Co jsou to regulární výrazy?

Regulárními výrazy se používají ke zpracovávání textových řetězců. Vytvoříme si tzv. masku a zjišťujeme, zda jí textový řetězec vyhovuje (např. zda se skládá ze samých čísel…). Zda zadaný řetězec odpovídá vzoru vyjádřeným regulárním výrazem.

http://wiki.jalakai.co.uk/include/preg.php

Použití v PHP?

Funkce **preg_match()** (ereg() se již dnes nepoužívá!)**.**
Při UTF-8 kódování pak především funkce **mb_ereg()**, **mb_eregi()** (totožné, jen mb_eregi() není citlivá na velikost písmen. Vrací true, vyhovuje-li řetězec dané masce.

Syntaxe:

```
mb_ereg (maska, řetězec)
```

Příklad:

```
if ( mb_ereg („den “, „Dobrý den “) )
  echo „řetězec obsahuje znaky ‚den ‘ “;
    else
      echo „řetězec neobsahuje znaky ‚den ‘ “;
```

```
if ( preg_match („/den/ “, „Dobry den “) )
```

```
if ( preg_match („/den/i “, „Dobry den “) )
 // nebude citlivá na velikost písmen
```

Je v řetězci obsažen podřetězec?

Maska skládající se z pouhých znaků hledá výskyt tohoto podřetězce v daném řetězci

maska:	`program`	
řetězec:	`dobrý den`	
	`programování`	

PŘÍKLAD: Je v řetězci obsažen podřetězec?

maska:	den	
řetězec:	dobrý den	
	denně se potkáváme	
	škola	

Tečka jako libovolný znak

Tečka zastupuje jakýkoliv jeden znak

maska:	gr.m	
řetězec:	program	
	grimasa	
	bagr může závodit	
	agronom	

PŘÍKLAD: Tečka jako libovolný znak

maska:	je.en	
řetězec:	jeden, dva, tři	
	Kdo je ten muž?	

maska:	pr....m	
řetězec:	složitý program	
	prodám počítač	

Jen vybrané znaky

Výběr z několika konkrétních znaků za pomoci hranatých závorek []

maska:	dobr[ýé]	
řetězec:	dobrý den	✓
	dobré jídlo	✓
	dobrou noc	✗

PŘÍKLAD: Jen vybrané znaky

maska:	á.[123]	
řetězec:	Kobra 11	
	Olomoucká 39	
	kódovaná 16-ti bity	

Příklad:

Napište regulární výraz, který potvrdí, že se v řetězci nachází alespoň jedna samohláska „*a*“, „*e*“, „*i*“, „*o*“, „*u*“, „*y*“.

Například:

strč prst skrz krk 

ahoj světe 

Řešení:

[aeiouy]

Jen vybrané znaky - rozsah

Často nemusíme do hranatých závorek vypisovat všechny znaky, které potřebujeme ošetřit, ale použít rozsah

Například:

```
[0-9]  místo  [0123456789]
[a-z]  místo  [abcdefghijklmnopqrstuvwxyz]
[A-Z]  místo  [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
```

Lze kombinovat:

```
[0-9a-dA-D]  místo  [0123456789abcdABCD]
```

Příklad:

Napište regulární výraz, který vyhodnotí, zda se v řetězci nachází časový údaj ve tvaru HH:MM.

Například: 16:15
08:59
00:01

Řešení:

```
[0-2][0-9]:[0-5][0-9]
```

Negace

Chceme-li vybrat znaky, které naopak nechceme, můžeme použít **negaci**. Značí se stříškou **^** a píše se hned za otevírací hranatou závorku, např. [^abc]

maska:	[^A-Z]
řetězec:	
a	
ABC	
AbC	

PŘÍKLAD: Negace

maska:	[^1][^][0-3]	
řetězec:	123	✗
	953	✓
	9 3	✗
	-123	✓

Jak zapsat speciální znaky jako je např. tečka?

Pokud mají znaky speciální význam a my jej chceme pro daný případ potlačit, připíšeme před ně zpětné lomítko \

maska:	AtdW.	
řetězec:	jablko, hruška atd.	
	jablko, hruška atd atd	

Příklad č. 1:

Napište regulární výraz, který vyhodnotí, zda řetězec obsahuje tři tečky.

Například:

leden, únor, březen…

Řešení:

\.\.\.

Příklad č. 2:

Napište regulární výraz, který vyhodnotí, zda se v řetězci nachází číslice v hranatých závorkách

Například:

[5]

Řešení:

₩[[0-9]₩]

Co když napíšeme do hranatých závorek metaznaky plnící speciální funkci?

Pokud je ve výčtu (v hranatých závorkách) uvedena **tečka**, je brána jako normální znak . a se zpětným lomítkem se nepíše.

Pokud je ve výčtu uvedena **stříška** ^ na jiné pozici než ihned za otevírací hranatou závorkou [, je rovněž brána jako normální znak.

maska:	`[.#^][a-z]`	
řetězec:	`^x`	✓
	`z-(x.y)`	✓

Počátek a konec

Pokud chceme ověřovat znaky na začátku řetězce, použijeme na počátku stříšku ^. Pro určení konce řetězce zase na konci dolar $.

maska:	^den$	
řetězec:	den	
	dobrý den	

PŘÍKLAD: Počátek a konec

maska:	[0-9]$	
řetězec:	náměstí Svobody	
	náměstí Svobody 18	
	agent 007	
	1	

Opakování výrazu pomocí hvězdičky I.

Pomocí kvantifikátoru * bude regulární výraz **opakován kolikrát to jen bude možné** (0x až n-krát)

maska:	[0-9]*	
řetězec:	646541216546541231123	✓
	9. třída	✓
	dobrý den	✓

PŘÍKLAD 1: Opakování výrazu pomocí * I.

Napište regulární výraz, který potvrdí, že se v řetězci nacházejí samotná čísla (nebo je prázdný)

Řešení:

```
^[0-9]*$
```

PŘÍKLAD 2: Opakování výrazu pomocí * I.

maska:	^[0-9][^0-9]*$	
řetězec:	1 slon	
	10 slonů	
	1 slon a 1 zebra	
	1	

Opakování výrazu pomocí hvězdičky II.

Zápis `.*` znamená libovolný řetězec znaků

maska:	`^[0-9].*[0-9]$`	
řetězec:	`0 jedna 2`	✓
	`0 jedna dvě`	✗
	`01`	✓
	`0`	✗

PŘÍKLAD 1: Opakování výrazu pomocí * II.

Napište regulární výraz, který potvrdí, že se v řetězci nachází věta – začíná velkým písmenem nebo číslem a končí tečkou. (Pomiňte, že vět v řetězci může být víc za sebou.)

Například:

Ahoj světe.

Řešení:

```
^[A-Z0-9].*₩.$
```

Opakování pomocí plus

Pomocí kvantifikátoru + bude regulární výraz **opakován minimálně jedenkrát** (1x až n-krát)

maska:	`[0-9]+`	
řetězec:	`Kill Bill`	✗
	`Kill Bill 2`	✓

PŘÍKLAD: Opakování pomocí plus

Napište regulární výraz, který ověří, že se zadaná přezdívka skládá jen z malých písmen bez diakritiky, která mohou být rozdělena podtržítkem _. Délka přezdívky je minimálně 2 znaky a podtržítko nemůže být na první pozici.

Například:

jirka		_jirka	
jirka_novak		Jirka_Novak	

Řešení:

```
^[a-z][a-z_]+$
```

Opakování pomocí otazníku

Pomocí kvantifikátoru ? bude regulární výraz **opakován maximálně jednou** (0x nebo 1x)

maska:	^škola?$	
řetězec:	škola	
	škol	
	střední škola	

PŘÍKLAD: Opakování pomocí otazníku

maska:	`pr?.*gram`	
řetězec:	`programátor`	✓
	`pentagram`	✓
	`pět gramů`	✓

Určený počet opakování I.

Chceme-li vymezit **přesný počet opakování**, použijeme zápis {x}

maska:	^x{2}$	
řetězec:	x	
	xx	
	xxx	
	xxxx	

Určený počet opakování II.

Chceme-li vymezit rozsah, **kolikrát se může výraz maximálně opakovat**, použijeme složené závorky `{x,y}`

maska:	`^x{2,3}$`	
řetězec:	x	
	xx	
	xxx	
	xxxx	

Určený počet opakování III.

Chceme-li vymezit rozsah způsobem **„minimálně x-krát“**, použijeme zápis `{x,}`

maska:	`^x{2,}$`	
řetězec:	x	
	xx	
	xxx	
	xxxx	

PŘÍKLAD 1: Určený počet opakování

maska:	`mate{0,3}matika`	
řetězec:	`matematika`	
	`automat a automatika`	
	`mate mě matematika`	

PŘÍKLAD 2: Určený počet opakování

Napište regulární výraz, který ověří, že zadaný řetězec má minimálně 4 znaky.

Řešení:

```
^.{4,}$
```

Jedno, nebo druhé

Použitím svislé čáry |, která má význam logického OR, bude platit jen určitá hodnota z daného výčtu.

maska:	^a|b|c$	
řetězec:		
	a	
	b	
	c	
	d	

Seskupování pomocí kulatých závorek

Kulaté závorky () používáme k vytvoření podvýrazu.
Dojde k seskupení jednotlivých částí řetězce
a doplněné kvantifikátory * + ? se pak budou vztahovat
na celý podvýraz.

maska:	(kočka|pes)+	
řetězec:	Na dvorku si hraje kočka.	✓
	Na dvorku si hraje pes.	✓
	Na dvorku si hrají kočky a psi.	✗
	Na dvorku si nikdo nehraje.	✗

PŘÍKLAD: Seskupování pomocí kulatých…

Napište regulární výraz, který ověří, že jsou v řetězci pouze čísla v rozsahu 0 – 99 a za každým číslem se nachází čárka.

Například:

1,8,

11,1,22,2,

Řešení:

```
^([0-9]{1,2},)+$
```

Začátek a konec slova

Zda stojí slovo samostatně můžeme ověřit pomocí metaznaků [[:<:]] a [[:>:]], které označují začátek a konec slova. Samozřejmě je možné použít každý z nich nezávisle.

maska:	[[:<:]]svět[[:>:]]	
řetězec:	svět kolem nás	
	ahoj světe	
	Takový je svět.	

Třídy znaků I.

Třídy jsou skupiny znaků, které pro snadnější práci s regulárními výrazy přednastavili sami autoři jazyka PHP.

Syntaxe:

```
[[:název třídy:]]
```

Jednotlivé třídy:

alnum	písmena anglické abecedy a desítkové číslice
alpha	písmena anglické abecedy

Třídy znaků II.

lower	malá písmena anglické abecedy
upper	velká písmena anglické abecedy
digit	čísla (desítková soustava)
xdigit	čísla (šestnáctková soustava)
punct	interpunkční znaménka a další znaky (závorky, zavináče atd.)
blank	mezera a tabulátor
space	prázdné znaky (mezera, tabulátor, nová řádka, nová stránka atd.)

Třídy znaků III.

cntrl	řídící znaky (\n, \t atd.)
print	tisknutelné znaky
graph	tisknutelné znaky bez mezer

Ukázkový příklad na třídu znaků

Regulární výraz pomocí tříd, který ověří, že řetězec je číslo desítkové soustavy.

```
^[[:digit:]]+$
```

PŘÍKLAD: Třídy znaků

Napište **základní** regulární výraz, který zkontroluje platnost e-mailové adresy

Řešení:

```
^[a-zA-Z0-9.]+@[[:alnum:]]+\.[[:alnum:]]{2,4}$
```

OBECNÝ PŘÍKLAD Č. 1:

Napište regulární výraz, který ověří, že uživatelem zadané znaky splňují tyto nároky na heslo:

- pouze číslice a malá písmena
- minimální délka 4 znaky
- první 3 znaky musejí být číslice

Řešení:

```
^[0-9]{3}[a-z0-9]+$
```

OBECNÝ PŘÍKLAD Č. 2:

Napište regulární výraz, který ověří, že uživatel zadal číslo (i z více číslic) a pokud ano, tak že nezačíná nulou nebo nulami.

Například:

42 

042 

Řešení:

```
^[1-9]{1}[0-9]*$
```

OBECNÝ PŘÍKLAD Č. 3:

Napište regulární výraz, který potvrdí, že se v řetězci nenachází žádná mezera.

Řešení:

```
^[^ ]*$
```

3. parametr u funkcí mb_ereg() a mb_eregi()

Třetím, nepovinným parametrem těchto funkcí je pole shod. Do něj se uloží ty části řetězce, které vyhovují danému podvýrazu regulárnímu výrazu (podvýraz je v kulatých závorkách). V indexu [0] pole se uloží ta část řetězce, která vyhovuje celému regulárnímu výrazu.

Například:

```
<?
 if (mb_ereg("(ahoj) světe (veliký)","ahoj světe veliký",$pole))
   print_r($pole);
// vypíše: Array ([0]=>ahoj světe veliký [1]=>ahoj [2]=>veliký)
?>
```

PŘÍKLAD: Třetí parametr u funkcí…

Doplňte regulární výraz, díky kterému bude možné vypsat první číslo (i z více číslic), které se v řetězci objeví.

```
<?
 if ( mb_ereg („.............","James Bond, agent 007.",$pole) )
   echo $pole[1];
?>
```

V tomto případě bude vypsáno: *007*

Řešení:

```
^[^0-9]*([0-9]*).*$
```

mb_ereg_replace() a mb_eregi_replace() I.

Tyto funkce se opět liší jen citlivostí na velikost písmen.
Mají tuto syntaxi:

mb_ereg_replace(regulární výraz, náhrada, pův.řetězec)

Příklad:

```
<?
  $a = mb_ereg_replace ("(ahoj) světe", "nazdar", "ahoj světe");
  echo $a;
// vypíše: nazdar
?>
```

mb_ereg_replace() a mb_eregi_replace() II.

Ve druhém parametru určujícím nahrazovaný řetězec lze použít metaznak **\\číslice** určující podvýraz regulárního výrazu.

Například:

```
<?
 $a = mb_ereg_replace ("(ahoj) (světe) (veliký)", "\\3 \\2 \\1",
                       "ahoj světe veliký");
  echo $a;
  // vypíše: veliký světe ahoj
?>
```

PŘÍKLAD: Funkce ereg_replace() a …

Pomocí funkce **mb_eregi_replace** nahraďte v textu *[B]* a *[/B]* za HTML tagy *<B>* a *</B>*. Například:

„následující bude tučně: [B]výraznější text[/B]“

změnit na

„následující bude tučně: <B>výraznější text</B>“

Řešení:

```
<?
   $text = "následující bude tučně: [B]výraznější text[/B]";
   echo mb_ereg_replace ("\[B\](.*)\[/B\]*", "<B>\\1</B>", $text);
?>
```

pokračování…

Hladovost

… daný příklad sice bude u naší testovací věty fungovat, nicméně stačilo by do řetězce doplnit další prvky pro zvýraznění textu a výraz nesplní, co očekáváme:

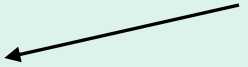
následující bude tučně: [B]výraznější text[/B] a další [B]tučný[/B]

Při použití stejného regulárního výrazu výsledkem bude:

následující bude tučně: <B>výraznější text[/B] a další [B]tučný</B>

Důvod je prostý – regulární výraz se snaží obsáhnout co nejvíce znaků z řetězce a za koncové [/B] tak vezme až

následující bude tučně: [B]výraznější text[/B] a další [B]tučný[/B]



Tuto skutečnost je třeba mít při sestavování výrazů na paměti.

Funkce split() a spliti()

Tyto funkce umějí rozdělit řetězec na několik částí a ty pak uložit do jednotlivých prvků pole. Mají tuto syntaxi:

split (regulární výraz, řetězec, [limit])

Řetězec je rozložen podle *regulárního výrazu*
(na maximální počet částí, které udává *limit*)

Například (rozdělení řetězce po slovech a uložení do pole):

```
<?
  $pole = split ("[[:blank:]]+", „ahoj světe veliký");
  print_r ($pole);
  // vypíše: Array ( [0] => ahoj [1] => světe [2] => veliký )
?>
```