334)
float(3.333)
min(3,4,6,9,1,0.2)
float(0.2)