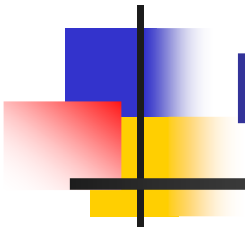
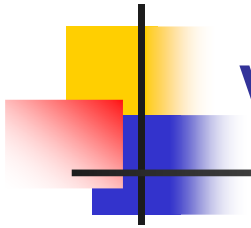


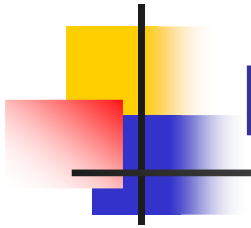
PHP – znakové sady a kódování

Záludnosti práce s různými znakovými sadami v PHP.



Výběr znakové sady

- Znaková sada by měla odpovídat zvolenému jazyku.
  - Nebo je možné použít UTF-8 a vyhnout se tak komplikacím.
    - Na druhou stranu nějaké komplikace přibudou…
- Ve vybrané znakové sadě by měly být kódovány všechny skripty a jiné soubory související s webem.
  - Všechna data by se měla hned při načítání konvertovat do zvolené sady.
- Zvolené kódování je třeba oznámit uživateli.
  - Nejlépe přidáním příslušné HTTP hlavičky do odpovědi.
    `header('Content-Type: text/html; charset=utf-8');`
  - V žádném případě nepoužívejte tag **`META http-equiv`**.

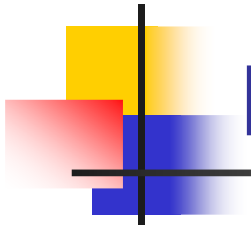


Práce s řetězci v multi-byte kódu

- Některá kódování (např. UTF-8, UCS-2, …) potřebují na jeden znak více bytů (na rozdíl od ASCII, Latin2, apod.).
  - Standardní funkce pro práci s řetězci s tím nepočítají.

- V PHP existuje knihovna "Multibyte string functions", která si poradí i se složitějším kódováním.
  - Má většinu standardních funkcí, ale s prefixem **`mb_`** (**`mb_strlen()`**, **`mb_strpos()`**, **`mb_ereg()`**, …).
  - Umí i konverzi kódování – **`mb_convert_encoding()`**.
  - Je vhodné si na začátku skriptu nastavit, jaké kódování bude knihovna interně používat (**`mb_internal_encoding()`**).

Příklad 1

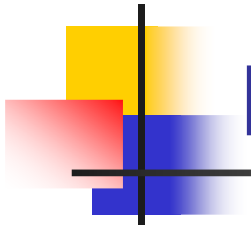


Další záludnosti

- Kódování vstupních dat z HTTP.
  - V konfiguraci lze aktivovat transparentní překódování na vnitřní reprezentaci (viz sekce "mbstring" v php.ini).
  - Případně to lze dělat ručně (**`mb_parse_str()`**).

- Při kódování je třeba myslet i na ostatní zdroje dat.
  - Např. v MySQL lze nastavit kódování komunikace s databází:

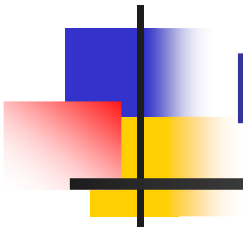
```
SET NAMES utf8
SET CHARACTER SET utf8
```



Další záludnosti

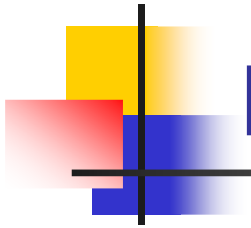
- Lexikografické porovnávání řetězců.
  - Nejlepší je nechat takové věci na úložišti dat (DB apod.).
  - Funkce **`strcmp()`** je binárně bezpečná.
  - Musí být správně nastavený jazyk porovnávání (**`setlocale()`**).

- Knihovna "iconv".
  - Alternativa k Multibyte string functions.
  - Obsahuje méně funkcí.



PHP – regulární výrazy

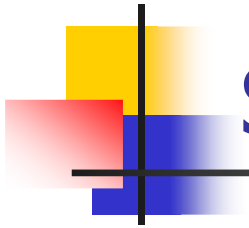
Syntaxe a funkce pro práci s regulárními výrazy.



Regulární výrazy

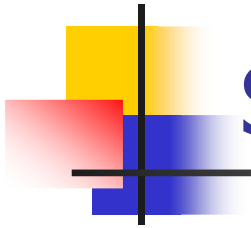
- Regulární výrazy
  - Vzory, se kterými se porovnávají (matchují) řetězce.
  - V PHP jsou dva typy reg. výrazů – POSIX a Perl.
    - Každý typ má vlastní funkce.
    - Zde se budeme zabývat pouze POSIXovými.

- S reg. výrazy můžeme provádět následující operace:
  - Porovnávání řetězce a vzoru, případně výběr podřetězce dle vzoru.
    - **`ereg()`**, **`eregi()`**, **`mb_ereg()`**, **`mb_eregi()`**
  - Nahrazení výskytů regulárního výrazu v řetězci.
    - **`ereg_replace()`**, **`eregi_replace()`**, **`mb_ereg_replace()`**
  - Dělení řetězce podle výskytů regulárního výrazu.
    - **`split()`**, **`spliti()`**, **`mb_split()`**, **`mb_spliti()`**.



Syntaxe regulárního výrazu

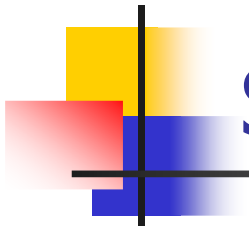
- Regulární výraz (regexp) je řetězec se spec. syntaxí.
  - Syntaxe odpovídá standardu POSIX Extended.
- Nejjednodušší reg. výraz je „obyčejný“ řetězec.
  - „Obyčejný“ řetězec neobsahuje žádné řídicí znaky.
    - Řídicí znaky je možné vložit jako obyčejné, pokud před nimi uvedeme zpětné lomítko.
  - Zadaný výraz se hledá jako podřetězec.
    - Např. regexpu "**abc**" bude odpovídat libovolný řetězec obsahující sekvenci "**abc**".
- Výraz lze ukotvit na začátku, nebo na konci řetězce.
  - "**^abc**" – řetězce začínající sekvencí "**abc**"
  - "**abc$**" - řetězce končící "**abc**"



Syntaxe regulárního výrazu

- Uvnitř výrazu je možné použít zástupný symbol '**.**'.
  - Nahrazuje libovolný znak.
  - Např. "**.ůl**" odpovídá slovům kůl, vůl, hůl, půl, sůl, důl, fůl, …

- Obdobně můžeme nabídnout množinu zástupných znaků:
  - Povolené znaky se zapisují do **[]**.
    - Např. "**[khs]ůl**" odpovídá slovům kůl, hůl a sůl, ale už ne půl, Hůl, …
  - Kromě výčtu můžeme znaky zadat i rozsahem.
    - Např. "**[a-zA-Z0-9]**" odpovídá libovolnému písmenu nebo číslici.
  - Pokud za **[** následuje **^**, použije se doplněk množiny.
  - Existují také speciální entity. Např.:
    - **[:alpha:]** – libovolný znak (bere v úvahu národní abecedy)
    - **[:blank:]** – bílý znak



Syntaxe regulárního výrazu

- Opakování atomů.
  - Jeden znak, symbol '**.**', množina a výraz uzavřený do závorek se označuje atom.
  - Atomy je možné nechat opakovat: <*atom*>**{***rozsah***}**
  - Rozsah může být jedno číslo **{***počet***}**, interval **{***min***, *** *max***}**, nebo otevřený interval do nekonečna **{***min***,}**
  - Existují zástupné symboly pro často používané intervaly:
    - **?** znamená **{0,1}**
    - ***** znamená **{0,}**
    - **+** znamená **{1,}**
- Výraz může obsahovat několik větví oddělených **|**.
  - Jednotlivé větve jsou alternativy (spojené logickým "nebo").

Vyhledávání podřetězců

- Funkce ***ereg*** umí i vyhledat podřetězce.
  - Jako podřetězec se bere cokoliv, co je ve výrazu uzavřeno do **()**.
  - Podřetězce se uloží do pole.
    - V pořadí, ve kterém jsou otevírací závorky v regulárním výrazu.

- Příklad:

1. 2. 3. 4.

```
$pattern = '^([0-9]{1,2}):([0-9]{2})(:([0-9]{2}))?$';
if (ereg($pattern, $str, $regs)) { ... }
```

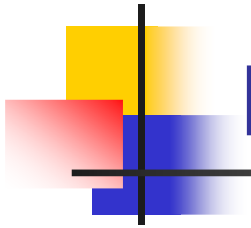
Pro `$str = '2:42'` bude:

```
$regs[0] = '2:42'
$regs[1] = '2'
$regs[2] = '42'
```

Nultá položka vždy obsahuje, co se namatchovalo na celý výraz.

Pro `$str = '2:42:54'` bude:

```
$regs[0] = '2:42:54'
$regs[1] = '2'
$regs[2] = '42'
$regs[3] = ':54'
$regs[4] = '54'
```



Nahrazování a dělení řetězce

- Funkce ***ereg_replace*** nahradí všechny výskyty výrazu jiným řetězcem.
  - V náhradním výrazu se lze odkazovat na nalezené podřetězce přes `\\`*`číslice`*, kde číslice je index podvýrazu (0-9). Např.:

    ```
    $s = ereg_replace('([0-9]+)', '\\1,- Kč', $s);
    ```

    - Udělá z každého čísla v textu cenu v korunách.

- Funkce ***split*** rozdělí řetězec na tokeny.
  - Zadaný regulární výraz použije jako oddělovač.
  - Např. `split('[[:blank:]]+', 'jedna dva  tři');` rozdělí řetězec podle bílých znaků a vrátí pole se třemi prvky.

Příklad 2