18. ÓRA

A szabályos kifejezések használata

A szövegek vizsgálatának és elemzésének nagyszerű módja a szabályos kifejezések (regular expressions) használata. Ezek lehetővé teszik, hogy egy karakterláncon belül mintákat keressünk, a találatokat pedig rugalmasan és pontosan kapjuk vissza. Mivel hatékonyabbak az előző órában tanult karakterlánc-kezelő függvényeknél, lassabbak is azoknál. Így azt tanácsoljuk, hogy ha nincs különösebb szükségünk a szabályos kifejezések által biztosított hatékonyságra, inkább használjuk a hagyományos karakterlánc-kezelő függvényeket.

A PHP a szabályos kifejezések két típusát támogatja. Egyrészt támogatja a Perl által használtakat, az azoknak megfelelő függvényekkel, másrészt a korlátozottabb tudású POSIX szabályos kifejezés formát. Mindkettővel meg fogunk ismerkedni.

Az óra során a következőkről tanulunk:

- Hogyan illesszünk mintát egy karakterláncra a szabályos kifejezések segítségével?

- Mik a szabályos kifejezések formai követelményei?

- Hogyan cseréljünk le karakterláncokat szabályos kifejezésekkel?
- Hogyan keressünk és cseréljünk le mintákat egy szövegben a hatékony Perl típusú szabályos kifejezésekkel?

A POSIX szabályos kifejezések függvényei

A POSIX szabályos kifejezéseket kezelő függvények lehetővé teszik, hogy egy szövegben bonyolult mintákat keressünk és cseréljünk le. Ezeket általában egyszerűen szabályoskifejezés-függvényeknek szokták nevezni, mi azonban POSIX szabályoskifejezés-függvényekként utalunk rájuk, egyrészt azért, hogy megkülönböztethessük őket a hasonló, ám hatékonyabb Perl típusú szabályos kifejezésektől, másrészt azért, mert a POSIX bővített szabályos kifejezés (extended regular expression) szabványát követik.

A szabályos kifejezések olyan jelegyüttesek, amelyek egy szövegben egy mintára illeszkednek. Használatuk elsajátítása így jóval többet jelent annál, hogy megtanuljuk a PHP szabályoskifejezés-függvényeinek be- és kimeneti paramétereit. Az ismerkedést a függvényekkel kezdjük és rajtuk keresztül vezetjük be az olvasót a szabályos kifejezések formai követelményeibe.

Minta keresése karakterláncokban az ereg() függvénnyel

Az `ereg()` bemenete egy mintát jelképező karakterlánc, egy karakterlánc, amiben keresünk, és egy tömbváltozó, amelyben a keresés eredményét tároljuk. A függvény egy egész számot ad vissza, amely a megtalált minta illeszkedő karaktereinek számát adja meg, illetve ha a függvény nem találja meg a mintát, a `false` (hamis) értéket adja vissza. Keressük a `"tt"`-t az `"ütődött tánctanár"` karakterláncban.

```
print ereg("tt", "ütődött tánctanár", $tomb);
print "<br>$tomb[0]<br>";
// a kimenet:
// 2
// tt
```

A `"tt"` az `"ütődött"` szóban megtalálható, ezért az `ereg()` 2-t ad vissza, ez az illeszkedő betűk száma. A `$tomb` változó első elemében a megtalált karakterlánc lesz, amit aztán a böngészőben megjelenítünk. Felmerülhet a kérdés, milyen esetben lehet ez hasznos, hiszen előre tudjuk, hogy mi a keresett minta. Nem kell azonban mindig előre meghatározott karaktereket keresnünk. A pontot (`.`) használhatjuk tetszőleges karakter keresésekor:

```
print ereg("d.", "ütődött tánctanár", $tomb);
print "<br>$tomb[0]<br>";
// a kimenet:
// 2
// dö
```

A `d.` minta illeszkedik minden olyan szövegre, amely megfelel annak a meghatározásnak, hogy „egy d betű és egy azt követő tetszőleges karakter”. Ebben az esetben nem tudjuk előre, mi lesz a második karakter, így a `$tomb[0]` értéke máris értelmet nyer.

Egynél többször előforduló karakter keresése mennyiségjelzővel

Amikor egy karakterláncban egy karaktert keresünk, egy mennyiségjelzővel (kvantorral) határozhatjuk meg, hányszor forduljon elő a karakter a mintában. Az `a+` minta például azt jelenti, hogy az `"a"`-nak legalább egyszer elő kell fordulnia, amit `0` vagy több `"a"` betű követ. Próbáljuk ki:

```
if ( ereg("a+","aaaa", $tomb) )
   print $tomb[0];
// azt írja ki, hogy "aaaa";
```

Vegyük észre, hogy ez a szabályos kifejezés annyi karakterre illeszkedik, amennyire csak tud. A 18.1. táblázat az ismétlődő karakterek keresésére szolgáló mennyiségjelzőket ismerteti.

18.1. táblázat Az ismétlődő karakterek mennyiségjelzői

Jel	*Leírás*	*Példa*	*Erre illeszkedik*	*Erre nem illeszkedik*
`*`	Nulla vagy több előfordulás	`a*`	`xxxx`	(Mindenre illeszkedik)
`+`	Egy vagy több előfordulás	`a+`	`xaax`	`xxxx`
`?`	Nulla vagy egy előfordulás	`a?`	`xaxx`	`xaax`
`{n}`	*n* előfordulás	`a{3}`	`xaaa`	`aaaa`
`{n,}`	Legalább *n* előfordulás	`a{3,}`	`aaaa`	`aaxx`
`{,n}`	Legfeljebb *n* előfordulás	`a{,2}`	`xaax`	`aaax`
`{n1,n2}`	Legalább *n1* és legfeljebb *n2* előfordulás	`a{1,2}`	`xaax`	`xaaa`

A kapcsos zárójelben levő számok neve korlát. Segítségükkel határozhatjuk meg, hányszor ismétlődjön egy adott karakter, hogy a minta érvényes legyen rá.

ÚJDONSÁG A korlát határozza meg, hogy egy karakternek vagy karakterláncnak hányszor kell ismétlődnie egy szabályos kifejezésben. Az alsó és felső határt a kapcsos zárójelen belül határozzuk meg. Például az

```
a{4,5}
```

kifejezés az `a` négynél nem kevesebb és ötnél nem több előfordulására illeszkedik.

Vegyünk egy példát. Egy klub tagsági azonosítóval jelöli tagjait. Egy érvényes azonosítóban egy és négy közötti alkalommal fordul elő a `"t"`, ezt tetszőleges számú szám vagy betű karakter követi, majd a 99-es szám zárja. A klub arra kért fel bennünket, hogy a tennivalókat tartalmazó naplóból emeljük ki a tagazonosítókat.

```
$proba = "A ttXGDH99 tagunk befizette már a tagsági
          díjat?";
if ( ereg( "t{1,4}.*99 ", $proba, $tomb ) )
print "A megtalált azonosító: $tomb[0]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99"
```

Az előző kódrészletben a tagazonosító két `t` karakterrel kezdődik, ezt négy nagybetű követi, majd végül a 99 jön. A `t{1,4}` minta illeszkedik a két `t`-re, a négy nagybetűt pedig megtalálja a `.*`, amellyel tetszőleges számú, tetszőleges típusú karakterláncot kereshetünk.

Úgy tűnik, ezzel megoldottuk a feladatot, pedig még attól meglehetősen távol állunk. A feltételek között a 99-cel való végződést azzal próbáltuk biztosítani, hogy megadtuk, hogy egy szóköz legyen az utolsó karakter. Ezt aztán találat esetén vissza is kaptuk. Még ennél is nagyobb baj, hogy ha a forráslánc

```
"Azonosítóm ttXGDH99 megkaptad már az 1999 -es tagsági díjat?"
```

akkor a fenti szabályos kifejezés a következő karakterláncot találná meg:

```
"tóm ttXGDH99 megkaptad már az 1999"
```

Mit rontottunk el? A szabályos kifejezés megtalálta a `t` betűt az azonosítóm szóban, majd utána a tetszőleges számú karaktert egészen a szóközzel zárt `99`-ig. Jellemző a szabályos kifejezések „mohóságára”, hogy annyi karakterre illeszkednek, amennyire csak tudnak. Emiatt történt, hogy a minta egészen az `1999`-ig illeszkedett, a `99`-re végződő azonosító helyett. A hibát kijavíthatjuk, ha biztosan tudjuk, hogy a `t` és a `99` közötti karakterek kizárólag betűk vagy számok és egyikük sem szóköz. Az ilyen feladatokat egyszerűsítik le a karakterosztályok.

Karakterlánc keresése karakterosztályokkal

Az eddigi esetekben megadott karakterekkel vagy a . segítségével kerestünk karaktereket. A karakterosztályok lehetővé teszik, hogy karakterek meghatározott csoportját keressük. A karakterosztály meghatározásához a keresendő karaktereket szögletes zárójelek közé írjuk. Az [ab] minta egyetlen betűre illeszkedik, ami vagy a vagy b. A karakterosztályokat meghatározásuk után karakterként kezelhetjük, így az [ab]+ az aaa, bbb és abababab karakterláncokra egyaránt illeszkedik.

Karaktertartományokat is használhatunk karakterosztályok megadására: az [a-z] tetszőleges kisbetűre, az [A-Z] tetszőleges nagybetűre, a [0-9] pedig tetszőleges számjegyre illeszkedik. Ezeket a sorozatokat és az egyedi karaktereket egyetlen karakterosztályba is gyúrhatjuk, így az [a-z5] minta tetszőleges kisbetűre vagy az 5-ös számra illeszkedik.

A karakterosztályokat „tagadhatjuk" is, a csúcsos ékezet (^) kezdő szögletes zárójel után való írásával: az [^A-Z] mindenre illeszkedik, csak a nagybetűkre nem.

Térjünk vissza legutóbbi példánkra. Mintát kell illesztenünk egy 1 és 4 közötti alkalommal előforduló betűre, amelyet tetszőleges számú betű vagy számjegy karakter követ, amit a 99 zár.

```
$proba = "Azonosítóm ttXGDH99 megkaptad már az 1999 -es
          tagsági díjat?";
if ( ereg( "t{1,4}[a-zA-Z0-9]*99 ", $proba, $tomb ) )
   print "A megtalált azonosító: $tomb[0]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99 "
```

Szép lassan alakul. Az a karakterosztály, amit hozzáadtunk, már nem fog szóközökre illeszkedni, így végre az azonosítót kapjuk meg. Viszont ha az azonosító után a próba-karakterlánc végére vesszőt írunk, a szabályos kifejezés megint nem illeszkedik:

```
$proba = "Azonosítóm ttXGDH99, megkaptad már az 1999 -es
          tagsági díjat?";
if ( ereg( "t{1,4}[a-zA-Z0-9]*99 ", $proba, $tomb ) )
   print "A megtalált azonosító: $tomb[0]";
// a szabályos kifejezés nem illeszkedik
```

Ez azért van, mert a minta végén egy szóközt várunk el, amely alapján meggyőződhetünk, hogy elértük az azonosító végét. Így ha egy szövegben az azonosító zárójelek közt van, kötőjel vagy vessző követi, a minta nem illeszkedik rá. Közelebb visz a megoldáshoz, ha úgy javítunk a szabályos kifejezésen, hogy mindenre illeszkedjék, csak szám és betű karakterekre nem:

```
$proba = "Azonosítóm ttXGDH99, megkaptad már az 1999 -es
          tagsági díjat?";
if ( ereg( "t{1,4}[a-zA-Z0-9]*99[^a-zA-Z0-9]", $proba,
$tomb ) )
   print "A megtalált azonosító: $tomb[0]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99, "
```

Már majdnem kész is vagyunk, de még mindig van két probléma. Egyrészt a vesszőt is visszakapjuk, másrészt a szabályos kifejezés nem illeszkedik, ha az azonosító a karakterlánc legvégén van, hiszen megköveteltük, hogy utána még egy karakter legyen. Más szóval a szóhatárt kellene megbízható módon megtalálnunk. Erre a problémára később még visszatérünk.

Az atomok kezelése

ÚJDONSÁG

Az atom egy zárójelek közé írt minta (gyakran hivatkoznak rá részmintaként is). Meghatározása után az atomot ugyanúgy kezelhetjük, mintha maga is egy karakter vagy karakterosztály lenne. Más szóval ugyanazt a 18.1. táblázatban bemutatott rendszer alapján felépített mintát annyiszor kereshetjük, ahányszor csak akarjuk.

A következő kódrészletben meghatározunk egy mintát, zárójelezzük, és az így kapott atomnak kétszer kell illeszkednie a keresendő szövegre:

```
$proba = "abbaxaabaxabbax";
if ( ereg( "([ab]+x){2}", $proba, $tomb ) )
    print "$tomb[0]";
// azt írja ki, hogy "abbaxaabax"
```

Az `[ab]+x` illeszkedik az `"abbax"` és az `"aabax"` karakterláncokra egyaránt, így az `([ab]+x){2}` illeszkedni fog az `"abbaxaabax"` karakterláncra.

Az `ereg()` függvénynek átadott tömbváltozó első elemében kapjuk vissza a teljes, megtalált, illeszkedő karakterláncot. Az ezt követő elemek az egyes megtalált atomokat tartalmazzák. Ez azt jelenti, hogy a teljes találat mellett a megtalált minta egyes részeihez is hozzáférhetünk.

A következő kódrészletben egy IP címre illesztünk mintát, és nem csupán a teljes címet tároljuk, hanem egyes alkotóelemeit is:

```
$ipcim = "158.152.55.35";
if ( ereg( "([0-9]+)\.([0-9]+)\.([0-9]+)\.([0-9]+)", $ipcim,
     $tomb ) )
  {
  foreach ( $tomb as $ertek )
      print "$ertek<BR>";
  }
// Az eredmény:
// 158.152.55.35
// 158
// 152
// 55
// 35
```


Vegyük észre, hogy a pontokat a szabályos kifejezésben fordított perjel előzte meg. Ezzel fejeztük ki, hogy a . karaktert itt minden különleges tulajdonsága nélkül, egyszerű karakterként szeretnénk kezelni. Minden olyan karakternél ez a követendő eljárás, amely egyedi szereppel bír a szabályos kifejezésekben.

Elágazások

A szűrőkarakter (|) segítségével mintákat összekötve a szabályos kifejezéseken belül elágazásokat hozhatunk létre. Egy kétágú szabályos kifejezés vagy az első mintára, vagy a második mintára illeszkedő karakterláncokat keresi meg. Ettől lesz a szabályos kifejezések nyelvtana még rugalmasabb. A következő kódrészletben a `.com`, a `.de` vagy a `.hu` karakterláncot keressük:

```
$domain = "www.egydomain.hu";
if ( ereg( "\.com|\.de|\.hu", $domain, $tomb ) )
print "ez egy $tomb[0] tartomány<BR>";
// azt írja ki, hogy "ez egy .hu tartomány"
```

A szabályos kifejezés helye

Nemcsak a keresendő mintát adhatjuk meg, hanem azt is, hol illeszkedjen egy karakterláncban. Ha a mintát a karakterlánc kezdetére szeretnénk illeszteni, a szabályos kifejezés elejére írjunk egy csúcsos ékezetet (^). A `^a` illeszkedik az `"alma"`, de nem illeszkedik a `"banán"` szóra.

A karakterlánc végén úgy illeszthetjük a mintát, hogy dollárjelet (`$`) írunk a szabályos kifejezés végére. Az `a$` illeszkedik a `"róka"`, de nem illeszkedik a `"hal"` szóra.

A tagazonosítót kereső példa újragondolása

Most már rendelkezésünkre állnak azok az eszközök, melyekkel megoldhatjuk a tagazonosítás problémáját. Emlékeztetőül, a feladat az volt, hogy szövegeket elmezve kigyűjtsük azokat a tagazonosítókat, amelyekben a `"t"` egy és négy közötti alkalommal fordul elő, ezt tetszőleges számú szám vagy betű karakter követi, amelyet a `99` zár. Jelenlegi problémánk az, hogyan határozzuk meg, hogy illeszkedő mintánknak szóhatárra kell esnie. Nem használhatjuk a szóközt, mivel a szavakat írásjelekkel is el lehet választani. Az sem lehet feltétel, hogy egy nem alfanumerikus karakter alkossa a határt, hiszen a tagazonosító lehet, hogy éppen a szöveg elején vagy végén található.

Most, hogy képesek vagyunk elágazásokat megadni és helyhez kötni a szabályos kifejezéseket, megadhatjuk feltételként, hogy a tagazonosítót vagy egy nem szám vagy betű karakter kövesse, vagy pedig a karakterlánc vége. Ugyanezzel a gondolatmenettel adhatjuk meg a kód elején előforduló szóhatárt is. A zárójelek használatával a tagazonosítót minden központozás és szóköz nélkül kaphatjuk vissza:

```
$proba = "Azonosítóm ttXGDH99, megkaptad már az 1999 -es
          tagsági díjat?";
if ( ereg( "(^|[^a-zA-Z0-9])(t{1,4}[a-zA-Z0-9]*99)([^a-zA-
     Z0-9]|$)", $proba, $tomb ) )
  print "A megtalált azonosító: $tomb[2]";
// azt írja ki, hogy "A megtalált azonosító: ttXGDH99"
```

Amint láthatjuk, a szabályos kifejezések elsőre egy kicsit riasztók, de miután kisebb egységekre bontjuk azokat, általában egész könnyen értelmezhetők. Elértük, hogy mintánk egy szóhatárra illeszkedjen (pontosan úgy, ahogy azt a feladat megkövetelte). Mivel nem vagyunk kíváncsiak semmilyen, a mintát megelőző vagy követő szövegre, a minta számunkra hasznos részét zárójelek közé írjuk. Az eredményt a `$tomb` tömb harmadik (2-es indexű) elemében találhatjuk meg.

Az ereg() megkülönbözteti a kis- és nagybetűket. Ha nem szeretnénk megkülönböztetni őket, használjuk az eregi() függvényt, amely egyébként minden más tekintetben megegyezik az ereg() függvénnyel.

Minták lecserélése karakterláncokban az ereg_replace() függvénnyel

Eddig csupán mintákat kerestünk a karakterláncokban, magát a karakterláncot nem módosítottuk. Az `ereg_replace()` függvény lehetővé teszi, hogy megkeressük és lecseréljük az adott szöveg egy mintára illeszkedő részláncát. Bemenete három karakterlánc: egy szabályos kifejezés, a csereszöveg, amivel felül szeretnénk írni a megtalált mintát és a forrásszöveg, amelyben keresünk. Találat esetén a függvény a megváltoztatott karakterláncot adja vissza, ellenkező esetben az eredeti forrásszöveget kapjuk. A következő kódrészletben egy klubtisztviselő nevét keressük ki, majd felülírjuk utódjának nevével:


```
$proba = "Titkárunk, Vilmos Sarolta, szeretettel
          üdvözli önt.";
print ereg_replace ("Vilmos Sarolta", "László István",
                    $proba);
// azt írja ki, hogy " Titkárunk, László István,
                      szeretettel üdvözli önt."
```

Fontos megjegyezni, hogy míg az `ereg()` függvény csak a legelső megtalált mintára illeszkedik, az `ereg_replace()` a minta összes előfordulását megtalálja és lecseréli.

Visszautalás használata az ereg_replace() függvénnyel

A visszautalások azt teszik lehetővé, hogy az illeszkedő kifejezés egy részét felhasználhassuk a felülíró karakterláncban. Ehhez szabályos kifejezésünk összes felhasználandó elemét zárójelbe kell írnunk. Ezekre a részminták segítségével megtalált karakterláncokra két fordított perjellel és az atom számával hivatkozhatunk (például `\\1`) a cserelláncokban. Az atomok sorszámozottak, a sorszámozás kívülről befelé, balról jobbra halad és `\\1`-től indul. A `\\0` az egész illeszkedő karakterláncot jelenti.

A következő kódrészlet a `hh/nn/éééé` formátumú dátumokat alakítja `éééé.hh.nn.` formátumúvá:

```
$datumt = "12/25/2000";
print ereg_replace("([0-9]+)/([0-9]+)/([0-9]+)",
                   "\\3.\\1.\\2.", $datum);
// azt írja ki, hogy "2000.12.25"
```

Vegyük észre, hogy a fenti kód csereszövegében a pontok nem különleges értelműek, így nem is kell `\` jelet tenni eléjük. A `.` az első paraméterben kapja a tetszőleges karakterre illeszkedés különleges jelentését.

Az ereg_replace() függvény megkülönbözteti a kis- és nagybetűket. Ha nem szeretnénk megkülönböztetni ezeket, használjuk az eregi_replace() függvényt, amely minden más tekintetben megegyezik az ereg_replace() függvénnyel.

Karakterláncok felbontása a split() függvénnyel

Az előző órában már láttuk, hogy az `explode()` függvény segítségével egy karakterláncot elemeire bontva tömbként kezelhetünk. Az említett függvény hatékonyságát az csökkenti, hogy határolójelként csak egy karakterhalmazt használhatunk. A PHP 4 `split()` függvénye a szabályos kifejezések hatékonyságát biztosítja, vele rugalmas határolójelet adhatunk meg. A függvény bemenete a határolójelként használt minta és a forráslánc, amelyben a határolókat keressük, kimenete pedig egy tömb. A függvény harmadik, nem kötelező paramétere a visszaadandó elemek legnagyobb számát határozza meg.

A következő példában egy elágazó szabályos kifejezéssel egy karakterláncot bontunk fel, úgy, hogy a határolójel egy vessző és egy szóköz vagy egy szóközökkel közrefogott és szó:

```
$szoveg = "almák, narancsok, körték és barackok";
$gyumolcsok = split(", | és ", $szoveg);
foreach ( $gyumolcsok as $gyumolcs )
    print "$gyumolcs<BR>";
// a kimenet:
// almák
// narancsok
// körték
// barackok
```

Perl típusú szabályos kifejezések

Ha Perlös háttérrel közelítjük meg a PHP-t, akkor a POSIX szabályos kifejezések függvényeit valamelyest nehézkesnek találhatjuk. A jó hír viszont az, hogy a PHP 4 támogatja a Perlnek megfelelő szabályos kifejezéseket is. Ezek formája még az eddig tanultaknál is hatékonyabb. Ebben a részben a két szabályoskifejezés-forma közötti különbségekkel ismerkedünk meg.

Minták keresése a preg_match() függvénnyel

A `preg_match()` függvény három paramétert vár: egy szabályos kifejezést, a forrásláncot és egy tömbváltozót, amelyben az illeszkedő karakterláncokat adja vissza. A `true` (igaz) értéket kapjuk vissza, ha illeszkedő kifejezést talál és `false` (hamis) értéket, ha nem. A `preg_match()` és az `ereg_match()` függvények közötti különbség a szabályos kifejezést tartalmazó paraméterben van. A Perl típusú szabályos kifejezéseket határolójelek közé kell helyezni. Általában perjeleket használunk határolójelként, bár más, nem szám vagy betű karaktert is használhatunk (kivétel persze a fordított perjel). A következő példában a `preg_match()` függvénnyel egy h-valami-z mintájú karakterláncot keresünk:


```
$szoveg = "üvegház";
if ( preg_match( "/h.*z/", $szoveg, $tomb ) )
     print $tomb[0];
// azt írja ki, hogy "ház"
```

A Perl típusú szabályos kifejezések és a mohóság

Alapértelmezés szerint a szabályos kifejezések megkísérelnek a lehető legtöbb karakterre illeszkedni. Így a

```
"/h.*z/"
```

minta a `h`-val kezdődő és `z` karakterre végződő lehető legtöbb karaktert közrefogó karakterláncra fog illeszkedni, a következő példában a teljes szövegre:

```
$szoveg = "ház húz hülyéz hazardíroz";
if ( preg_match( "/h.*z/", $szoveg, $tomb ) )
     print $tomb[0];
// azt írja ki, hogy "ház húz hülyéz hazardíroz"
```

Viszont ha kérdőjelet (`?`) írunk a mennyiséget kifejező jel után, rávehetjük a Perl típusú szabályos kifejezést, hogy egy kicsit mértékletesebb legyen. Így míg a

```
"h.*z"
```

minta azt jelenti, hogy „h és z között a lehető legtöbb karakter”, a

```
"h.*?z"
```

minta jelentése a „`h` és `z` közötti lehető legkevesebb karakter”.

A következő kódrészlet ezzel a módszerrel keresi meg a legrövidebb szót, amely h-val kezdődik és z-vel végződik:

```
$szoveg = "ház húz hülyéz hazardíroz";
if ( preg_match( "/h.*?z/", $szoveg, $tomb ) )
     print $tomb[0];
// azt írja ki, hogy "ház"
```

A Perl típusú szabályos kifejezések és a fordított perjeles karakterek

A Perl típusú szabályos kifejezésekben is vezérlőkarakterré változtathatunk bizonyos karaktereket, ugyanúgy, mint ahogy azt karakterláncokkal tettük. A \t például a tabulátor karaktert jelenti, az \n pedig a soremelést. Az ilyen típusú szabályos kifejezések olyan karaktereket is leírhatnak, amelyek teljes karakterhalmazokra, karaktertípusokra illeszkednek. A fordított perjeles karakterek listáját a 18.2. táblázatban láthatjuk.

18.2. táblázat Karaktertípusokra illeszkedő beépített karakterosztályok

Karakter	*Illeszkedik*
\d	Bármely számra
\D	Mindenre, ami nem szám
\s	Bármely elválasztó karakterre
\S	Mindenre, ami nem elválasztó karakter
\w	Bármely szóalkotó karakterre (aláhúzást is beleértve)
\W	Mindenre, ami nem szóalkotó karakter

Ezek a karakterosztályok nagymértékben leegyszerűsíthetik a szabályos kifejezéseket. Nélkülük karaktersorozatok keresésekor saját karakterosztályokat kellene használnunk. Vessük össze a szóalkotó karakterekre illeszkedő mintát az ereg() és a preg_match() függvényben:

```
ereg( "h[a-zA-Z0-9_]+z", $szoveg, $tomb );
preg_match( "/h\w+z", $szoveg, $tomb );
```

A Perl típusú szabályos kifejezések több váltókaraktert is támogatnak, melyek hivatkozási pontként működnek. A hivatkozási pontok a karakterláncon belül helyekre illeszkednek, nem pedig karakterekre. Ismertetésüket a 18.3. táblázatban találhatjuk.

18.3. táblázat Hivatkozási pontként használt váltókarakterek

Karakter	*Illeszkedik*
`\A`	Karakterlánc kezdetére
`\b`	Szóhatárra
`\B`	Mindenre, ami nem szóhatár
`\Z`	Karakterlánc végére (az utolsó soremelés vagy a karakterlánc vége előtt illeszkedik)
`\z`	Karakterlánc végére (a karakterlánc legvégére illeszkedik)

Emlékszünk még, milyen problémáink voltak a szóhatárra való illesztéssel a tagsági azonosítást megvalósító példában? A Perl típusú szabályos kifejezések jelentősen leegyszerűsítik ezt a feladatot. Vessük össze, hogyan illeszt mintát szóalkotó karakterre és szóhatárra az `ereg()` és a `preg_match()`:

```
ereg ( "(^|[^a-zA-Z0-9_])(y{1,4}[a-zA-Z0-9_]*99)
 ➥ ([^a-zA-Z0-9_]|$)", $szoveg, $tomb );
preg_match( "\bt{1,4}\w*99\b", $szoveg, $tomb );
```

Az előző példában a `preg_match()` függvény meghívásakor a szóhatáron levő legalább egy, de legfeljebb négy `t` karakterre illeszkedik, amelyet bármennyi szövegalkotó karakter követhet és amit a szóhatáron levő `99` karaktersor zár. A szóhatárt jelölő váltókarakter igazából nem is egy karakterre illeszkedik, csupán megerősíti, hogy az illeszkedéshez szóhatár kell. Az `ereg_match()` meghívásához először létre kell hoznunk egy nem szóalkotó karakterekből álló mintát, majd vagy arra, vagy pedig szóhatárra kell illeszteni.

A váltókaraktereket arra is használhatjuk, hogy megszabaduljunk a karakterek jelentésétől. Ahhoz például, hogy egy `.` karakterre illeszthessünk, egy fordított perjelet kell elé írnunk.

Teljeskörű keresés a preg_match_all() függvénnyel

A POSIX szabályos kifejezésekkel az az egyik probléma, hogy a minta karakterláncon belüli összes előfordulását bonyolult megkeresni. Így ha az `ereg()` függvénnyel keressük a `b`-vel kezdődő, `t`-vel végződő szavakat, csak a legelső találatot kapjuk vissza. Próbáljuk meg magunk is:

```
$szoveg = "barackot, banánt, bort, búzát és
          békákat árulok";
if ( ereg( "(^|[^a-zA-Z0-9_])(b[a-zA-Z0-9_]+t)
     ➥ ([^a-zA-Z0-9_]|$)", $szoveg, $tomb ) )
   {
   for ( $x=0; $x< count( $tomb ); $x++ )
       print "\$tomb[$x]: $tomb[$x]<br>\n";
   }
// kimenete:
// $tomb[0]: barackot,
// $tomb[1]:
// $tomb[2]: barackot
// $tomb[3]: ,
```

Ahogy azt vártuk, a `$tomb` harmadik elemében találjuk az első találatot, a `"barackot"`. A tömb első eleme tartalmazza a teljes találatot, a második a szóközt, a negyedik a vesszőt. Hogy megkapjuk az összes illeszkedő kifejezést a szövegben, az `ereg_replace()` függvényt kellene használnunk, ciklikusan, hogy eltávolítsuk a találatokat a szövegből, mielőtt újra illesztenénk a mintát.

Fel kell, hogy hívjuk a kedves olvasó figyelmét arra, hogy a magyar szövegekre az ékezetes betűk miatt alapbeállításban nem alkalmazható a fentihez hasonló egyszerű szabályos kifejezés. A nemzeti beállítások testreszabására használható `setlocale()` függvényt kell alkalmaznunk, hogy a kívánt eredményt elérjük.

A `preg_match_all()` függvényt használhatjuk arra, hogy egyetlen hívásból a minta összes előfordulását megkapjuk. A `preg_match_all()` bemenete egy szabályos kifejezés, a forráslánc és egy tömbváltozó. Találat esetén `true` (igaz) értéket ad vissza. A tömbváltozó egy többdimenziós tömb lesz, melynek első eleme a szabályos kifejezésben megadott mintára illeszkedő összes találatot tartalmazza.

A 18.1. programban a `preg_match_all()` függvénnyel illesztettük mintánkat egy szövegre. Két `for` ciklust használunk, hogy megjelenítsük a többdimenziós eredménytömböt.

18.1. program Minta teljeskörű illesztése a preg_match_all() függvénnyel

```
1: <html>
2: <head>
3: <title>18.1. program Minta teljeskörű illesztése
          a preg_match_all() függvénnyel</title>
4: </head>
5: <body>
```

18.1. program (folytatás)

```
 6: <?php
 7: $szoveg = "barackot, banánt, bort, búzát és
              békákat árulok";
 8: if ( preg_match_all( "/\bb\w+t\b/", $szoveg, $tomb ) )
 9:      {
10:      for ( $x=0; $x< count( $tomb ); $x++ )
11:          {
12:          for ( $y=0; $y< count( $tomb[$x] ); $y++ )
13:              print "\$tomb[$x][$y]:
                        ".$tomb[$x][$y]."<br>\n";
14:          }
15:      }
16: // A kimenet:
17: // $tomb[0][0]: barackot
18: // $tomb[0][1]: banánt
19: // $tomb[0][2]: bort
20: // $tomb[0][3]: búzát
21: // $tomb[0][4]: békákat
22: ?>
23: </body>
24: </html>
```

A `$tomb` változónak a `preg_match_all()` függvénynek történő átadáskor csak az első eleme létezik és ez karakterláncok egy tömbjéből áll. Ez a tömb tartalmazza a szöveg minden olyan szavát, amely `b`-vel kezdődik és `t` betűre végződik.

A `preg_match_all()` ebből a tömbből azért készít egy többdimenziósat, hogy az atomok találatait is tárolhassa. A `preg_match_all()` függvénynek átadott tömb első eleme tartalmazza a teljes szabályos kifejezés minden egyes találatát. Minden további elem a szabályos kifejezés egyes atomjainak (zárójeles részmintáinak) értékeit tartalmazza az egyes találatokban. Így a `preg_match_all()` alábbi meghívásával

```
$szoveg = "01-05-99, 01-10-99, 01-03-00";
preg_match_all( "/(\d+)-(\d+)-(\d+)/", $szoveg, $tomb );
a $tomb[0] az összes találat tömbje:
$tomb[0][0]: 01-05-99
$tomb[0][1]: 01-10-99
$tomb[0][2]: 01-03-00
A $tomb[1] az első részminta értékeinek tömbje:
$tomb[1][0]: 01
```

```
$tomb[1][1]: 01
$tomb[1][2]: 01
A $tomb[2] a második részminta értékeinek tömbjét tárolja:
$tomb[2][0]: 05
$tomb[2][1]: 10
$tomb[2][2]: 03
```

és így tovább.

Minták lecserélése a preg_replace() függvénnyel

A `preg_replace()` használata megegyezik az `ereg_replace()` használatával, azzal a különbséggel, hogy hozzáférünk a Perl típusú szabályos kifejezések által kínált kényelmi lehetőségekhez. A `preg_replace()` bemenete egy szabályos kifejezés, egy csere-karakterlánc és egy forráslánc. Találat esetén a módosított karakterláncot, ellenkező esetben magát a forrásláncot kapjuk vissza változtatás nélkül. A következő kódrészlet a `nn/hh/éé` formátumú dátumokat alakítja át `éé/hh/nn` formátumúvá:

```
$szoveg = "25/12/99, 14/5/00";
$szoveg = preg_replace( "|\b(\d+)/(\d+)/(\d+)\b|",
 ➥ "\\3/\\2/\\1",  $szoveg );
print "$szoveg<br>";
// azt írja ki, hogy "99/12/25, 00/5/14"
```

Vegyük észre, hogy a szűrőkaraktert (`|`) használtuk határolójelként. Ezzel megtakarítottuk a perjelek kikerülésére használandó váltókaraktereket az illesztendő mintában. A `preg_replace()` ugyanúgy támogatja a visszautalásokat a csereszövegben, mint az `ereg_replace()`.

A `preg_replace()` függvénynek a forráslánc helyett karakterláncok egy tömbjét is átadhatjuk, az ebben az esetben minden tömbelemet egyesével átalakít. Ilyenkor a visszaadott érték az átalakított karakterláncok tömbje lesz.

Emellett a `preg_replace()` függvénynek szabályos kifejezések és csere-karakterláncok tömbjét is átadhatjuk. A szabályos kifejezéseket egyenként illeszti a forrásláncra és a megfelelő csereszöveggel helyettesíti. A következő példában az előző példa formátumait alakítjuk át, de emellett a forráslánc szerzői jogi (copyright) információját is módosítjuk:

```
$szoveg = "25/12/99, 14/5/00. Copyright 1999";
$miket = array( "|\b(\d+)/(\d+)/(\d+)\b|",
 ➥ "/([Cc]opyright) 1999/" );
$mikre = array( "\\3/\\2/\\1",   "\\1 2000" );
```

```
$szoveg = preg_replace( $miket, $mikre,  $szoveg );
print "$szoveg<br>";
// azt írja ki, hogy "99/12/25, 00/5/14. Copyright 2000"
```

Két tömböt hozunk létre. A `$miket` tömb két szabályos kifejezést, a `$mikre` tömb pedig csere-karakterláncokat tartalmaz. A `$miket` tömb első eleme a `$mikre` tömb első elemével helyettesítődik, a második a másodikkal, és így tovább.

Ha a csere-karakterláncok tömbje kevesebb elemet tartalmaz, mint a szabályos kifejezéseké, a csere-karakterlánc nélkül maradó szabályos kifejezéseket a függvény üres karakterláncra cseréli.

Ha a `preg_replace()` függvénynek szabályos kifejezések egy tömbjét adjuk át, de csak egy csereláncot, akkor az a szabályos kifejezések tömbjének összes mintáját ugyanazzal a csereszöveggel helyettesíti.

Módosító paraméterek

A Perl típusú szabályos kifejezések lehetővé teszik, hogy módosító paraméterek segítségével pontosítsuk a minta illesztési módját.

ÚJDONSÁG A módosító paraméter olyan betű, amelyet a Perl típusú szabályos kifejezések utolsó határolójele után írunk, és amely tovább finomítja szabályos kifejezéseink jelentését.

A Perl típusú szabályos kifejezések módosító paramétereit a 18.4. táblázatban találhatjuk.

18.4. táblázat A Perl típusú szabályos kifejezések módosító paraméterei

Minta	*Leírás*
`i`	Nem különbözteti meg a kis- és nagybetűket.
`e`	A `preg_replace()` csereláncát PHP-kódként kezeli.
`m`	A `$` és a `^` az aktuális sor elejére és végére illeszkedik.
`s`	A soremelésre is illeszkedik (a soremelések általában nem illeszkednek a `.`-ra).
`x`	A karakterosztályokon kívüli elválasztó karakterekre az olvashatóság érdekében nem illeszkedik. Ha elválasztó karakterekre szeretnénk mintát illeszteni, használjuk a `\s`, `\t`, vagy a `\` (fordított perjel–szóköz) mintákat.

18.4. táblázat. (folytatás)

Minta	*Leírás*
`A`	Csak a karakterlánc elején illeszkedik (ilyen megszorítás nincs a Perlben).
`E`	Csak a karakterlánc végén illeszkedik (ilyen megszorítás nincs a Perlben).
`U`	Kikapcsolja a szabályos kifejezések „mohóságát”, azaz a lehető legkevesebb karaktert tartalmazó találatokat adja vissza (ez a módosító sem található meg Perlben).

Ahol ezek a megszorítások nem mondanak egymásnak ellent, egyszerre többet is használhatunk belőlük. Az `x` megszorítást például arra használhatjuk, hogy könnyebb legyen olvasni a szabályos kifejezéseket, az `i` megszorítást pedig arra, hogy mintánk kis- és nagybetűktől függetlenül illeszkedjék a szövegre. A `/ k \S* r /ix` kifejezés például illeszkedik a `"kar"` és `"KAR"` szavakra, de a `"K A R"` szóra már nem. Az `x`-szel módosított szabályos kifejezések váltókarakter nélküli szóközei nem fognak illeszkedni semmire sem a forrásláncban, a különbség csupán esztétikai lesz.

Az `m` paraméter akkor jöhet jól, ha egy szöveg több sorában szeretnénk hivatkozási pontos mintára illeszteni. A `^` és `$` minták alapértelmezés szerint a teljes karakterlánc elejét és végét jelzik. A következő kódrészletben az `m` paraméterrel módosítjuk a `$^` viselkedését:

```
$szoveg = "név: péter\nfoglalkozás: programozó\nszeme: kék\n";
preg_match_all( "/^\w+:\s+(.*)$/m", $szoveg, $tomb );
foreach ( $tomb[1] as $ertek )
    print "$ertek<br>";
// kimenet:
// péter
// programozó
// kék
```

Egy olyan szabályos kifejezést hozunk létre, amely olyan mintára illeszkedik, amelyben bármely szóalkotó karaktert vessző és tetszőleges számú szóköz követ. Ezután tetszőleges számú karaktert követő „karakterlánc vége” karakterre (`$`) illesztünk. Az `m` paraméter miatt a `$` az egyes sorok végére illeszkedik a teljes karakterlánc vége helyett.

Az s paraméter akkor jön jól, ha a . karakterrel szeretnénk többsoros karakterláncban karakterekre illeszteni. A következő példában megpróbálunk a karakterlánc első és utolsó szavára keresni:

```
$szoveg = "kezdetként indítsunk ezzel a sorral\nés így
 ➥ egy következtetéshez\n érkezhetünk el végül\n";
preg_match( "/^(\w+).*?(\w+)$/", $szoveg, $tomb );
print "$tomb[1] $tomb[2]<br>";
```


Ez a kód semmit nem ír ki. Bár a szabályos kifejezés megtalálja a szóalkotó karaktereket a karakterlánc kezdeténél, a . nem illeszkedik a szövegben található soremelésre. Az s paraméter ezen változtat:

```
$szoveg = "kezdetként indítsunk ezzel a sorral\nés így
 ➥ egy következtetéshez\n érkezhetünk el végül\n";
preg_match( "/^(\w+).*?(\w+)$/s", $szoveg, $tomb );
print "$tomb[1] $tomb[2]<br>";
// azt írja ki, hogy "kezdetként végül"
```

Az e paraméter különlegesen hasznos lehet számunkra, hiszen lehetővé teszi, hogy a preg_replace() függvény csereszövegét PHP kódként kezeljük. A függvényeknek paraméterként visszautalásokat vagy számokból álló listákat adhatunk át. A következő példában az e paraméterrel adunk át illesztett számokat egy függvénynek, amely ugyanazt a dátumot más formában adja vissza.

```
<?php
function datumAtalakito( $honap, $nap, $ev )
     {
     $ev = ($ev < 70 )?$ev+2000:$ev;
     $ido = ( mktime( 0,0,0,$honap,$nap,$ev) );
     return date("Y. m. j. ", $ido);
     }

$datumok = "3/18/99<br>\n7/22/00";
$datumok = preg_replace(  "/([0-9]+)\/([0-9]+)\/([0-9]+)/e",
                          "datumAtalakito(\\1,\\2,\\3)",
                          $datumok);
print $datumok;

// ezt írja ki:
// 1999. 03. 18.
// 2000. 07. 22.
?>
```

Három, perjelekkel elválasztott számhalmazt keresünk és a zárójelek használatával rögzítjük a megtalált számokat. Mivel az `e` módosító paramétert használjuk, a cserelánc paraméterből meghívhatjuk az általunk meghatározott `datumAtalakito()` függvényt, úgy, hogy a három visszautalást adjuk át neki. A `datumAtalakito()` csupán fogja a számbemenetet és egy jobban festő dátummá alakítja, amellyel aztán kicseréli az eredetit.

Összefoglalás

A szabályos kifejezések hatalmas témakört alkotnak, mi csak felületesen ismerkedtünk meg velük ezen óra során. Arra viszont már biztosan képesek leszünk, hogy bonyolult karakterláncokat találjunk meg és cseréljünk le egy szövegben.

Megismerkedtünk az `ereg()` szabályoskifejezés-függvénnyel, amellyel karakterláncokban mintákat kereshetünk, illetve az `ereg_replace()`-szel, amellyel egy minta összes előfordulását cserélhetjük le. A karakterosztályok használatával karakterláncokat, a mennyiségjelzők használatával több mintát, az elágazásokkal mintákat vagylagosan kereshetünk. Részmintákat is kigyűjthetünk és rájuk visszautalásokkal hivatkozhatunk. A Perl típusú szabályos kifejezésekben váltókarakterekkel hivatkozási pontos illesztést kérhetünk, de karakterosztályok illesztésére is rendelkezésre állnak váltókarakterek. Ezenkívül a módosító paraméterekkel képesek vagyunk finomítani a Perl típusú szabályos kifejezések viselkedésén.

Kérdések és válaszok

A Perl típusú szabályos kifejezések nagyon hatékonynak tűnnek. Hol található róluk bővebb információ?

A szabályos kifejezésekről a PHP weboldalán (`http://www.php.net`) találhatunk némi felvilágosítást. Emellett még a `http://www.perl.com` oldalain találhatunk információt, a Perl típusú szabályos kifejezésekhez pedig a `http://www.perl.com/pub/doc/manual/html/pod/perlre.htm` címen és Tom Christiansen cikkében, a `http://www.perl.com/pub/doc/manual/html/pod/perlfaq6.html` címen találhatunk bevezetést.

Műhely

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

Kvíz

1. Melyik függvényt használnánk egy karakterláncban minta illesztésére, ha POSIX szabályos kifejezéssel szeretnénk a mintát meghatározni?

2. Milyen szabályos kifejezést használnánk, ha a `"b"` betű legalább egyszeri, de hatnál nem többszöri előfordulására szeretnénk keresni?

3. Hogyan határoznánk meg a `"d"` és az `"f"` betűk közé eső karakterek karakterosztályát?

4. Hogyan tagadnánk a 3. kérdésben meghatározott karaktertartományt?

5. Milyen formában keresnénk tetszőleges számra vagy a `"bokor"` szóra?

6. Melyik POSIX szabályos kifejezés függvényt használnánk egy megtalált minta lecserélésére?

7. A `.bc*` szabályos kifejezés mohón illeszkedik, azaz inkább az `"abc0000000bc"` karakterláncra illeszkedik, mint az `"abc"`-re. A Perl típusú szabályos kifejezésekkel hogyan alakítanánk át a fenti szabályos kifejezést úgy, hogy a megtalált minta legelső előfordulását keresse meg?

8. A Perl típusú szabályos kifejezéseknél melyik fordított perjeles karakterrel illesztünk elválasztó karakterre?

9. Melyik Perl típusú szabályos kifejezést használnánk, ha az a célunk, hogy egy minta összes előfordulását megtaláljuk?

10. Melyik módosító karaktert használnánk egy Perl típusú szabályoskifejezés-függvényben, ha kis- és nagybetűtől függetlenül szeretnénk egy mintára illeszteni?

Feladatok

1. Gyűjtsük ki az e-mail címeket egy fájlból. A címeket tároljuk egy tömbben és jelenítsük meg az eredményt a böngészőben. Nagy mennyiségű adattal finomítsuk szabályos kifejezéseinket.