Karakterláncos (string) kezelése

A karakterláncoknak kiemelt jelentőségük van a PHP nyelvben, mivel a böngészőből érkező adatoknak nincs típusa, így karakterláncként érkezik.

A karakterlánc karakterek sorozata, amely karakterenkét egy byte-ot használnak, nem támogatja alapértelmezetten a PHP a Unicode-t.

Karakterláncokat négyféle képen lehet megadni

Aposztrófók között

Dupla aposztrófok (macskaköröm) között

Heredoc szintaxissal

Nowdoc szintaxissal

Szimpla aposztróf

```
<?php
$szoveg = 'Példa szöveg ';
?>
```

Dupla aposztróf

```
<?php
$szoveg = "Példa szöveg ";
?>
```

A két módozat között jelentős eltérés van. A szimpla aposztrófok közötti szövegek egy az egyben megjelennek a böngészőben.

```
<?php
$szoveg = 'Példa szöveg \n';
Print($szoveg);
?>
Eredmény: Példa szöveg \n
```

Ezzel szemben a dupla aposztróf vagy más néven macskaköröm.

```
<?php
$szoveg = "Példa szöveg \n";
Print($szoveg);
?>
Eredmény: Példa szöveg
```

Itt egy nem látható sortörés van, amelyet a böngészőben nem veszünk észre. Továbbá a dupla aposztróf vayg heredoc esetén a szövegbe ágyazott változókat kiértékeli a PHP.

```
<?php
$szam = 55;
$szoveg = "Érték: $szam";
Print($szoveg);
?>
```

```
Eredmény: Érték: 55
```

PHP 7 től lehetőség van Unicode karakterkód formában is megadni egy Unicode karakterláncot.

```
<?php
echo "\u{9999}";
```

Heredoc szintaxis

```
<?php
// no indentation
echo <<<END
      a
     b
    c
\n
END;
```

```
Eredmény:
      a
     b
    c
```

```
<?php
echo <<<EOT
\u{01f418}
EOT;
```

Nowdoc szintaxis

```
<?php
echo <<<'EOD'
Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always treated literally,
e.g. \\ and \'.
EOD;
```

```
Eredmény: Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always treated literally,
e.g. \\ and \'.
```

A karakterlánc literálok megadása után nézzük meg, milyen műveleteket lehet végezni a karakterláncokkal.

Karakterlánc kiíratása böngészőben, a print, illetve az echo függvényekkel oldhatóak meg. A print és az echo körül éles vita szokott kialakulni, semmi értelme, nem lesz mérhetően gyorsabb, ha az egyiket vagy a másikat használjuk. A print egy függvény, amely egy 1 -est ad vissza minden esetben, míg az echo egy utasítás.

```
<?php
print("Ez egy szöveg");
echo "Ez egy szöveg";
```
```
Eredmény: Ez egy szöveg Ez egy szöveg
```
Észrevehető, hogy nincs sortörés, a böngésző a HTML tag-eket érti, így ezeket is kell használnunk.

```
<?php
print("Ez egy szöveg<br>");
echo "Ez egy szöveg";
```
```
Eredmény: Ez egy szöveg
Ez egy szöveg
```
Ha bármit el akarunk érni, akkor a HTML tag-eket is ismerni kell.

A karakterlánc hosszát az **strlen** függvénnyel lehet megtudni, bemenete a karakterlánc kimenete a hossz. Figyelem a kimenet csak ASCII karakterek esetén ad helyes értéket UNICODE esetén az **mb_strlen** függvényt kell használni.

A karakterláncon belül keresésre több lehetőségünk van, a függvények között apró eltérések vannak. Az **strpos** függvény bemenete a karakterlánc, amiben keresünk , a keresett karakterlánc, opcionálisan egy indexet is meg lehet adni, hogy mely indextől kezd keresni a szövegben. Visszatérési értéke egy egész szám (int) az index vagy ha nincs találat akkor false.

```
<?php
$amibenKeresek = "Ez egy feladat";
$amitKeresek = "e";
$index = strpos($amibenKeresek, $amitKeresek);
Print($index);
```

Amennyiben több találat is lenne, akkor használni kell az opcionális index paramétert is, amire az alábbi példa szolgál.

```
<?php
$amibenKeresek = "Ez egy feladat";
$amitKeresek = "e";
```

```
$index = 0;
while($index = strpos($amibenKeresek, $amitKeresek, $index)){
      print($index);
      $index++;

}
```

Ez a függvény egy egész számot vagy false értéket ad vissza, itt figyelni kell, ha 0 értéket ad vissza és azt == vagy != operátorral hasonlítjuk össze egy logikai/bool értékkel, akkor a 0. poiciót false-nak értelmezi a típus konverzió miatt, helyette használjuk a === vagy a !== operátort.

A többi függvény eltérései.

stripos Ugyanaz mint a strpos, de nem érzékeny a kis és nagybetűre. Case insensitive.

strrpos A keresett karakterlánc utolsó előfordulását adja vissza.

strripos A keresett karakterlánc utolsó előfordulását adja vissza, de nem kis -nagy betű érzékeny.

A fenti függvények az **strpos** változatai.

A gyakorlati feladatok közül az 5. feladatban ez a függvény kulcs szerepet játszik.

Egy egy apró de néha lényeges témakör a kis betű nagybetű közötti váltás. Ezeket az **strtoupper** és az **strtolower** függvényekkel érhetjük el. A bemeneteik egyszerűen az adott karakterlánc és a kimenet a csupa nagybetűs vagy csupa kisbetűs szöveg. Ehhez a témához kapcsolódik az **ucwords** függvény, amely minden szó kezdőbetűjét nagybetűssé tesz, azaz a szöveg első betűjét a szóközök utáni első betűt. Ez jól jöhet, ha a böngészőből beküldött neveket kell egyforma formára hozni.

A következő gyakran használt függvény egy szöveg részét veszi ki és adja vissza. **substr** , itt a bemenet a következő, a karakterlánc, amiben a rész szöveg van, a kezdő pozíció és a hossz, amennyi szöveget kiszedjen a karakterláncból.

Számjegyeket tartalmazó szövegből hogyan lesz szám? Mivel jeleztük feljebb, hogy a böngészőből érkező adatok karakterláncként érkeznek a PHP felé, így ez egy lényeges kérdés, ha mennyiséget vagy árat kérünk be a böngészőn keresztül.

```
<?php
$szamSzoveg = "9";
$szam = (int)$szamSzoveg;
```

A fenti példakód a típuskényszerítést (type cast) mutatja be. Amelyben egy karakterláncot egész számmá alakítunk. Ebben az esetben a PHP egy nagyon rugalmas nyelv, és nem kéri, hogy csak számjegyeket tartalmazzon a karakterlánc, addig alakítja számmá, ameddig számjegyeket talál, az első karakternél megáll, ha csak karaktereket tartalmaz a szöveg, akkor pedig 0 lesz belőle, azaz hibát nem dob a rendszer.

A karakterláncot számmá alakítani lehet függvények segítségével is.

Commented [LF1]: Itt példákkal folytatni kell.

A karakterláncokkal további műveletek a szöveg részekre bontása. Erre több függvény is van. A egyik amelyik karakterekre bontja a karakterláncot az **str_split** a bemenete a karakterlánc a kimenete egy karakterekből álló tömb.

A másik függvény az **explode**, amely egy szeparátor jelet kér és a darabokra szedhető karakterlánc a második paramétere. Ez a 7. gyakorló feladatnál lesz fontos függvény.

Szövegek feldolgozásában egy fontos függvény a **sscanf**.

Ez a függvény egy karakterláncot értelmez és ad vissza tömbként. Bemenete egy szöveg és egy formátum, ekkor egy tömböt ad vissza a talált értékekkel, ha az opcionális paramétert is használjuk, akkor a talált értékek számát adja vissza és a talált értéket az opcionális paraméterbe teszi.

D Decimális szám.

i integer/egész szám.

n az eddig feldolgozott karakterek számát jelöli.

s egy szövegrészt olvas be szóközig

* instead of argnum$ suppresses the assignment of this conversion specification.

```
<?php
// getting the serial number
list($serial) = sscanf("SN/2350001", "SN/%d");
// and the date of manufacturing
$mandate = "January 01 2000";
list($month, $day, $year) = sscanf($mandate, "%s %d %d");
echo "Item $serial was manufactured on: $year-" . substr($month, 0, 3)
. "-$day\n";
?>
```

```
<?php
// get author info and generate DocBook entry
$auth = "24\tLewis Carroll";
$n = sscanf($auth, "%d\t%s %s", $id, $first, $last);
echo "<author id='$id'>
    <firstname>$first</firstname>
    <surname>$last</surname>
</author>\n";
?>
```

Szövegekben szövegrészek átírására szolgál a **str_replace** függvény.

str_ireplace

wordwrap, nl2br

trim, ltrim, rtrim

Érdekesség soundex,

Number_format

Money_format

Printf

Sprintf

Str_word_count

Hash függvények md5, sha1

Regex függvények

Bejövő karakterláncok ellenőrzése

Filter_var

Gyakorlatok

1. Többször szereplő érték keresése

2. Palindrom-e egy karakterlánc

3. Tajszám ellenőrzés

4. Bankszámlaszám ellenőrzés

5. Egy hosszú (5-600 karakter) karakterlánc intelligens vágása, azaz ne szó közepén, hanem írásjelnél mondat végén vágjon.

6. Szám kiírása szövegként

7. Elsőfokú egyenlet megoldása