WEBTERVEZÉS – GYAKORLATI JEGYZET

7. gyakorlat

A PHP nyelvi alapjai

Készítették:

Cservenák Bence

Farkas Anikó

Savanya Sándor

FIGYELEM!

A jegyzet folyamatosan készül, így előfordulhatnak benne apróbb hibák, hiányosságok, elírások. Ha valaki esetleg ilyet találna, kérem, írjon a `Cservenak.Bence@stud.u-szeged.hu` címre, hogy mihamarabb javíthassuk.

MEGJEGYZÉS

A jegyzetben szereplő kódok esetén helytakarékossági célokból nem írjuk ki a `<?php` és `?>` tageket. Természetesen ahhoz, hogy a kód PHP kódként legyen értelmezve, `<?php` és `?>` közé kell azt tenni.

1. Kommentek

PHP-ban is lehetőségünk van a kódba kommenteket (megjegyzéseket) írni.

- Az egysoros kommenteket `//` vagy `#` után írjuk
- A több soros kommenteket `/*` és `*/` között adjuk meg

```
// ez egy egysoros komment
 # ez szintén egy egysoros komment

/* ez egy több
   soros komment */
```

2. Kiíratás

- **`echo`**`(…)`, **`print`**`(…)`: kiírják a paraméterül kapott kifejezést
  - zárójelek nélkül is működnek
  - HTML markup is használható (pl. sortörés a `"<br/>"` segítségével lehetséges)
  - az `echo`-nak több paramétere is lehet (vesszővel elválasztva)
- **`print_r`**`(…)`: rekurzívan kiírja a paraméterül kapott kifejezést
  - általában tömbök és objektumok kiíratására használjuk
- **`die`**`()`, **`exit`**`()`: kilépés a programból
  - átadható nekik string (hibaüzenet) vagy egész szám (hibakód)

3. Változók

- Létrehozás: `$valtozonev;`
  - a kódban a változók neve előtt **minden esetben** dollárjel (`$`) szerepel
  - a változónév csak betűket, számokat és alulvonást tartalmazhat, és betűvel vagy alulvonással kell kezdődnie
  - a változónév érzékeny a kis- és nagybetűkre (tehát pl. `$foo` és `$FOO` különböző változók)
- Az értékadás az = operátorral történik
  - alapértelmezett az **érték szerinti értékadás**
    - ezután a változók értékei egymástól függetlenül módosíthatók
  - az **`&`** operátorral lehetőségünk van **referencia szerinti értékadás**ra is
    - ezután ha az egyik változó értéke módosul, ugyanúgy módosul a másiké is

```
// érték szerinti értékadás

$a = 1;
$b = $a;     // $a: 1, $b: 1

$a = 2;      // $a: 2, $b: 1
$b = 3;      // $a: 2, $b: 3
```

```
// referencia szerinti értékadás

$a = 1;
$b = &$a;    // $a: 1, $b: 1

$a = 2;      // $a: 2, $b: 2
$b = 3;      // $a: 3, $b: 3
```

4. Gyengén típusosság

A PHP egy **gyengén típusos nyelv**.

- Ez azt jelenti, hogy PHP automatikusan "kitalálja" a változó típusát a benne tárolt érték alapján
- Egyik következménye, hogy a változók létrehozásakor nem írjuk ki expliciten a típust

```
$val = 42;   // ugyanez C-ben: int val = 42;
```

- Másik következménye, hogy ugyanaz a változó más-más típusú értékeket is tárolhat

```
$val = 42;       // val típusa: integer
$val = true;     // val típusa: boolean
$val = "sajt";    // val típusa: string
```

5. Konstansok (Állandók)

- A változókkal ellentétben kezdőértékük nem változtatható meg
- Elnevezés
  - a konstansok neve előtt **nem** szerepel `$`
  - konvenció: a konstans nevét csupa nagybetűvel írjuk, és ha az több szóból áll, akkor a szavakat alulvonással választjuk el (pl. `EZ_EGY_JO_KONSTANSNEV`)
- Létrehozás: `define("KONSTANS_NEVE", ertek, [(bool) case_insensitive]);`
  - ha a harmadik (opcionális) paraméter értékét `true`-ra állítjuk, akkor a konstans neve nem lesz érzékeny a kis- és nagybetűkre (alapból `false`, tehát érzékeny)

```
// konstansok létrehozása
define("NEV", "Józsi");
define("ELETKOR", 21, TRUE);

// ELETKOR = 42;  // hiba: konstans értéke nem módosítható
// echo nev;      // hiba: a NEV érzékeny a kis- és nagybetűkre

echo NEV . " egy " . eLeTkOr . " éves hallgató.";
```

```
Józsi egy 21 éves hallgató.
```

- **Mágikus konstansok:** előre definiált, speciális konstansok
  - szintaxis: dupla alulvonással kezdődnek és végződnek
  - néhány fontosabb mágikus konstans
    - **__LINE__**: a fájl aktuális sorának száma
    - **__FILE__**: a fájl teljes elérési útvonala
    - **__CLASS__**: az osztály neve
    - **__METHOD__**: az osztály metódusának neve
    - **__NAMESPACE__**: a névtér neve

6. Adattípusok

A. Boolean – Logikai adattípus

- Lehetséges értékei: **`TRUE`** vagy **`FALSE`** (ezek nem érzékenyek a kis- és nagybetűkre)

B. Integer – Egész szám

- Az értelmezési tartományán kívüli értékek automatikus lebegőpontosra konvertálódnak
- Mindig előjeles (tehát nincs előjeltelen egész)
- Az osztás művelete (`$num1 / $num2`) mindig valós osztást végez a számokon
  - egészosztáshoz az **`intdiv`**`($num1, $num2)` függvényt használjuk

C. Float – Lebegőpontos szám

- Nincs külön egyszeres és dupla pontosságú lebegőpontos típus
- Tizedesvessző helyett tizedespontot használunk (pl. `3.14`)

D. String – Szöveg

- Leggyakrabban aposztrófok ('…') vagy idézőjelek ("…") között adjuk meg
  - különbség: az idézőjeles megadás esetén a változók értékei behelyettesítődnek, míg az aposztrófos megadás esetén nem
  - több soros stringeknél használatos még a heredoc és nowdoc megadási mód is

```php
$s = "szeretem a webtervet";
echo '$s értéke: ';    // nem helyettesíti be
echo "$s <br/>";       // behelyettesíti
```

```
$s értéke: szeretem a webtervet
```

- A stringek összefűzésére a `.` (pont) operátort használjuk

```php
$s = "almás" . "pite";
echo $s;
```

```
almáspite
```

- A karakterek indexelése a már jól ismert tömbindex operátorral történik
  - a karaktereket minden esetben 0-tól kezdve indexeljük
  - negatív index esetén a string végéről kezd el számolni (utolsó karakter indexe: `-1`)

```php
$s = "Hello World";
echo $s[0] . "<br/>";    // első karakter
echo $s[-1]. "<br/>";    // utolsó karakter
```

```
H
d
```

- Rengeteg beépített stringkezelő függvény
  - **strlen**($s): visszaadja az $s string hosszát
  - **strtolower**($s): csupa kisbetűssé alakítja az $s stringet
  - **strtoupper**($s): csupa nagybetűssé alakítja az $s stringet
  - **substr**($s, $start, $length): visszaadja az $s string részstringjét, annak $start. indexétől kezdődően, $length karakteren keresztül
  - **strpos**($s, $text): visszaadja a $text részstring kezdőindexét az $s stringben
    - **false**-ot ad vissza, ha nem találja
  - **str_replace**($old, $new, $s): az $s stringben az összes $old részstringet $new-ra cseréli
  - **strrev**($s): megfordítja az $s stringet
  - …

```php
$s = "A Citromos Fagyi A Legjobb Fagyi";

echo strlen($s) . "<br/>";                  // string hossza
echo strtolower($s) . "<br/>";              // kisbetűsítés
echo strtoupper($s) . "<br/>";              // nagybetűsítés
echo substr($s, 0, 8) . "<br/>";            // az első 8 karakter
echo strpos($s, "Citrom") . "<br/>";        // keresés
echo str_replace("Fagyi", "Süti", $s);      // csere
```

```
32
a citromos fagyi a legjobb fagyi
A CITROMOS FAGYI A LEGJOBB FAGYI
A Citrom
2
A Citromos Süti A Legjobb Süti
```

E. Array – Tömb

- Több (akár eltérő típusú) elem tárolására alkalmas, dinamikus méretű adatszerkezet
- Tulajdonképpen egy rendezett leképezést valósít meg → **kulcs-érték párok**at tartalmaz
  - a kulcs csak string vagy integer típusú lehet, az érték típusára nincs megkötés
  - ha nem adunk meg kulcsokat, akkor azok a tömbindexek lesznek (0-tól kezdődően)
- Létrehozás (2 módszer):
  - `$tomb = [kulcs1 => érték1, kulcs2 => érték2, …];`
  - `$tomb = array(kulcs1 => érték1, kulcs2 => érték2, …);`
- Elemek indexelése: `$tomb_neve[kulcs]`
- Új kulcs-érték pár hozzáfűzése a tömb végéhez: `$tomb_neve[kulcs] = ertek`

- Rengeteg beépített tömbkezelő függvény
  - `count($t)`: visszaadja a `$t` tömb hosszát
  - `sort($t)`: rendezi a `$t` tömböt
  - `array_pop($t)`: törli a `$t` tömb utolsó elemét
  - `array_push($t, $a, $b, …)`: beszúrja az `$a`, `$b`, … elemeket a `$t` tömb végére
  - `array_shift($t)`: törli a `$t` tömb első elemét
  - `array_unshift($t, $a, $b, …)`: beszúrja az `$a`, `$b`, … elemeket a `$t` tömb elejére
  - …

```php
// tömbök létrehozása
$jegyek = ["Sanyi" => 5, "Jóska" => 2, "Benedek" => 4];
$lottoszamok = [56, 42, 12, 81, 25];

array_pop($lottoszamok);                // törlés a végéről
array_push($lottoszamok, 18, 67);       // beszúrás a végére
array_shift($lottoszamok);              // törlés az elejéről
sort($lottoszamok);                     // rendezés
echo count($lottoszamok) . "<br/>";     // tömb hossza
print_r($lottoszamok);
```

```
5
Array ( [0] => 12 [1] => 18 [2] => 42 [3] => 67 [4] => 81 )
```

- Többdimenziós tömbök is létrehozhatók, ezek lényegében tömbön belüli tömbök

```php
$users = [
   ["username" => "admin", "password" => "Meow123"],
   ["username" => "guest", "password" => "Guest01"]
];

$users[0]["username"];       // admin
```

F. Object – Objektum típus

- Egy osztály példánya *(bővebben: lásd objektumorientált programozás)*

G. Resource – Erőforrás típus

- Általában egy fájlra vagy adatbázis-kapcsolatra mutató referencia

H. Null típus

- Egyetlen lehetséges értéke: **NULL**
- `isset($val)`: igazat ad vissza, ha `$val` értéke **nem** **NULL**
  - tipikusan űrlapfeldolgozásnál használjuk

7. Típuskonverzió, típusellenőrzés

- PHP-ban az adattípusok közötti konverzió automatikusan történik

```
"5" . 7;                // "5" . "7" = "57"
"5" + 7;                // 5 + 7 = 12
"5" + 7.3;              // 5.0 + 7.3 = 12.3
5 + "7 törpe";          // 12, de figyelmeztetést kapunk
```

- Természetesen PHP-ban is létezik a más nyelvekből ismerős **típuskényszerítés (typecasting)**
  - szintaxisa: `(céltípus)érték`

```
(int)false;         // 0
(int)3.14;          // 3
(string)42;         // "42"
```

- Az **`is_`** kezdetű beépített függvényekkel lekérdezhető, hogy egy érték milyen típusú
  - fontosabb függvények: **`is_bool`**(…), **`is_int`**(…), **`is_float`**(…), **`is_string`**(…), **`is_array`**(…), **`is_object`**(…), **`is_resource`**(…), **`is_null`**(…)
- A **`var_dump`**`($val)` függvénnyel rekurzívan kiírathatjuk a `$val` típusát és értékét

```
$szoveg = "sajt";
$tomb = [10, 20, 30];

var_dump($szoveg);
var_dump($tomb);
var_dump(is_array($tomb));
```

```
string(4) "alma"

array(3) {[0]=> int(10) [1]=> int(20) [2]=> int(30)}

bool(true)
```

8. Fontosabb operátorok

- **Aritmetikai operátorok:** + (összeadás), – (kivonás), * (szorzás), / (valós osztás), % (maradékos osztás), ** (hatványozás)
- **Hozzárendelő operátorok:** =, +=, -=, *=, /= (ugyanúgy működnek, mint C-ben)
- **Inkrementáló, dekrementáló operátorok:** ++, -- (ugyanúgy működnek, mint C-ben)
- **Összehasonlító operátorok:** == (egyenlő), === (azonos érték és típus), != (nem egyenlő), !== (eltérő érték vagy típus), < (kisebb), <= (kisebb vagy egyenlő), > (nagyobb), >= (nagyobb vagy egyenlő)
- **Logikai operátorok:** && (logikai ÉS), || (logikai VAGY), ! (logikai NEM)

Példa: Szemléltessük egy példán az == és === operátorok működésbeli különbségét!

```
$a = 0;
$b = false;

var_dump($a == $b);     // true
var_dump($a === $b);    // false
```

Magyarázat: Mivel a `0` egész érték a `false` logikai értéknek felel meg, ezért az `==` operátor igazat fog visszaadni. Viszont mivel az egyik változó típusa `integer`, míg a másiké `boolean`, ezért az `===` operátorral végzett összehasonlítás eredménye hamis lesz.

9. Vezérlési szerkezetek

A. Szelekciós vezérlés

Az **if**, **else if** (vagy **elseif**), **else** és **switch** utasítások ugyanúgy működnek, mint C-ben

```php
$num = 5;

if ($num < 0) {      // if, else if, else példa
   echo "Negatív <br/>";
} elseif ($num === 0) {
   echo "Nulla <br/>";
} else {
   echo "Pozitív <br/>";
}

switch ($num) {     // switch példa
   case 1: echo "Elégtelen <br/>"; break;
   // ...
   case 5: echo "Jeles <br/>"; break;
   default: echo "Hibás érték! <br/>";
}
```

```
Pozitív
Jeles
```

B. Ismétléses vezérlés (Ciklusok)

- A **while**, **do… while** és **for** utasítások ugyanúgy működnek, mint C-ben
  - **Példa:** Írassuk ki az egész számokat 1-től 10-ig mindhárom ciklussal!

```php
$i = 1;
while ($i <= 10) {
  echo $i . "<br/>";
  $i++;
}
```

```php
$i = 1;
do {
  echo $i . "<br/>";
  $i++;
} while ($i <= 10);
```

```php
for ($i = 1; $i <= 10; $i++) {
   echo $i . "<br/>";
}
```

- Külön érdemes viszont beszélni a **foreach** utasításról
  - általában tömbök és objektumok listaszerű bejárására használjuk

```php
$tomb = ["a" => 1, "b" => 10, "c" => 100];
foreach ($tomb as $e) {      // $e felveszi a tömb értékeit
   echo $e . "<br/>";
}
foreach ($tomb as $k => $e) {   // $k: kulcsok, $e: értékek
   echo "$k értéke: $e <br/>";
}
```

```
1
10
100

a értéke: 1
b értéke: 10
c értéke: 100
```

  - alapértelmezett módon csak a tömb/objektum másolatát kapja meg
    - ha az eredeti tömbbel/objektummal szeretnénk dolgozni, referenciát adunk át
    - példa a következő oldalon

```php
$tomb = [1, 2];
foreach ($tomb as $e) {    // nem módosítja az eredetit
    $e = 0;
}
print_r($tomb);
foreach ($tomb as &$e) {   // módosítja az eredetit
    $e = 0;
}
print_r($tomb);
```

```
Array ([0] => 1 [1] => 2)
Array ([0] => 0 [1] => 0)
```

10. Függvények

- Függvénydefiníció:

```php
function fuggveny_neve($param1, $param2, …) {
    // függvény törzse
}
```

- Függvényhívás:

```php
fuggveny_neve($param1, $param2, …);
```

  - a globális scope-ban létrehozott függvények már a definíciójuk előtt is hívhatók
- PHP-ban **nem lehet** a kódban két azonos nevű függvény (még eltérő paraméterezés esetén sem!)
- Változók hatásköre (scope)
  - a fájlban létrehozott változók a globális scope-ba kerülnek, míg a függvényen belüli változók lokálisak lesznek a függvény törzsében (azaz kívülről nem lehet őket elérni)
  - PHP-ban a függvényen kívüli globális változók nem használhatók a függvény törzsében

```php
$a = 5;
function foo() {
    // $a innen nem érhető el!
}
```

  - a függvényen kívüli változókat 2-féle módon érhetjük el a függvény törzsében

```php
// 1. módszer
$a = 5;
function foo() {
    global $a;
    $a = 10;
}
foo();    // $a értéke: 10
```

```php
// 2. módszer
$a = 5;
function foo() {
    $GLOBALS['a'] = 10;
}
foo();    // $a értéke: 10
```

- a `static` kulcsszóval létrehozhatunk **statikus változók**at is
  - a létrehozó blokkon (*itt:* függvényen) belül lokális, azon kívül pedig megőrzi értékét

```
function szamol() {
   static $count = 0;    // statikus változó
   $count++;
   echo "Ez a(z) $count. függvényhívás <br/>";
}

szamol();
szamol();
szamol();
```

```
Ez a(z) 1. függvényhívás
Ez a(z) 2. függvényhívás
Ez a(z) 3. függvényhívás
```

- Paraméterátadás
  - alapértelmezett módon **érték szerinti paraméterátadás** történik
    - ekkor az átadott változónak csupán egy másolatával dolgozunk
  - lehetőségünk van viszont **referencia szerinti paraméterátadás**ra is
    - ekkor közvetlenül az átadott változóval dolgozunk

```
function modosit1($val) { $val = 10; }    // érték szerinti…
function modosit2(&$val) { $val = 10; }   // referencia szerinti…

$a = 5;
modosit1($a);    // $a értéke továbbra is 5 ("másolat")
modosit2($a);    // $a értéke 10 lesz
```

- PHP-ban lehetőségünk van a függvényparamétereknek **default érték**et adni
  - ha a hívás során nem adjuk meg a paraméter értékét, akkor a default értéket veszi fel
  - fontos, hogy ha a függvénynek több paramétere van, és nem minden paraméternek adunk default értéket, akkor a default értékek a paraméterlista **jobb** oldalán állnak

```
function koszon($nev = "senki") {
   echo "Szia, $nev! <br/>";
}

koszon();
koszon("Tamás");
```

```
Szia, senki!
Szia, Tamás!
```

- A függvényeknek lehet visszatérési értéke is, ezt a `return` kulcsszóval adjuk meg
  - ha a függvény referenciával tér vissza, akkor a függvény neve elé `&`-jelet írunk mind a függvénydefiníció, mind a függvényhívás helyén
- PHP-ban megszorításokat tehetünk a függvényparaméterek és visszatérési értéknek típusára
  - **Példa:** egy függvény, ami két `int` típusú paramétert vár, és `float` értékkel tér vissza

```
function osztas(int $a, int $b) : float { … }
```

- Változó számú paraméterlistát a `...` operátorral adhatunk meg
  - **Példa:** egy függvény, ami tetszőleges számú `int` típusú paramétert vár

```
function osszead(int ...$szamok) : float { … }
```