20. Session v PHP

Protokol HTTP, který slouží ke komunikaci mezi www serverem a prohlížečem je **bezstavový**. Tzn., že mezi jednotlivými přechody stránek **se neudržuje žádné spojení.** Když kliknete na odkaz, pouze se spojí klient se serverem, server pošle stránku a spojení se ukončí. Pokud ale potřebujete znát obsah hodnoty proměnné, kterou uživatel odeslal formulářem asi tak o 3 stránky dříve, bez session se neobejdeme.

Jaká je hlavní činnost session:

1. **identifikaci uživatele**
2. **uchovávání obsahu proměnných**

Pro identifikaci uživatele stačí správně identifikovat opětovné přístupy od téhož uživatele. Samotná data přiřazená ke konkrétnímu uživateli již není třeba mezi stránkami sdílet, ale lze je ukládat na server a odkazovat se na ně.
Na tomto principu jsou založeny sessions. Při jejich vytváření je uživateli vygenerován unikátní identifikátor sloužící k jeho rozpoznání a umožňující přistupovat ke konkrétnímu datovému souboru. Pro sdílení mezi jednotlivými stránkami tento identifikátor uložíme do cookie, kterou prohlížeč na server odesílá při každém načítání stránky. Data samotná jsou defaultně uložena v serializované podobě v souboru uloženém na serveru a pojmenovaném podle uvedeného identifikátoru.

Příklad http requestu (vyžádání nějaké URL adresy):

```
GET /wiki/Wikipedie HTTP/1.1
Host: cs.wikipedia.org
Accept-Charset: UTF-8,*
```

Zde vidíme zcela běžnou obecnou hlavičku, kde není spuštěn příkaz (funkce session) a server odpovídá anonymnímu uživateli.

```
GET /programujeme-v-php/sessions?rev=1 HTTP/1.1
Host: pehapko.cz
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en,cs;q=0.8
Cookie: PHPSESSID=9g0jg1mgs26c1i0e4hok7rgbd1; nette-browser=spx2kdfq2y
```

V druhém případě již hlavička obsahuje záznamy o cookies, vidíme tady hned dvě. Cookie s název PHPSESSID je klíč k identifikaci uživatele. Cookie nette-browser není tak důležitý. Identifikuje prohlížeč.

Pokud není na serveru nastavený session.auto_start (což ve výchozím stavu není), zahájíme session relaci funkcí **session_start()**. Kvůli nastavování cookie musíme tuto funkci volat před jakýmkoliv výstupem, typicky zcela na začátku souboru (až na výjimky…). V této chvíli již můžeme začít pracovat se superglobálním polem $_SESSION (dříve $HTTP_SESSION_VARS).

Hodnotu do session přiřadíme běžným operátorem přiřazení $_SESSION[‚time‘] = time(). Přečíst (s vypsáním) ji můžeme standardní konstrukcí echo $_SESSION[‚time‘]. Obecně lze říci, že se session pracujeme jako s běžným polem, můžeme používat in_array, isset, unset a podobně všechny funkce pro práci s poli, jak byly uvedeny v díle o polích. Jediný rozdíl při práci mezi session a běžnými poli je jejich superglobálnost, tj. **jsou viditelé v libovolné funkci**.
Pozn. Na rozdíl od nastavení cookie pomocí setcookie() se změna session projeví ještě na téže stránce (změněná cookie až po reloadu, protože obvykle neupravujeme přímo $_COOKIE).

Příklad základní práce se session – čas strávený na stránce a další statistiky

```
<?php
session_start();

$firstVisit = false;
if (!isset($_SESSION['firstVisit']))
{
    $_SESSION['firstVisit'] = new \Datetime;
    $_SESSION['lastVisit'] = new \Datetime;
    $_SESSION['count'] = 0;
    $firstVisit = true;
}
$timeSum = (new \Datetime)->diff($_SESSION['firstVisit']);
$timeLast = (new \Datetime)->diff($_SESSION['lastVisit']);
$lastVisit = $_SESSION['lastVisit'];
$_SESSION['lastVisit'] = new \Datetime;
$_SESSION['count']++;

echo ($firstVisit ? 'Vítejte, jste u nás poprvé.' : 'Děkujeme, že se k nám vracíte.') . PHP_EOL;
echo '<br>Čas první návštěvy: ' . $_SESSION['firstVisit']->format('H:i:s - d.m.Y') . PHP_EOL;
echo '<br>Celkový čas strávený na našich stránkách: '. $timeSum->format('%H:%i:%s') . PHP_EOL;
echo '<br>Čas předchozí návštěvy: ' . $lastVisit->format('H:i:s') . PHP_EOL;
echo '<br>Čas od předchozí návštěvy: ' . $timeLast->format('%H:%i:%s') . PHP_EOL;
echo '<br>Počet návštěv: ' . $_SESSION['count'] . PHP_EOL;
?>
```

Vyzkoušejte si tento příklad takto:

- Aktualizacemi téže stránky se budou měnit časové údaje. Aplikace si pamatuje svoji historii.
- Otevřete si stránku ve dvou různých prohlížečích (nebo v jednom + anonymním režimu) a paralelně jednotlivé stránky aktualizujte. Ověřte, že se jednotlivé relace neovlivňuji.
- Naopak otevřete stránku ve dvou běžných oknech prohlížeče (sdílejících cookies). Zjistíte, že nyní se relace vzájemně ovlivňují.

- Můžete vytvořit několik (volitelně odkazy provázaných) stránek a includovat do nich výše uvedený kód. Zjistíte, že se session sdílí mezi stránkami (session_start() musí být na každé stránce).
- Můžete si měnit jednotlivé cookie, když je smažete, bude vaše návštěva označena jako první. Zakážete-li ve svém prohlížeči přijímat cookies, pokaždé bude návštěva označena jako první.
- Výchozí nastavení pamatování si session bývá 24 minut (1440 sekund), nechte relaci delší dobu neaktivní a pak zkuste aktualizovat. Zřejmě uvidíte, že se v relaci pokračuje a nic se nestalo, mazání starých záznamů je totiž pravděpodobnostní. Existence po dobu 24 minut od poslední aktualizace je zaručena. Smazání po této době nikoliv, to závisí na aktivitě garbage collectoru, která je při výchozím nastavení 0.01, tj. s pravděpodobností 1 % při každé inicializaci smaže staré session záznamy. **Na některých systémech (Debian, Ubuntu) bývá gc zcela neaktivní a mazání periodicky provádí cronová úloha.

Demo k testování.

Zdroj: http://www.pehapko.cz/programujeme-v-php/sessions

---

21. Počítadlo přístupů v PHP

Toto jednoduché počítadlo přístupů v PHP se skládá z několika částí.

V první části příkazem session_start() zajistíme, aby se započítal z naší adresy pouze jeden přístup (tedy alespoň co se týče defaultní doby, po kterou si nás server pamatuje)

Počet přístupů se zapisuje do textového souboru, umístěného např. ve stejné složce, jako samotné počítadlo. V tomto případě se jedná o soubor pocitadlo.txt, který je potřeba si vytvořit.

V druhé části se právě příkazem fopen tento soubor „otevírá“ a parametrem „r“ zajistíme, aby z něj skript mohl číst (více o funkci fopen).

Dále funkcí fgets přečteme data souboru pocitadlo.txt. Číslice 1000 je maximální počet znaků. Pokud tedy budeme předpokládat vyšší počet, je potřeba číslo zvětšit (více o funkci fgets).

Po přečtení je vhodné soubor zase zavřít – fclose (více o funkci fclose).

Nyní je v paměti tedy uloženo číslo, které reprezentuje posledního návštěvníka a následně jej potřebujeme zvětšit o jedno (+1) a nechat vypsat.

V poslední části je kontrola stavu session a v případě, že se jedná o novou návštěvu, je tento stav opět za pomoci otevření souboru pocitadlo.txt s parametrem „w“ (právo zápisu) zapsán a následně je soubor uzavřen.

```
<?php
 session_start();

// Otevření souboru pocitadlo.txt ke čtení
$data = fopen("pocitadlo.txt","r");
$pocet = fgets($data,1000);
fclose($data);
$pocet=$pocet + 1 ;
echo "$pocet" ;
echo " přístupů" ;
echo "\n" ;

if(!isset($_SESSION['hasVisited'])){
$_SESSION['hasVisited']="yes";
// Otevření souboru pocitadlo.txt k zápisu
$data = fopen("pocitadlo.txt","w");
fwrite($data, $pocet);
fclose($data);
}
?>
```

Demo počítadla si můžete vyzkoušet.

Základy informatiky, HTML, CSS, PHP, JavaScript, grafika a další

Na tomto portále najdete studijní materiály pro studijní obor Informační technologie pro předmět Webové aplikace, ale i další.

Veškeré návody i kódy jsou volně šiřitelné a nepodléhají žádné z nesčetných licencí.

Funkčnost i správnost jsou bez záruky.

Pokud je někde nějaká chyba, budu rád, pokud se o tom dozvím. Díky.

Martin Benda

Barevné modely

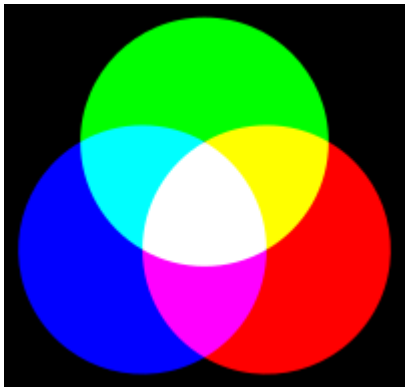
Barevný model RGB neboli červená-zelená-modrá je **aditivní způsob míchání**

barev.

Používá se především v barevných monitorech a projektorech (jde o míchání vyzařovaného světla), tudíž nepotřebuje vnější světlo (monitor zobrazuje i v naprosté tmě) na rozdíl např. od CMYK modelu.

Aditivní míchání barev

Jedná se o princip, kdy se při společném maximálním osvitu plochy všemi třemi barvami dosáhne bílé barvy.



Subtraktivní míchání barev

Opakem aditivního míchání barev je **subtraktivní míchání barev. Což** je způsob míchání barev, kdy se s každou další přidanou barvou ubírá část původního světla. Pokud například skládáme na sebe barevné filtry nebo mícháme pigmentové barvy, mícháme je subtraktivní metodou.

Základní barvy jsou: **žlutá, azurová, purpurová.**

Smícháním vždy dvou z těchto tří barev vznikne:

- smícháním žluté a azurové vznikne zelená
- smícháním žluté a purpurové vznikne červená
- smícháním purpurové a azurové vznikne modrá

Smícháním všech tří základních barev dostaneme barvu černou.

Tento princip je použit v tiskárnách. Z důvodu úspory pigmentu (inkoustu, toneru) při tisku tmavých odstínů se navíc používá i samostatný černý pigment

(CMYK).

RGBA model

Jedná se o **označení červené, zelené, modré a navíc tzv. Alpha kanálu**.

Alpha kanál

Alpha kanál je použit pro průhlednost. Je to složka pixelu udávající hodnotu průhlednosti tohoto pixelu. Bitové rozlišení alfa kanálu může být jen 1 bit, potom hovoříme o masce průhlednosti – pixel je buď 100% průhledný nebo neprůhledný. Jednobitová průhlednost může být použita například v obrázku ve formátu GIF. Pro vyšší bitové rozlišení lze již spočíst průhlednost pixelu, nejčastější bitové rozlišení je 8 bitů (model RGBA) a lze definovat 28 (= 256) úrovní průhlednosti pixelu. Nejtypičtějším příkladem bitmapy s průhledností je obrázek použitý jako ukazatel polohy na obrazovce počítače (kurzor). Alfa kanál používá například grafický formát PNG.

CMY a CMYK model

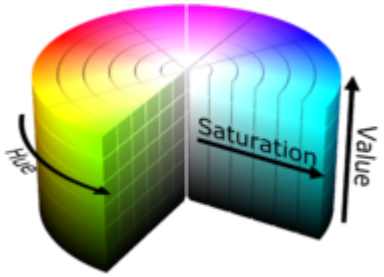
CMYK je barevný model založený na subtraktivním míchání barev. Používá se hlavně u reprodukčních zařízení (tiskáren), která vytvářejí barvy mícháním pigmentů. Model CMY obsahuje jen tři základní barvy – azurovou (Cyan), purpurovou (Magenta) a žlutou (Yellow). Jejich složením by měla vzniknout černá, ale při použití běžných tiskových barev není takto vzniklá černá příliš kvalitní. Proto se používá model CMYK, kde je navíc čtvrtá barva – černá (Key black). Jejím přidáním se navíc snižují náklady na tisk (černý pigment je levnější než barevný).

Všechny barvy vyjádřené v RGB nelze zobrazit v CMYK a naopak. Důvodem jsou rozdílné barevné trojúhelníky (gamuty). Nastává tedy problém s tiskem fotografií, hlavně se ztrátou brilance barev – barvy na monitoru budou vypadat jinak, než barvy na papíře.

HSV a HSB model

HSV (**Hue – barevný tón, Saturation – sytost barvy, Value – hodnota jasu**), někdy také HSB (Hue, Saturation, Balance) je barevný model odpovídající lidskému intuitivnímu popisu barev.

Tento model se nepoužívá pro ukládání fotografií, ale má dobré uplatnění při jejich editaci. Podle HSV se zadávají barvy, ovládá se saturace a přebarvuje obraz.



HSL model

Model HSL (**Hue – barevný tón, Saturation – sytost barev, Lightness – světlost**) je velmi podobný HSV. Zavedla jej firma Tektronix a podařilo se jí odstranit některé nedostatky HSV. Sytost leží na vodorovné ose, světlost na svislé ose a barevný tón představuje úhlová hodnota. Tvar modelu odpovídá skutečnosti – schopnost rozlišování barevných odstínů skutečně klesá se ztmavováním a zesvětlováním základní čisté barvy, zvyšování a snižování světlosti barvy skutečně spočívá v přidávání světlého nebo tmavého pigmentu.

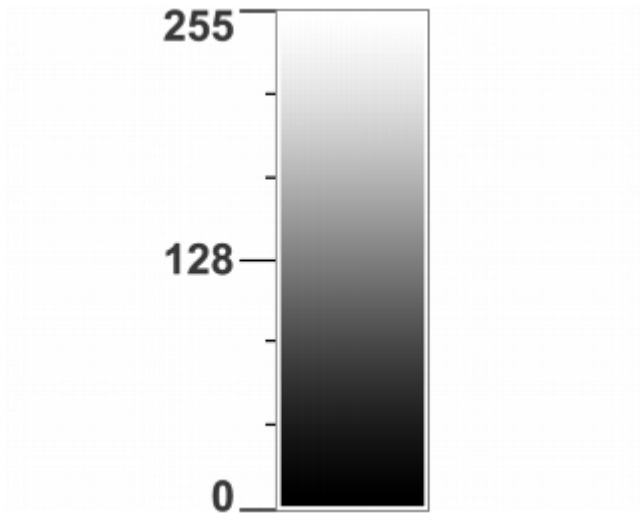
YUV model

Model YUV se používá v televizním vysílání v normě PAL a HDTV.

Y označuje jasovou složku a U a V chrominanci – barevnou složku. Vznikl, když bylo potřeba vytvořit způsob přenosu barevného signálu, který by byl kompatibilní s černobílým vysíláním. K stávající jasové složce byla přidána složka barevná. Výhodou YUV je oddělení jasové složky, kterou člověk přesněji vnímá. Pak je možné vyhradit pro chromatickou složku menší šířku přenosového pásma. Barevné složky se používají v rozsahu od −0.5 do +0.5, jasová složka má rozsah od 0 do 1.

Barevný model Stupně šedé

Barevný model Stupně šedé definuje barvu prostřednictvím jediné komponenty, světlosti, které se udává v rozsahu hodnot od 0 do 255. Každá z barev stupňů šedé má stejné hodnoty pro složky červená, zelená a modrá z barevného modelu RGB. Převedením barevné fotografie do stupňů šedé vytvoříte černobílou fotografii.



Barevná schémata

Nejen při návrhu webových stránek, ale čehokoli, kde se používají barvy (počítačové aplikace, návrh interiéru atd.), potřebujeme najít vhodné barevné schéma – tedy sadu barev, které mají produkovat co nejlepší celkový dojem. Ne všechny barvy však mohou koexistovat, některé kombinace působí nepříjemně či rušivě.

V praxi se nejčastěji používají tyto barevná schémata:

- **černobílé**
- **monochromatické** – používají jednu barvu, ale různé její odstíny

- **analogové** barevné schéma – toto schéma tvoří základní barva a její barvy sousední, ideálně hodnoty mezi 15-30°

- **kontrastní** barevné schéma – základní barva je doplněna o svůj doplněk (barvu přesně na opačné straně barevného kola).

- **triadické** barevné schéma – schéma je tvořeno odstíny rovnoměrně rozmístěnými kolem barevného středu

- **Dvojkontrast – tetráda** – schéma je tvořeno dvojicí barev a jejich doplňky. Vychází z tzv. tetrády – tedy čtveřice barev rozmístěných rovnoměrně na čtvrtinách kruhu (po 90°).

Barevná hloubka

Barevná hloubka popisuje počet bitů použitých k popisu určité barvy pixelu v bitmapovém obrázku nebo rámečku videa. Jinak také počet bitů na pixel, zejména je-li uvedeno spolu s počtem použitých pixelů. Větší

barevná hloubka zvětšuje škálu různých barev a přirozeně také paměťovou náročnost obrázku či videa.

Používané barevné hloubky

- 1bitová barva (2 barvy) – také označováno jako Mono Color (nejpoužívanější je, že bit 0 = černá a bit 1 = bílá)
- 4bitová barva (16 barev)
- 8bitová barva (256 barev)
- 15bitová barva (32 768 barev) také označováno jako Low Color
- 16bitová barva (65 536 barev) také označováno jako High Color
- 24bitová barva (16 777 216 barev) také označováno jako True Color
- 32bitová barva (4 294 967 296 barev) také označováno jako Super True Color (někdy také jako True Color)
- 48bitová barva (281 474 976 710 656 = 281,5 biliónů barev) také označováno jako Deep Color

Lidské oko je velmi kvalitní orgán a údajně dokáže od sebe odlišit až čtyři miliardy různých odstínů. Na základě toho jsou už barvy True Color („pravé“ barvy) považovány za vhodné pro kvalitní tisk fotografií v barevných časopisech.

Zápisy barev

Barvy je možné v informatice zapisovat v několika podobách. Většina z těchto způsobů je použitelná i na webových stránkách:

- **jménem v angličtině** – např. red. (seznam tzv. pojmenovaných barev se stále rozšiřuje)
- **procentuálním RGB zápisem** – např. rgb(100%,0%,0%)
- **desetinným RGB zápisem** – např. rgb(255,0,0)
- **desetinným RGBA zápisem** – např. rgba(255,0,0,0.3), stejné, jako předešlé, ale doplněné o alfa kanál – průhlednost
- **šestnáctkovým (hexadecimálním) RGB zápisem** – např. #FF0000 (tento způsob je nejpoužívanější)
- **zkráceným šestnáctkovým (hexadecimálním) RGB zápisem** – např. #F00 (pouze v případě, že se všechny dvojice cifer shodují)
- **HSL zápis** – např. hsl(0,100%,50%)
- **HSLA zápis** – např. hsla(0,100%,50%,0.8) – stejné jako předešlé, ale doplněné o alfa kanál, tedy průhlednost
- **šestnáctkovým (hexadecimálním) RGBA zápisem** – např. #FF00007A (doplněn ještě alfa kanál – průhlednost) – *nové*

Barevná věrnost a kalibrace

Již víme, že monitory vytvářejí barvy v režimu RGB a tiskárny v režimu CMYK, navíc všechna zařízení jsou nedokonalá. Výsledkem je to, že obrázek na monitoru se liší od snímané předlohy a výtisk na tiskárně bude mít opět trochu jiné barvy.

Zajistit co nejlepší barevnou věrnost můžeme např. kalibrací monitoru a

tiskárny. K tomuto účelu slouží např. výtisk kalibračního obrázku, který máme v „papírové“ podobě. Dále si stáhneme soubor se stejným obrázkem. Potom se pomocí ovládacích prvků monitoru pokusíme zajistit co nejlepší barevnou shodu. Profesionálové používají přesnou kalibraci pomocí měřicích sond a barevné profily jednotlivých zařízení (tzv. ICC profily).

Základní pojmy (témata) grafiky

Vhodné i pro maturanty k maturitnímu opakování

1. Vektorová grafika
    1. Základy vektorové grafiky
    2. Vektorové grafické formáty.
    3. Instalace písma, software pro vektorovou grafiku
    4. Rozlišení vektorových obrázků.
    5. Vstupní a výstupní hardwarová zařízení vhodná pro vektorovou grafiku
    6. Text ve vektorovém editoru – řetězcový, odstavcový
2. Rastrová grafika
    1. Základy rastrové grafiky
    2. Rastrové grafické formáty
    3. Instalace písma, software pro rastrovou grafiku
    4. Rozlišení rastrových obrázků
    5. Vstupní a výstupní hardwarová zařízení vhodná pro rastrovou grafiku
3. Barevné modely
    1. Barevný model RGB
    2. Barevný model CMYK
    3. Barevná hloubka, velikost souborů a jednotlivých typů
    4. Barevná schémata (škály)
    5. Barevná věrnost, kalibrace
4. Práce s barvami
    1. Psychologické působení barev, příklady
    2. Zásady používání barev
    3. Míchání barev, barvy doplňkové
5. Ofsetový tisk
    1. Princip ofsetové tisku
    2. Struktura nákladů offsetového a malonákladového tisku
6. Typografie
    1. Pravidla pro odstavce, znaky, spojení znaků atd.
    2. Používání písem a kombinovaných písem, vyznačování textu
    3. Umístění titulků a objektů na stránce
7. PDF
    1. Principy postscriptových souborů, typy
    2. Vytváření a čtení PDF
8. Text
    1. Velikost, řez a další vlastnosti písma

    2. Sady (druhy) písma, fonty, instalace
9. Kvalita, gramáž, velikost a barva papíru

Zajímavé odkazy pro grafiku

- Grafické prvky (ikony, vektory, PSD soubory, loga…) ke stažení – https://www.freepik.com/

On-line grafické nástroje pro bitmapovou grafiku

- **Photopea** – https://www.photopea.com/
- **Polotno Studio** – https://studio.polotno.com/
- **PIXLR editor** – https://pixlr.com/editor/
- **Sumo Paint** – https://www.sumopaint.com/app/
- **iPiccy editor** – https://ipiccy.com/
- **Polarr editor** – https://photoeditor.polarr.co/
- **Fottor editor** – https://www.fotor.com/
- **piZap editor** – https://www.pizap.com
- **FotoFlexer** – http://fotoflexer.com
- **Photolab** – https://photolab.me/
- **Ribbet** – https://www.ribbet.com/app
- **Photovisi** – https://www.photovisi.com/
- **Toolpic** – https://www.toolpic.com/
- **Photoshop Express** – http://www.photoshop.com/tools?wf=editor

---

On-line grafické nástroje pro vektorovou grafiku:

- **Boxy SVG –** https://boxy-svg.com/app
- **SVG-Edit** – http://www.clker.com/inc/svgedit/svg-editor.html
- **Inkscape** – https://www.rollapp.com/
- **Vectr** – https://vectr.com
- **PrimalDraw** – https://www.primaldraw.com
- **Gravit** – https://gravit.io/

---

23. Datum a čas v PHP česky

Pokud potřebujeme, aby se datumové údaje vypisovaly česky, nestačí nám pouze vestavěná funkce PHP Date.

Lze to vyřešit např. pomocí polí:

```
<html>
<head>
<meta charset="utf-8">
<title>Datum a čas v PHP česky</title>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<center><font face="Arial CE, Arial" size="5">
<?php
$mesice = array ("ledna", "února", "března", "dubna", "května", "června",
"července", "srpna", "září", "října", "listopadu", "prosince");

$den = array("neděle", "pondělí", "úterý", "středa", "čtvrtek", "pátek",
"sobota");

echo "<h1>Datum a čas v PHP česky: </h1><p>Je " . $den[Date ("w")]. ", " .
Date ("d") . ". ". $mesice[Date ("n") - 1] . " " . Date ("Y") . ", " . Date
("H:i:s") . "</p>";
?>
</font></center>
</body>
</html>
```

---

19. Články v databázi přes PHP

Nejdříve si v databázi vytvoříme tabulku ***Clanky***, do které budeme ukládat všechny články.

Jednotlivé sloupce tabulky budou tyto:

- ID (zároveň primární klíč),
- název článku,
- autor,
- obsah článku
- a datum.

Tuto tabulku můžete buď vytvořit ručně přes nějaké rozhraní typu Adminer nebo PHPMyAdmin a nebo jej můžete spustit přes soubor, který vidíte níže. Název může být např. ***vytvor-clanky.php***. Soubor obsahuje příkazy SQL a je napojen na ***db.php***, který zabezpečuje připojení k databázi. Ten samozřejmě musíte vytvořit nejdříve (viz předcházející kapitola).

Soubor ***vytvor-clanky.php*** může vypadat např. takto:

```
<?php
// Napojení souboru se spojením k databázi
```

```
require 'db.php';
// SQL příkaz k vytvoření tabulky Clanky v databázi
$vytvor_clanky = "CREATE TABLE Clanky (
id INT(6) AUTO_INCREMENT PRIMARY KEY,
nazev VARCHAR(30) NOT NULL,
autor VARCHAR(30) NOT NULL,
obsah TEXT,
datum VARCHAR(30) NOT NULL
)";
if (mysqli_query($spojeni, $vytvor_clanky)) {
    echo "Tabulka Clanky byla vytvořena";
} else {
    echo "Chyba, tabulka nebyla vytvořena: " . mysqli_error($spojeni);
}
mysqli_close($spojeni);
?>
```

Samotný příkaz SQL vypadá takto:

```
CREATE TABLE Clanky (
id INT(6) AUTO_INCREMENT PRIMARY KEY,
nazev VARCHAR(30) NOT NULL,
autor VARCHAR(30) NOT NULL,
obsah TEXT,
datum VARCHAR(30) NOT NULL
)
```

Dále si vytvoříme soubor, kterým budeme články vytvářet – ***novy-clanek.php:***

```
<?php
require 'db.php' ;
?>
<form action="" method="post">
Název článku:<br>
<input type="text" name="nazev"><br>
Autor:<br>
<input type="text" name="autor"><br>
Obsah článku:<br>
<textarea name="obsah"></textarea><br>
<input type="submit" value="Uložit článek">
</form>
<?php
if (!empty ($_POST))
// empty - prázdný a jeho negace nebo-li
// pokud bylo cokoliv zadáno do formuláře, začne se vykonávat následující
podmínka
{
// přijme všechny zadané informace do formuláře
$nazev = $_REQUEST["nazev"];
$autor = $_REQUEST["autor"];
$obsah = $_REQUEST["obsah"];
$datum = StrFTime("%d.%m.%Y", Time());
```

```
// uložíme data do databáze
$novy_clanek = mysqli_query($spojeni, "INSERT INTO Clanky (nazev, autor,
obsah, datum)
VALUES ('$nazev', '$autor', '$obsah','$datum')");
// Po uložení vypíše hlášku
echo ("Článek byl uložen<br>");
}
echo ('<a href="vypis.php">Výpis článků</a>');
mysqli_close($spojeni);
?>
```

V závěru předchozího kódu „voláme“ soubor ***vypis.php***, kterým si necháme vypsat všechny články v databázi:

```
<?php
require 'db.php';
$vysledek = mysqli_query($spojeni, "SELECT * FROM Clanky ORDER BY datum
DESC");
// Načtení článků seřazených podle datumu
while($data = mysqli_fetch_assoc($vysledek))
{
echo "<i>Název: </i><b>";
echo $data['nazev'];
echo "</b><br><i>Autor: </i>";
echo $data['autor'];
echo "<br><i>Vytvořeno: </i>";
echo $data['datum'];
echo "<br><i>Obsah článku:</i><br>";
echo $data['obsah'];
echo "<hr>";
}
?>
```

Na konec ještě vytvoříme základní rozhraní, např. takto:

```
<h1>Jednoduchý systém na ukládání článků do databáze</h1>
<p><a href="novy-clanek.php">Vytvořit nový článek</a></p>
<p><a href="vypis.php">Výpis článků</a></p>
```

Prohlédněte si demo

Takovýto systém ovšem neumožňuje editovat uložené články.

Úkoly ke kapitole:

- vytvořte ve formuláři pro odesílání článků ochranu před roboty
- doplňte stránky o CSS např. formou frameworku

---

18. Jednoduchá návštěvní kniha v PHP

Vytvoření návštěvní knihy v PHP s návazností na MySQL databází vyžaduje v první řadě připojení na server s touto databází. Než toto připojení vytvoříte, je třeba v dané databázi také tabulku pro návštěvní knihu vytvořit. Např. přes rozhraní Adminer nebo PhpMyAdmin takto můžeme potřebnou tabulku vytvořit tímto příkazem:

```
CREATE TABLE kniha
(
ID INT NOT NULL AUTO_INCREMENT ,
NICK VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL ,
EMAIL VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NULL ,
WEB VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NULL ,
VZKAZ VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL ,
DATUM VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL ,
PRIMARY KEY (ID)
);
```

Dále již můžeme zrealizovat připojení, které bývá umístěno většinou v samostatném souboru. Např. můžeme vytvořit soubor **db.php**, který obsahuje následující kód:

POZOR! – údaje pro připojení je třeba změnit podle vašeho připojení na databázový server.

```
<?php
//Zadání jednotlivých údajů potřebných pro připojení do databáze:
$dbhost = 'server.xy'; //server, kde se nachází databáze (často bývá
localhost)
$dbname = 'databaze'; //jméno databáze
$dbuser = 'uzivatel'; //uživatelské jméno
$dbpass = 'heslo'; //heslo
//Vytvoření připojení:
$spojeni = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname);
//Kontrola připojení s případnou hláškou
if (!$spojeni)
 {
 echo 'Spojení s mysql serverem se nepodařilo navázat.<br>';
 }
//Stav, kdy se podaří navázat spojení s databází již nevypisujeme
//Následuje nastavení kódování a následná práce s daty
 mysqli_set_charset($spojeni, "utf8");
?>
```

Ujistěte se, že soubor, který vytváříte má kódování UTF-8.

Tabulku v databázi můžeme vytvořit také tak, že se SQL příkazy vloží do PHP kódu. Tato varianta však vyžaduje nejprve vytvořit spojení s databází, tedy nejprve vytvořit soubor **db.php**. Následný soubor (můžeme ho nazvat např.

vytvor-knihu.php) může vypadat např. takto:

```
<?php
// Napojení souboru se spojením k databázi
require 'db.php';
// SQL příkaz k vytvoření tabulky Clanky v databázi
$vytvor_knihu = "
CREATE TABLE kniha
(
ID INT NOT NULL AUTO_INCREMENT ,
NICK VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL ,
EMAIL VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NULL ,
WEB VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NULL ,
VZKAZ VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL ,
DATUM VARCHAR(255) CHARACTER SET utf8 COLLATE utf8_czech_ci NOT NULL ,
PRIMARY KEY (ID)
)";
if (mysqli_query($spojeni, $vytvor_knihu)) {
    echo "Tabulka Clanky byla vytvořena";
} else {
    echo "Chyba, tabulka nebyla vytvořena: " . mysqli_error($spojeni);
}
mysqli_close($spojeni);
?>
```

V dalším kroku je vytvořen soubor, který umožňuje vepsat a následně poslat data do návštěvní knihy. Můžeme jej např. nazvat **kniha.php**

```
<?php
 header("Content-Type: text/html; charset=utf-8");
//řádek kvůli správnému kódování
 include('db.php');
//vložení externího souboru s připojením
?>

<form action="" method="post">
  <div class="form-group">
    <label for="nick">Nick:</label>
    <input type="text" class="form-control" name="nick">
  </div>
  <div class="form-group">
    <label for="email">Email:</label>
    <input type="text" class="form-control" name="email">
  </div>
  <div class="form-group">
    <label for="web">Váš web:</label>
    <input type="text" class="form-control" name="web">
  </div>
  <div class="form-group">
    <label for="comment">Vzkaz:</label>
    <textarea id="summernote" class="form-control" rows="5"
```

```
name="vzkaz"></textarea>
  </div>
  <div class="form-group">
    <label for="kontrola">Kontrolní otázka - Kolik je jedna a jedna
(slovem):</label>
    <input type="text" class="form-control" name="kontrola">
  </div>
  <button type="submit" class="btn btn-success">Odeslat vzkaz</button>

</form>
<!-- všimněte si, že formulář není součástí php kódu -->
<?php
if (!empty ($_POST))
// empty - prázdný a jeho negace nebo-li
// pokud bylo cokoliv zadáno do formuláře, začne se vykonávat následující
podmínka
{
 // přijme všechny zadané informace do formuláře
 $nick = $_REQUEST["nick"];
 $email = $_REQUEST["email"];
 $web = $_REQUEST["web"];
 $vzkaz = $_REQUEST["vzkaz"];
 $datum = StrFTime("%d.%m.%Y", Time());
 // Kontrolní otázka kvůli SPAMu
 $kontrola = $_REQUEST["kontrola"];
 // pokud byly vyplněn alespoň nick a vzkaz a pokud je správně zodpovězena
kontrolní otázka, může se nový příspěvek uložit do databáze
 if(($nick!="") and ($vzkaz!="") and ($kontrola=="dva"))
 {
 // uloží se nový vzkaz do databáze a vrátí 1
 $zapis = mysqli_query($spojeni, "INSERT INTO kniha (NICK, EMAIL, WEB, VZKAZ,
DATUM)
 VALUES ('$nick','$email','$web','$vzkaz','$datum');");
 // Hláška o zdárném zápisu
 echo ("<h1>Vzkaz přidán</h1>");
 }

// pokud nebyl doplněn nick a vzkaz nebo je chybná kontrolní otázka, zahlasí
výzvu
else {
  echo ('<div class="alert alert-danger">
  <strong>Chyba!</strong> Musíte vyplnit alespoň Nick, Vzkaz a Kontrolní
otázku.</div>');
}
}
 // vloží externí soubor s výpisem všech vzkazů, které jsou uloženy v
databázi
 include('vzkazy.php');
?>
```

Poslední částí je externí soubor, který zabezpečuje vypsání dat z databáze s

názvem **vzkazy.php**:

```
<?php

$pocet_vzkazu = mysqli_query($spojeni, "SELECT COUNT(*) as pocet FROM
kniha;");
//přečte počet vzkazů v databázi
$cislo = mysqli_fetch_array($pocet_vzkazu);
//sql příkaz na počet řádků
$celkem_vzkazu = $cislo["pocet"];

 echo "<b>Celkový počet vzkazů: ".$celkem_vzkazu."</b><hr>";
$vzkazy = mysqli_query($spojeni, "SELECT * FROM kniha;");

 // pokud nebyly načteny žádné vzkazy, vypiš chybu
 if(!$vzkazy){
 echo "Chyba při načtení vzkazů !!";
 }
 // jinak ulož vzkazy do pole a postupně je vypiš
 else{
 // cyklus while projde všechny vzkazy,
 // pomoci pole $vzkaz[""] vypíše všechny informace jednotlivých vzkazů
 while ($vzkaz = mysqli_fetch_array($vzkazy))
 {
 echo $vzkaz["ID"]."<br />";
 echo $vzkaz["NICK"]."<br />";
 echo $vzkaz["EMAIL"]."<br />";
 echo $vzkaz["WEB"]."<br />";
 echo $vzkaz["VZKAZ"]."<br />";
 echo $vzkaz["DATUM"]."<br />";
 }
 }
?>
```

Funkční návštěvní knihu (bez jakéhokoliv CSS) najdete v tomto demu.

Úkoly ke kapitole:

1. Pokuste se vložit tuto návštěvní knihu do webové stránky s bootstrapem a doplňte jak formulář, tak výpisy vzkazů kaskádovými styly (můžete se inspirovat např. zde)
2. Dále zkuste změnit tvar (podobu) datumu zápisu na den + přesný čas.
3. Přejmenujte patřičný soubor a případné odkazy na něj tak, aby návštěvní kniha „startovala“ z určitého adresáře, ve kterém jsou soubory návštěvní knihy uloženy
4. Vytvořte ochranu proti robotům např. ve formě kontrolní otázky (tedy za pomoci podmínky…)
5. Do formulářového prvku Vzkaz importujte jednoduchý editor (např. https://summernote.org/).
6. Další větší úpravou je vytvoření stránkování (např. jako v tomto demu). Zdrojové kódy k tomuto demu – navstevni-kniha-bs.zip

Další položky, které lze ukládat jako součást zápisu do návštěvní knihy

IP adresa

Jako další informaci, kterou můžeme ukládat do databáze je např. IP adresa uživatele. To můžeme v PHP detekovat velmi jednoduše:

```
echo 'Vaše IP adresa je ' . $_SERVER['REMOTE_ADDR'];
```

Toto zjištění IP adresy **$_SERVER[‚REMOTE_ADDR‘]** je možné pouze v případě, že bylo PHP voláno z prohlížeče. V CLI režimu (například spuštění z Terminálu cronem) není IP adresa k dispozici. Do databáze lze pak tuto adresu uložit např. jako hodnotu **varchar(39)**, kam se vejde jak IP verze 4. tak IP verze 6.

Zjišťování, ukládání i následná práce s IP adresami je však značně obšírnější a přináší bohužel další problémy. Více např. v tomto článku – php.baraja.cz/ip-adresa.

---

17. BMI – příklad k formulářům v PHP

Tento příklad je pomůcka pro zjištění správné váhy. Je zde využit tzv. Body Mass Index (BMI), který se vypočítá takto:

BMI = váha / (výška x výška)

POZN.: váha musí být v kg a výška v metrech!

Tabulka figury člověka podle BMI:

Podvyživený	**BMI je menší než 15**
Podváha	BMI je menší než 18,5
Ideální	BMI je od 18,5 do 25
Nadváha	BMI je od 25 do 30
Obezita	BMI je od 30 do 40
Nadměrně obézní	BMI je větší než 40

První stránka pro zadání hodnot vypadá takto:

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="utf-8">
    <title>Zadejte Vaše parametry</title>
</head>
```

```
<body>
  <form action="vysledek.php" method="post">
  Zadej váhu: <input type="text" name="vaha"> kg<br />
  Zadej výšku: <input type="text" name="vyska"> cm<br />
  <input type="submit" value="Spočítej figuru">
  </form>
</body>
</html>
```

Hodnota výšky se zadává běžně v cm, což je následně ve vyhodnocovacím skriptu potřeba převést na metry.

Druhá stránka, která má spočítat BMI:

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="utf-8">
    <title>Informace o figuře</title>
</head>
<body>
 <?php
  $vaha = $_POST['vaha'];
  $vyska = $_POST['vyska'];
  $vyska_v_metrech = $vyska / 100.0;
  //tato proměnná se vytváří jen proto, aby se převedla výška na metry
  $BMI = $vaha / ($vyska_v_metrech * $vyska_v_metrech);
  echo 'Vaše BMI = ', $BMI;
  echo '<br /><br />';
if ($BMI < 15)
  echo 'Jste podvyživený/á';
else if ($BMI < 18.5)
  echo 'Máte podváhu';
else if ($BMI < 25)
  echo 'Máte ideální postavu';
else if ($BMI < 30)
  echo 'Máte nadváhu';
else if ($BMI < 40)
  echo 'Jste obézní';
else
  echo 'Jste nadměrně obézní';
?>
</body>
</html>
```

---

16. Formuláře v PHP

Formuláře jsou naprostým základem webových stránek, které chcete mít interaktivní, neboli chcete, aby bylo možné pracovat s odezvou uživatele.

Naprostým předpokladem je, že v tomto případě musíte mít alespoň základní znalosti HTML (XHTML) v oblasti tvorby formulářů a jejich prvků.

Formuláře a jejich parametry v HTML (XHTML)

Při vytváření jakéhokoliv formuláře HTML začínáme značkou <form>. Ta má několik dalších parametrů. Pro naše současné účely nám ovšem budou postačovat dva hlavní.

Prvním je parametr action

Ten určuje příjemce dat z formuláře. Pro nás to bude nějaký PHP skript, který bude schopen tyto údaje přijmout a zpracovat, popř. vypsat. O tomto parametru později.

Druhým veledůležitým parametrem je parametr method

Ten určuje, jakým způsobem se budou data z formuláře posílat. Zde se v praxi využívá prakticky pouze dvou metod a to je metoda post a get.

Abychom mohli data z formuláře odeslat, musíme k tomu vytvořit nějaký příkaz. Používá se odesílací tlačítko, které se tvoří vložením
značky <input> s parametrem type="submit". To zařídí, že se všechna data odešlou do námi zadaného skriptu.

Typické použití formulářů vypadá asi takto:

```
<form action="mujskript.php" method="post">
<!-- datové položky formuláře -->
<input type="submit" value="Odeslat data formuláře">
</form>
```

Dále si zkusíme do formuláře přidat textové pole, aby bylo co odesílat:

```
<form action="mujskript.php" method="post">
Jméno: <input type="text" name="jmeno"><br />
<input type="submit" value="Odeslat data formuláře" />
</form>
```

Nyní si vytvoříme kompletní jednoduchou webovou stránku s formulářem:

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <meta charset="utf-8">
    <title>Jednoduchý formulář</title>
```

```
</head>
<body>
      <form action="vypis_dat.php" method="post">
      Zadej svoje jméno: <input type="text" name="jmeno"><br />
      <input type="submit" value="Odeslat data formuláře" />
      </form>
</body>
</html>
```

Jak můžete z kódu vypozorovat, data mají být zpracovány souborem **vypis_dat.php**. Ten ovšem do této chvíle neexistuje. Proto si jej vytvoříme:

```
<!DOCTYPE html>
<html lang="cs">
<head>
    <meta charset="utf-8">
    <title>Zpracovaná data z formuláře</title>
</head>
<body>
      <?php
            $jmeno = $_POST['jmeno'];
            echo 'Vaše jméno je ', $jmeno;
      ?>
</body>
</html>
```

Aby se vám skutečně data z formuláře vypsaly, musí se tento soubor nacházet **ve stejné složce jako je ten, ze kterého data odesíláte** (pokud tedy nemáte zadanou jinou cestu k souboru, což v našem případě nemáme).

Ještě se zaměříme na předešlý kód. První řádek v PHP kódu je deklarace proměnné ***$jmeno***, ke které je přiřazen záznam z odeslaného formuláře a to konkrétně s názvem ***jmeno***. Jak vidíte je zde i zmíněna právě metoda zasílaných dat – **$_POST**. Pokud tedy odesíláte jednou metodou, nemůžete pro zpracování použít druhou metodu. **Dnes se prakticky v 99% pracuje s metodou POST**.

Rozdíly odesílacích metod:

Metoda GET	Metoda POST
Neumí odeslat rozsáhlá data	Umí odesílat rozsáhlá data
Odesílá data jako součást adresy URL	Odesílá data jinou cestou, než je adresa URL

Formuláře s podmínkou

První příklad je zaměřen na formulář, u kterého přijímací skript ověřuje podmínku – zadání správného hesla. Nejprve tedy stránka s formulářem, do kterého musí uživatel napsat správné heslo:

```
<!DOCTYPE html>
<html lang="cs">
```

```
  <head>
    <meta charset="utf-8">
    <title>Zadej prosím heslo</title>
</head>
<body>
 <h2>Pokud se chceš dozvědět podrobnosti o srazu, musíš zadat správné
heslo</h2>
  <form action="vysledek.php" method="post">
   Zadej heslo: <input type="password" name="heslo"><br>
   <input type="submit" value="Odeslat heslo">
  </form>
</body>
</html>
```

Po spuštění se vyhodnocuje odeslaný výraz z formuláře **v souboru vysledek.php** takto:

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <meta charset="utf-8">
    <title>Informace o srazu</title>
</head>
<body>
  <?php
    if ($_POST['heslo'] == 'lokomotiva')
     echo 'Sraz se bude konat zítra ve 3 hodiny u Tygra';
    else
     echo 'Bohužel zadal jsi špatné heslo, nic se nedozvíš';
?>
</body>
</html>
```

Skript je doplněn o podmínku if. Správné heslo je „lokomotiva“. Pokud jej tedy uživatel nezadá, nezobrazí se mu požadovaný text.

Pokuste se doplnit podmínku o další prvek, který je nutno správně do formuláře zadat.

Doplňte stránku frameworkem Bootstrap.

---

15. Komunikace PHP s databází

Vlastní komunikace z hlediska skriptu PHP vypadá

následovně:

1. **připojení** k databázovému serveru MySQL a výběr databáze
2. **zaslání příkazů a převzetí výsledných dat** (či aktualizace těchto dat) z databázového serveru MySQL
3. **odpojení** se od databázového serveru MySQL

Než tyto tři kroky provedeme, měli bychom znát několik základních funkcí PHP, kterými „ovládáme“ databázi. Tyto funkce používáme za pomoci ovladačů.
Nejčastější typy ovladačů:

- **mysql** (pouze pro verze PHP 5.5 a nižší, zastaralé a již se nedoporučuje používat)
- **mysqli** (MySQL Improved, parametry se do dotazů předávají velmi nepohodlně)
- **PDO** (PHP Database Objects, jedná se o nejnovější a velmi kvalitní objektový ovladač, který se jednoduše používá a podporuje kromě MySQL ještě několik databází.
- a další

Funkcím, které začínají výrazem ***mysql***, je již v PHP lepší se **úplně vyhnout**. Pro PHP do verze 5.4 se mohlo používat funkce **MySQL. Od vyšších verzí PHP je nutné používat funkci MySQLi nebo PDO_MySQL.**

Více informací najdete např. v tomto článku (odkaz na web http://jecas.cz)

Několik základních příkazů pro práci s databází:

- **mysqli_connect** – otevírá spojení se serverem MySQL
- **mysqli_select_db** – vybíra MySQL databázi
- **mysqli_query** – posíla MySQL příkaz
- **mysqli_num_rows** – vrací počet řádků výsledku příkazu
- **mysqli_affected_rows** – vrací počet řádků, které byly ovlivněny poslední operaci
- **mysqli_fetch_array** – zpracovává jeden řádek výsledku příkazu, vrací asociativní pole, číselné pole či oboje
- **mysqli_fetch_row** – zpracovává jeden řádek výsledku příkazu, vrací číselné pole
- **mysqli_fetch_assoc** – zpracovává jeden řádek výsledku příkazu, vrací asociativní pole
- **mysqli_unbuffered_query** – posílá MySQL příkaz bez automatického zpracování výsledku a bez ukládání do vyrovnávací paměti
- **mysqli_error** – vrací chybný řádek posledního MySQL příkazu
- **mysqli_change_user** – mění uživatele pro určité připojení
- **mysqli_free_result** – uvolňuje paměť od výsledku příkazu
- **mysqli_result** – vrací data výsledku příkazu
- **mysqli_fetch_object** – zpracovává jeden řádek výsledku příkazu a vrací objekt
- **mysqli_client_encoding** – vrací kódování spojení
- **mysqli_close** – ukončuje spojení s MySQL serverem
- **mysqli_pconnect** – vytvoří trvalé připojení se serverem MySQL
- **mysqli_create_db** – vytvoří MySQL databázi

- **mysqli_db_name** – vrací název databáze
- **mysqli_db_query** – přepíná se na uvedenou databázi a posílá příkaz
- **mysqli_field_len** – vrací délku pole
- **mysqli_get_host_info** – vrací informaci o MySQL připojení
- **mysqli_get_client_info** – vrací informaci o MySQL uživateli
- **mysqli_field_flags** – vrací atributy uvedeného pole
- **mysqli_drop_db** – smaže MySQL databázi

Připojení k databázovému serveru MySQL a výběr databáze

Pro připojení k databázovému serveru MySQL slouží funkce **mysqli_connect.** Často se doplňuje o kontrolu, zda se připojení podařilo. Dále se provádí samotné dotazy do databáze a nasleduje odpojení, i když to PHP provede většinou samo i bez našeho přičinění. Patří to ale ke slušnému stylu:

```
<?php
//Zadání jednotlivých údajů potřebných pro připojení do databáze:
$dbhost = 'server.xy'; //server, kde se nachází databáze (často bývá
localhost)
$dbname = 'databaze'; //jméno databáze
$dbuser = 'uzivatel'; //uživatelské jméno
$dbpass = 'heslo'; //heslo
//Vytvoření připojení:
$spojeni = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname);
//Kontrola připojení s případnou hláškou
if (!$spojeni)
 {
 echo 'Spojení s mysql serverem se nepodařilo navázat.<br>';
 }
 else
 {
 echo 'Spojení s mysql serverem bylo úspěšně navázáno.<br>';
//Následuje nastavení kódování a následná práce s daty
 mysqli_set_charset($spojeni, "utf8");

//Odpojení od databáze s kontrolou
//Odpojení je možné až po skončení všech skriptů, které čerpají data z
databáze
 $zavreni = mysqli_close($spojeni);
 if(!$zavreni)
 {
 echo 'Spojení s mysql serverem se nepodařilo ukončit.';
 }
 else
 {
 echo 'Spojení s mysql serverem se podařilo ukončit.';
 }
}
?>
```

Tento zdrojový kód můžete mít přímo v souboru s jakýmkoliv dalším skriptem nebo bývá často jako samostatný soubor, např. db.php. **Pokud se bude jednat o samostatný soubor, musíte vynechat „Odpojení od databáze“, protože jinak nebude možné s databází cokoliv dělat.** Soubor db.php tedy může vypadat např. takto:

```
<?php
//Zadání jednotlivých údajů potřebných pro připojení do databáze:
$dbhost = 'server.xy'; //server, kde se nachází databáze (často bývá
localhost)
$dbname = 'databaze'; //jméno databáze
$dbuser = 'uzivatel'; //uživatelské jméno
$dbpass = 'heslo'; //heslo
//Vytvoření připojení:
$spojeni = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname);
//Kontrola připojení s případnou hláškou
if (!$spojeni)
 {
 echo 'Spojení s mysql serverem se nepodařilo navázat.<br>';
 }
//Stav, kdy se podaří navázat spojení s databází již nevypisujeme
//Následuje nastavení kódování a následná práce s daty
 mysqli_set_charset($spojeni, "utf8");
?>
```

Tento soubor (obvykle s názvem db.php, config.php, admin-config.php configuration.php apod….) pak bývá součástí dalších souborů, kteří jej mají vložen např. přes příkaz include nebo require. Ukončení spojení s databází pak bývá právě tam.

Zaslání příkazů a převzetí výsledných dat

Výpis dat z databáze se prování nejčastěji pomocí příkazu jazyka SQL, což je příkaz **SELECT** Jeho základní struktura je tato:

- **SELECT** – seznam požadovaných položek
  - FROM seznam tabulek
  - WHERE – podmínka
  - GROUP BY – seznam položek
  - HAVING – skupinová podmínka
  - ORDER BY – třídění;

Následuje výběr tabulky z databáze. V našem příkladu to bude tabulka „knihy“. **V minulé kapitole byla vytvořena a následně smazána, proto si ji musíte vytvořit znovu:**

```
create table knihy
(
CisloKnihy integer not null primary key,
NazevKnihy varchar(60) not null,
Autor varchar(60) not null
)
```

Dále ještě znovu vložíme 3 knihy:

```
insert into knihy values (1, 'Malý Bobeš','J.V.Pleva');
insert into knihy values (2, 'R.U.R.','Karel Čapek');
insert into knihy values (3, 'Bylo nás pět','Karel Poláček');
```

Výběr tabulky provedeme funkcí mysqli_query např. takto:

```
$vyber_tabulky = mysqli_query("SELECT * FROM knihy");
```

Abychom výsledná data skutečně viděli, je třeba, abychom byli připojeni na databázi a aby data ve volané tabulce existovaly.

Dále si necháme vypsat data z této tabulky a to funkcí **mysqli_fetch_array**. Cyklus WHILE obsahuje nově vzniklou proměnnou $vysledek, která pak postupně načítá data do polí s určitým názvem. Název pole odpovídá názvům sloupců z tabulky v databázi.

```
<?php
// Krok 1: připojení do databáze externím souborem
include 'db.php';
// Krok 2: Napojení na konkrétní tabulku včetně kontroly
$vyber_tabulky = mysqli_query($spojeni, "SELECT * FROM knihy");
     if(!$vyber_tabulky)
      {
        echo 'Dotaz vykazuje chybu.<br>';
      }
      else
      {
// Krok 3: Pokud je vše v pořádku, zpracuje se dotaz a vypíše
// Je použit cyklus while, který vypisuje data tolikrát, kolik jich v tabulce
je
       while($radek = mysqli_fetch_array($vyber_tabulky))
        {
          echo
          "Číslo knihy: ".$radek["CisloKnihy"]."<br>".
          "Název knihy: ".$radek["NazevKnihy"]."<br>".
          "Autor: ".$radek["Autor"]."<br><hr>";
        }
      }
// Krok 4: Odpojení od databáze
$zavreni = mysqli_close($spojeni);
    if(!$zavreni)
    {
      echo 'Spojení s mysql serverem se nepodařilo ukončit.';
    }
?>
```

Informaci o zdárném ukončení spojení s databází již nevypisuje. Návštěvníky webu by tak zbytečně obtěžovala.

Pokud chceme seřadit vypsané záznamy podle nějakého kritéria (v tomto případě

podle čísel knih od největšího po nejmenší), přidáme k příkazu SELECT následující:

```
$dotaz = mysqli_query("SELECT * FROM knihy ORDER BY CisloKnihy DESC");
```

Dále můžeme vypsat např. počet záznamů v tabulce funkcí **mysql_num_rows:**

```
$pocet = mysqli_num_rows($dotaz);
echo '<br>Celkový počet záznamů v tabulce je: '.$pocet;
```

Kompletní zdrojový kód:

```
<?php
// Krok 1: připojení do databáze externím souborem
include 'db.php';
// Krok 2: Napojení na konkrétní tabulku včetně kontroly
$vyber_tabulky = mysqli_query($spojeni, "SELECT * FROM knihy ORDER BY
CisloKnihy DESC");
 if(!$vyber_tabulky)
 {
  echo 'Dotaz vykazuje chybu.<br>';
 }
 else
 {
  echo 'Dotaz byl úspěšně vykonán.<br>';
// Krok 3: Pokud je vše v pořádku, zpracuje se dotaz a vypíše
// Je použit cyklus while, který vypisuje data tolikrát, kolik jich v tabulce
je
while($radek = mysqli_fetch_array($vyber_tabulky))
 {
  echo
   "Číslo knihy: ".$radek["CisloKnihy"]."<br>".
   "Název knihy: ".$radek["NazevKnihy"]."<br>".
   "Autor: ".$radek["Autor"]."<br><hr>";
}
// Výpis celkového počtu knih
$pocet = mysqli_num_rows($vyber_tabulky);
 echo '<h2>Celkový počet záznamů v tabulce je: '.$pocet."</h2>";
 }
// Krok 4: Odpojení od databáze
 $zavreni = mysqli_close($spojeni);
 if(!$zavreni)
 {
 echo 'Spojení s mysql serverem se nepodařilo ukončit.';
 }
?>
```

SQL příkazy za pomoci ovladače PDO

Protože ovladač **mysqli** přináší určitá bezpečnostní rizika (především SQL injection), je někdy vhodnější používat ovladač PDO. Zdrojový kód pro vložení dat (v tomto případě nové knihy č. 4) by pak vypadal např. takto:

```
<?php
$dbhost = "server.xy";
$dbname = "databaze";
$dbuser = "uzivatel";
$dbpass = "heslo";
try {
 $conn = new PDO("mysql:host=$dbhost;dbname=$dbname", $dbuser, $dbpass);
 // set the PDO error mode to exception
 $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
 $conn->exec("set names utf8");
 $sql = "INSERT INTO knihy (CisloKnihy, NazevKnihy, Autor)
 VALUES ('4', 'Létající Čestmír', 'Karel Jarolím')";
 // use exec() because no results are returned
 $conn->exec($sql);
 echo "Nová kniha byla vložena";
 }
catch(PDOException $e)
 {
 echo $sql . "<br>" . $e->getMessage();
 }$conn = null;
?>
```

Jedná se o objektově orientované programování, které není součástí tohoto základního materiálu.

Zdroj – http://jecas.cz/pdo

---

14. PHP a databáze

Pro spojení databáze a PHP je potřeba znát alespoň několik základních informací o databázích a především o jazyce MySQL.

Databáze MySQL

Dnešní PHP stránky už si bez spolupráce s nějakou databází prakticky nedovedeme představit. V největší míře se používá databáze **MySQL**. Další variantou může být také např. **PostgreSQL** a další. Pro základní operace a funkčnost se však zaměříme pouze na databázi MySQL. Jedná se o produkt, který je zdarma, i když ne úplně ve všech případech.

Nástroje pro práci s MySQL
Nejčastějším nástrojem je phpMyAdmin. Ten umožňuje prakticky jakékoliv operace s databází i se všemi daty v ní. Bývá také součástí většiny předpřipravených balíčků webových serverů (např. PHP Web Server, PHP Easy, WAMP Server apod.).

Dalším velmi zdařilým a přitom jednoduchým nástrojem je Adminer českého autora Jakuba Vrány. Jeho výhodou je především velmi jednoduchá instalace. Obsahuje totiž **pouze jeden soubor**, který nakopírujeme na webový server a spustíme v prohlížeči. Srovnání Adminera a PHPMyAdmin najdete také na stránce autora – http://www.adminer.org/cs/phpmyadmin/.

Připojení k databázi
Pro připojení k databázi potřebujeme tyto základní údaje:

1. **název hostitele (adresa serveru)** – každá databáze běží na nějakém „stroji“ (virtuálním nebo běžném serveru). Název tohoto serveru může být buď název v podobě nějaké adresy typu **mysql.server.cz** (případně IP adresa tohoto serveru) a nebo v případě lokálního umístění databáze v podobě tvaru **localhost** (případně verze IP adresy – 127.0.0.1). Název hostitele obdržíme vždy od poskytovatele webhostingu
2. **název databáze** – název je opět buď vytvořen poskytovatelem webhostingu nebo si ho sami můžeme vytvořit v administraci našeho webhostingového účtu
3. **uživatelské jméno** – opět platí stejná pravidla, jako u názvu databáze.
4. **heslo** – i zde platí stejná pravidla

Někdy bývá (opravdu jen výjimečně) potřeba zadat ještě port. MySQL naschlouchá na portu 3306, pokud není při konfiguraci serveru určeno jinak.

Práce s databází
Prvním krokem je vytvoření databáze. Přes rozhraní phpMyAdminu nebo Admineru to můžeme udělat velmi jednoduše přes formulářové pole (tzv. „na kliknutí“). Velmi často také bývá minimálně jedna databáze vytvořena již se vznikem webohostingového prostoru. Každá databáze (pokud není prázdná) obsahuje tabulky. Současně s tvorbou tabulky také vytváříme sloupce tabulky, u kterých musíme definovat jejich obsah. Tedy jaká data v nich budou obsažena, jejich **datový typ**. Můžeme také definovat, zda je v daném sloupci povolena nulová hodnota nebo zda je daný sloupec definován jako primární klíč.

Datové typy MySQL

Základní informace

- pro omezení délky řetězce (maximální velikost je 255) používáme parametr „m“, zápis je: datovy_typ(m) – např.: TINYINT(1), nebo VARCHAR(100)
- u reálných čísel používáme navíc parametr „d“ (maximální velikost je 30), tímto parametrem omezíme délku čísla za desetinou čárkou, zápis je: datovy_typ(m,d) – např.: FLOAT(5,3)
- sloupce určené jako INDEXY (nebo i PRIMARY KEY) označíme na konci deklarace tabulky – např.: CREATE TABLE pokus (jm CHAR(20) NOT NULL, cis INT, PRIMARY KEY (jm), INDEX(cis));
- název indexu (INDEX nazev (sloupec)) zadáváme pokud bude indexů více

Datové typy se nejčastěji dělí na tři základní skupiny:

1. číselné
2. textové (někdy označované jako znakové nebo řetězcové)

3. datumové (datum a čas)

1. Číselné

Celočíselné typy

Typ	Popis	Atributy	Rozsah
TINYINT	velmi malá čísla	AUTO_INCREMENT, UNSIGNED, SIGNED, ZEROFILL	−128 až 127; UNSIGNED => 0 až 255
SMALLINT	malá čísla	AUTO_INCREMENT, UNSIGNED, SIGNED, ZEROFILL	−32768 až 32767; UNSIGNED => 0 až 65535
MEDIUMINT	střední celá čísla	AUTO_INCREMENT, UNSIGNED, SIGNED, ZEROFILL	−8388608 až 8388607; UNSIGNED => 0 až 16777215
INT nebo INTEGER	běžné celé číslo	AUTO_INCREMENT, UNSIGNED, SIGNED, ZEROFILL	−2147483648 až 2147483647;UNSIGNED => 0 až 4294967295
BIGINT	velká celá čísla	AUTO_INCREMENT, UNSIGNED, SIGNED, ZEROFILL	−9223372036854775808 až 9223372036854; UNSIGNED => 0 až 18446744073709551615

Reálné typy (s desetinnou čárkou)

Typ	Popis	Atributy	Rozsah
FLOAT	malá desetinná čísla	UNSIGNED, SIGNED a ZEROFILL	nejmenší nenulové hodnoty jsou ±1,175494351E−38; největší nenulové hodnoty jsou −±3,402823466E+38; UNSIGNED => záporné hodnoty jsou zakázány
DOUBLE	velká desetinná čísla	UNSIGNED, SIGNED a ZEROFILL	nejmenší nenulové hodnoty jsou ±2,2250738585072014E-308; největší nenulové hodnoty jsou −±1,17976931348623157E+308; UNSIGNED => záporné hodnoty zakázané
DECIMAL	velká desetinná čísla ukládaná jako řetězce	UNSIGNED, SIGNED a ZEROFILL	stejné jako DOUBLE

2. Textové typy

Používají se pro ukládání textu; některé typy dokáží rozlišovat i velká a malá písmena; lze do nich ukládat i čísla

Typ	Popis	Atributy
CHAR	řetězec pevně dané délky 0-255; delší text bude oříznut, kratší bude doplněn mezerami do zadané délky	BINARY, CHARACTER SET

VARCHAR	stejné jako CHAR, ale kratší text není doplněn mezerami	BINARY, CHARACTER SET
TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB	klasické hodnoty BLOB	–
TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT	klasický text	CHARACTER SET
ENUM	výčtový typ	–
SET	množinový typ	–

3. Typy pro datum a čas

Typ	Popis	Rozsah
DATE	datum ve formátu CCYY-MM-DD	1000-01-01´ až ,9999-12-31
TIME	čas ve formátu ‚hh:mm:ss'; pro záporné hodnoty ‚-hh:mm:ss'; reprezentuje uplynulý čas nebo čas dne	-838:59.59´ až ‚838:59:59
TIMESTAMP	časová značka, obsahuje datum a čas; pokud zadáme NULL vyplní se zde aktuální datum a čas	–
DATETIME	datum a čas ve formátu ‚CCYY-MM-DD hh:mm:ss'	1000-01-01 00:00:00´ až ‚999-12-31 23:59:59

Modifikátory

Modifikátory jsou nastavení tabulek, kterými určujeme chování jednotlivých záznamů.

1. **AUTO_INCREMENT** – systém si sám ve sloupci generuje unikátní (jedinečné) číselné hodnoty
   – modifikátor lze použít pouze na celočíselný datový typ
   – za deklarací nové tabulky můžeme ještě navíc určit výchozí hodnotu: …AUTO_INCREMENT=50;
2. **CHARACTER SET –** znaková sada sloupce
3. **BINARY** – pro CHAR a VARCHAR; tento typ bude brán jako binární a budou se tak rozlišovat malá a velká písmena
4. **DEFAULT** vychozi_hodnota – pokud bude buňka prázdná, systém do ní automaticky přiřadí hodnotu „vychozi_hodnota"
   – řetězce nezapomeňte psát v uvozovkách
5. **FULLTEXT INDEX** – platí pro sloupce typu CHAR, VARCHAR a TEXT
   – fultextový index slouží k rychlejšímu hledání dat v textových polích
   – hledání v takovýchto polích provádíme pomocí píkazů MATCH a AGAINST
   – př.: SELECT * FROM tabulka WHERE MATCH(sloupec) AGAINST(„hledana_hodnota");
6. **INDEX** – sloupec/sloupce označené jako INDEX umožní rychlejší přístup k datům která obsahují
7. **NOT NULL** – pokud použijeme tento modifikátor, označený typ bude muset v každé buňce obsahovat nějakou hodnotu
8. **NULL** – opak NOT NULL; buňka může být prázdná
9. **PRIMARY KEY** – označený typ bude sloužit jako primární klíč – při jeho

použití musíme zároveň použít UNIQUE – sloupec nám tedy jedinečným způsobem identifikuje záznamy v tabulce
10. **UNIQUE** – v daném sloupci nesmějí být v buňkách stejné hodnoty, tedy co kus to unikát
11. **UNSIGNED** – pokud použijeme modifikátor UNSIGNED, datový typ bude bez znaménka a posune se interval hodnot
    – u čísel s pohyblivou desetinou čárkou se interval použitím UNSIGNED neposunuje a berou se jen kladná čísla
    – př.: TINYINT má rozsah -118 až +127 a TINYINT UNSIGNED má rozsah 0 až 255
12. **ZEROFILL** – použití u čísel, dotaz doplní před číslo nuly v celé jeho šířce
    – př.: pokud máme definováno MEDIUMINT(6) ZEROFILL a je v něm hodnota 123, tak se nám zobrazí 000123

Základní příkazy SQL

SQL příkazy můžete používat jak v prostředí Adminera (nebo phpMyAdminu), ale i následně v rámci PHP kódu (to je častější použití v praxi). Nejprve si ukažme práci v okně SQL příkazů:

Create table – tvorba tabulek

Pomocí tohoto příkazu vytvoříte tabulku v databázi. Zjednodušená syntaxe vypadá takto:

```
create table název tabulky (název sloupce datový typ null/not null)
```

Příklad:

```
create table knihy
(
CisloKnihy integer not null primary key,
NazevKnihy varchar(60) not null,
Autor varchar(60) not null
)
```

Tabulka nese název ***knihy*** a obsahuje celkem 3 sloupce:

1. **číslo** knihy (CisloKnihy), datový typ integer, nesmí být nulový a je to primární klíč tabulky
2. **název** knihy (NazevKnihy), datový typ varchar o maximální délce 60 znaků a nesmí být nulový
3. **autora** knihy (Autor), datový typ varchar o maximální délce 60 znaků a nesmí být nulový

Insert – vložení dat

Příkazem insert se vkládají data do tabulky. Syntaxe:

```
insert into NazevTabulky values (hodnota,…)
```

Příklad:

```
insert into knihy values (1, 'Malý Bobeš','J.V.Pleva');
insert into knihy values (2, 'R.U.R.','Karel Čapek');
insert into knihy values (3, 'Bylo nás pět','Karel Poláček');
```

Select – načtení z tabulky

Příkaz select načte data z tabulky. Může být spojen i s nějakou podmínkou.

```
select NazevSloupce from NazevTabulky [where NazevSloupce=hodnota]
```

Příkaz v hranatých závorkách je volitelný

Příklad:

```
select CisloKnihy, NazevKnihy from knihy where Autor = 'Karel Čapek'
```

(vyber údaje ze sloupce CisloKnihy a NazevKnihy z tabulky Knihy, kde je uveden autor Karel Čapek)

Další příklady:

```
select * from knihy
```

– vypíše všechny položky záznamů

```
select NazevKnihy, Autor from knihy
```

– vypíše jen vybrané položky záznamů (název a autora)

```
select NazevKnihy, Autor from knihy order by NazevKnihy
```

– vypíše názvy a autory seřazené podle názvů knih

```
select NazevKnihy, Autor from knihy order by NazevKnihy desc
```

– vypíše názvy a autory seřazené podle názvů knih, ale v opačném abecedním pořadí (parametr „desc“)

```
select NazevKnihy, Autor from knihy where NazevKnihy like "P%" or autor like "M%"
```

– vypíše názvy a autory knih, jejichž názvy začínají písmenem P nebo pokud jméno autora začíná písmenem M

Delete – odstranění řádků

Příkazem delete se odstraní řádky tabulky. Syntaxe vypadá následovně:

```
delete from NazevTabulky [where NazevSloupce=hodnota]
```

Příklad:

delete from knihy where Autor = 'Karel Čapek'
(odstraní z tabulky knihy všechny knihy, které mají jako autora uvedeného Karla Čapka)

Update – změna hodnot

Příkazem update se změní hodnoty sloupce v tabulce. Zjednodušená syntaxe:

update NazevTabulky set NazevSloupce = NovaHodnota

Příklad:

update knihy set Autor = 'Božena Němcová' where CisloKnihy = 3

Tento příklad změní autora knihy s číslem 3 na novou hodnotu – Božena Němcová.

Drop Table – odstranění tabulky

Tímto příkazem odstraníte celou tabulku z databáze.

Syntaxe:

drop table NazevTabulky

Příklad:

drop table knihy

To je jen několik základních nejčastěji používaných příkazů. Existuje jich samozřejmě daleko více.

---

13. Vložené soubory v PHP

Pro vkládání souborů do kódu se používají funkce **include**, **require, include_once, require_once**.

Základní rozdíl mezi funkcí include a require je v tom, že pokud funkce **include volá soubor, který neexistuje, vypíše chybové hlášení**, ale pokračuje ve skriptu. Kdežto funkce **require skončí fatální chybou**.

Funkce include_once a require_once se používají v případě, **kdy potřebujeme zamezit vícenásobnému vložení stejného externího souboru**. Varianty s _once na konci zaručí, že se v případě vícenásobného použití stejného souboru vloží jen ten první.
Nejčastěji se ovšem používá funkce include.

Příkaz include

Příkaz include je základní příkaz, kterým je možné do skriptu PHP vložit jiný skript PHP. Typické je to např. v případě vkládání menu webových stránek, kdy je takovéto menu uloženo v jednom souboru a ostatní si jej pouze vkládají v podobě funkce include. Při změně menu se tak nemusejí pracně jednotlivé stránky měnit a automaticky načítají novou podobu takového menu.

Pro začátek vložíme do stránky jinou, která obsahuje adresu. Nejprve vytvoříme soubor s adresou:

```
<em>Kontakt:</em><br />
<strong>Jaroslav Novák</strong><br />
Holická 33<br />
111 11 Počernice<br />
e-mail: jaroslav.novak[zavináč]firmaxy.cz<br />
tel.: 123 456 789<br />
```

Tento soubor načítá adresu funkcí include:

```
<html>
<head>
<title>Příklad na include</title>
<meta charset='utf-8'>
</head>
<body>
<p>Tady je vlastní text stránky.</p>
<?php
 include 'adresa.php';
?>
</body>
</html>
```

Pomocí příkazu include je možné vložit cokoliv, teda i kód PHP. Např.:

```
<?php
 echo 'Tady je kousek PHP kódu<br />';
?>
```

Počet vložení externího souboru do kódu pomocí funkce include není omezen.

Když vkládaný soubor chybí

Pokud vložený soubor neexistuje, vypíše PHP chybové hlášení (ovšem záleží na nastavení serveru). Např:
Warning: include(test-include.php) [function.include]: failed to open stream: No such file or directory in C:\wamp\www\adresa.php on line 9

V praxi se samozřejmě příkazy pro vložení externího souboru využívají v daleko větší míře a především s následným zapojením databáze.

Další jednoduchý příklad ukazuje, jak lze „poskládat“ webovou stránku tak,

abychom si při vytváření dalších nových stránek webu ulehčili práci.

Nejprve je dobré si vytvořit jednotlivé části webu, které se nebudou v celém systému webových stránek měnit. Jedná se např. o hlavičku dokumentu, menu, zápatí apod. Tak že např. zahlavi.php, menu.php a paticka.php. Následné složení stránky může vypadat např. takto:

```
<?php
include ("hlavicka.php");
include ("menu.php");
?>
Zde bude unikátní část webu, na každé stránce jiný...
<?php
include ("paticka.php");
?>
```

Každá část webu pak musí vypadat takto, pouze unikátní část html kódu se bude lišit.

Dalším vylepšením bude to, že si každou tuto „unikátní část webu“ budeme chtít načítat např. z databáze. Pak nám bude stačit pouze jeden soubor (např. index.php). Další stránky nebudeme potřebovat, protože hlavní („unikátní“) část webu se bude dynamicky načítat podle toho, jakou budeme „volat“ stránku. K tomuto účelu se využívá metoda _GET, která dokáže přidat do adresy URL jakoukoliv hodnotu (parametr) a tím „zavolat“ patřičný obsah.

Nejprve začneme jednoduchým vložením externího souboru za pomoci funkce require. Napojení do databáze si vyzkoušíme až po prostudování kapitoly o databázích.

```
<html>
<head>
<title>Příklad na vkládání externích souborů</title>
<meta charset='utf-8'>
</head>
<body>
<h1>Menu:</h1>
<ul>
  <li><a href="index.php?stranka=uvod">Úvod</a></li>
  <li><a href="index.php?stranka=reference">Reference</a></li>
  <li><a href="index.php?stranka=kontakt">Kontakt</a></li>
</ul>
<?php
echo "<p>Dynamický obsah:</p>";
require($_GET['stranka'] . '.php');
?>
</body>
</html>
```

Jak pracuje metoda _GET s parametrem

Pro pochopení si nejprve všimněte jednotlivých položek menu. Odkazy

jednotlivých položek mají adresu **index.php?stranka=xxx**. Tato konstrukce (tvar) adresy funguje tak, že do následně volané metody _GET v kódu PHP se doplní právě parametr, který vidíme za rovnítkem. Tedy např. slovo kontakt. Tím, že k tomuto slovu ještě připojíme (tečkou) příponu .php se načte do tohoto místa tento externí soubor. Pokud tedy existuje.

```
require($_GET['stranka'] . '.php');
```

Vytvořte tedy podle tohoto příkladu startovací stránku webu – index.php a dále tři „podstránky“ – uvod.php, reference.php a kontakt.php. Zde je demo.

Po kliknutí na jednotlivé položky sice vše funguje, ale základní startovací stránka webu – index.php vykazuje chybu. Proto je potřeba doplnit skript o podmínku, která kontroluje, zda je v adrese nějaký parametr:

```
<html>
<head>
<title>Příklad na include</title>
<meta charset='utf-8'>
</head>
<body>
<h1>Menu:</h1>
<ul>
  <li><a href="index.php">Úvod</a></li>
  <!-- Zde již můžeme nechat pouze odkaz na index.php -->
  <li><a href="index.php?stranka=reference">Reference</a></li>
  <li><a href="index.php?stranka=kontakt">Kontakt</a></li>
</ul>
<?php
echo "<p>Dynamický obsah:</p>";

if (isset($_GET['stranka']))
// pokud je něco v parametru adresy URL, pak tento parametr načti.
   require($_GET['stranka'] . '.php');
else
// Pokud ne, načti následující kód.
   echo "Toto je obsah titulní strany webu..."
?>
</body>
</html>
```

Podmínka „kontroluje“ zda je v parametru URL adresy nějaký text. Pokud ano, vypíše jej. Pokud ne, vypíše se část else podmínky.

Tady je výsledek.

Pokuste se takovýto příklad vložit do stránky s nějakým frameworkem (např. Bootstrap…)

Příklad možného řešení:

DEMO | Celý web ke stažení

12. Funkce v PHP

Funkce v PHP dělíme na dvě základní skupiny – vestavěné a uživatelské.

Vestavěné funkce

Samotné PHP obsahuje doslova tisíce funkcí, které můžete libovolně používat ve svých skriptech. Těmto tedy říkáme vestavěné funkce. Jedním ze základních příkladů je funkce phpinfo:

```
<?php
phpinfo();
?>
```

Tato vestavěná funkce nám například vypíše kompletní informace o PHP serveru.

Funkci můžeme přidat také určité **parametry**. Ty jsou umístěny v závorkách za názvem funkce. Příklad:

```
<?php
print('ahoj');
// V tomto případě pouze vypíše ahoj.
?>
```

Dále může funkce tzv. něco vracet (například vypisovat).

```
<?php
$cislo = pi();
// Proměnná $cislo bude obsahovat číslo pí

echo $cislo;
// Výpis čísla pí

echo '<br />';
// Zařídí, aby se další vypsalo na nový řádek

echo pi();
// Číslo pí je možné také vypsat přímo
?>
```

Nebo příklad z funkcí max:

```
<?php
$cislo = max(4,1);
// Proměnná $cislo bude obsahovat největší číslo z čísel 4 a 1.
echo $cislo;
// Výpis hodnoty proměnné $cislo
echo '<br>';
```

```
// Zařídí, aby se další číslo vypsalo na nový řádek. I tuto funkci můžeme
vypsat takto:
echo max(4,1);
// Vypíše největší číslo z čísel 4 a 1
?>
```

Příklad využití vestavěné funkce náhodného čísla – mt_rand (elektronická verze házecí kostky)

Funkce mt_rand vrací náhodné číslo. Má 2 parametry, a to minimální a maximální číslo z rozsahu, který má vrátit. V tomto případě 1 až 6.

```
<?php
 $nahodne_cislo = mt_rand(1,6);
  switch ($nahodne_cislo) {
  case '1':
   echo 'Padlo číslo 1';
   break;
  case '2':
   echo 'Padlo číslo 2';
   break;
  case '3':
   echo 'Padlo číslo 3';
   break;
  case '4':
   echo 'Padlo číslo 4';
   break;
  case '5':
   echo 'Padlo číslo 5';
   break;
  case '6':
   echo 'Padlo číslo 6';
   break;
 }
?>
```

Seznam všech vestavěných funkcí PHP lze najít v oficiálním manuálu PHP – http://php.net nebo např zde – http://php.baraja.cz/

Uživatelské funkce

Pro vytváření uživatelských neboli vlastních funkcí se používá klíčové slovo funkcion. Za klíčovým slovem funkcion následuje název (jméno) funkce, které si sami zvolíme. Zjednodušeně to vypadá následovně:

```
<?php
function jmeno_funkce()
{
telo_funkce;
}
?>
```

Nyní si zkusíme konkrétní jednoduchý příklad, kde se pouze vypíše pozdrav Ahoj:

```
<?php
function vypis()
// Zde jsme vytvořili funkci s názvem vypis. Dále následuje její tělo. V tomto případě je to pouze příkaz k vypsání slova Ahoj
{
echo 'Ahoj';
}
vypis();
// Zde voláme funkci s názvem vypis. V tomto případě provede vypsání slova Ahoj.
?>
```

Uživatelské funkce s parametry

Další příklad ukazuje složitější funkci s parametry:

```
<?php
// Deklarace a definice funkce:
function TretiMocnina($Cislo)
{
$Vysledek = $Cislo * $Cislo * $Cislo;
return $Vysledek;
}
// Volání funkce TretiMocnina():
echo TretiMocnina(5); // Vypíše: 125
?>
```

Nejprve je vytvořena funkce s názvem TretiMocnina. Ta obsahuje parametr, který v tomto případě tvoří novou proměnnou s názvem $Cislo. V tělu funkce následuje operace, kdy vytvoříme novou proměnnou s názvem $Vysledek. Tomu je přiřazena operace, která násobí třikrát za sebou proměnnou $Cislo (protože chceme tvořit třetí mocninu, což je trojnásobek). Na dalším řádku máme slůvko returt což je příkaz, který nám vrací hodnotu proměnné $Vysledek. Následuje příkaz echo, který vypíše hodnotu proměnné TretiMocnina s parametrem 5. Neboli do proměnné $Cislo dosadíme tímto číslo 5, které se následně třikrát po sobě vynásobí a vytvoří tedy třetí mocninu.

Další příklad ukazuje, že funkci můžeme ve skriptu použít, kolikrát chceme. Což se velmi často používá:

```
<?php
// Deklarace a definice funkce:
function TretiMocnina($Cislo)
{
$Vysledek = $Cislo * $Cislo * $Cislo;
return $Vysledek;
}
// Volání funkce TretiMocnina():
echo TretiMocnina(5); // Vypíše: 125
```

```
?>
```

Další ukázka uživatelské funkce (zdroj http://php.vrana.cz):

```
<?php
// Vrácení českého názvu měsíce

function cesky_mesic($mesic) {
    static $nazvy = array(1 => 'leden', 'únor', 'březen', 'duben', 'květen',
'červen', 'červenec', 'srpen', 'září', 'říjen', 'listopad', 'prosinec');
    return $nazvy[$mesic];
}
echo cesky_mesic(date("n")) . "\n";

// Vrácení českého názvu dne v týdnu
function cesky_den($den) {
    static $nazvy = array('neděle', 'pondělí', 'úterý', 'středa', 'čtvrtek',
'pátek', 'sobota');
    return $nazvy[$den];
}
echo cesky_den(date("w")) . "\n";
?>
```

Pozn.: znaky **\n se v PHP používají pro odřádkování v kódu. Nikoliv ve webové stránce! Pro tu se musí použít klasické
**

Globální a lokální proměnné

Jakákoliv proměnná, která je použita mimo jakoukoliv funkci, se v PHP nazývá globální proměnná. Zato ta, která je použita uvnitř funkce se nazývá lokální proměnná. Tyto proměnné se navzájem nijak nemíchají a jsou jakoby oddělené. Příklad:

```
<?php
$x = 4;
// Globální proměnná $x nastavena na 4.
function vypis_x()
{
echo $x;
// Tohle je pokus o výpis lokální proměnné $x.
}
echo vypis_x();
?>
```

Po spuštění dostanete asi toto chybové hlášení:

Notice: Undefined variable: x in C:\wamp\www\index.php on line 7

Jde o to, že uvnitř funkce s názvem vypis_x se pokoušíme příkazem echo vypsat proměnnou $x. Takováto proměnná ale uvnitř této funkce neexistuje. Nachází se totiž mimo tuto funkci, tak že ji tento příkaz echo uvnitř funkce marně zkouší vypsat. Proměnná $x sice ve skriptu existuje, ale je je to globální

proměnná, na kterou „zevnitř“ funkce nevidíme.

Poznámky k funkcím

- Funkce **muže mít více vstupních parametrů**. Ty se pak oddělují čárkou.
- Funkce v PHP **mohou předávat parametry odkazem**. To se provede tak, že před název proměnné v hlavičce funkce se uvede **ampersand (&)**
- **V těle funkce mohou být definovány proměnné s klíčovým slovem *static***. Hodnotu takových proměnných si PHP mezi jednotlivými voláními funkce pamatuje.
- Funkce v PHP **mohou být rekurzivní. To znamená, že funkce muže volat sebe samu**.
- Funkce **nemůže vracet více než jeden výstupní parametr**. Ale muže vracet pole, takže se to dá obejít.
- Jedna **uživatelská funkce muže volat jinou. Na pořadí, v jakém jsou uvedeny ve skriptu, přitom nezáleží**. Volání funkce muže být uvedeno dokonce již dříve než definice funkce samotné.

---

11. Pole v PHP

Pole si můžeme představit jako **množinu dat, které vzájemně nějak souvisí**. Například abeceda, skupina čísel, názvy ovoce, skupina žáků ve třídě apod. Jedná se o prostředek pro hromadné zpracování dat. Jinak řečeno, **pole je místem, do kterého můžeme uskladnit libovolné množství dat**. A to do slova. Tedy **velikost pole je závislá pouze na velikosti dostupných prostředků počítače (velikost paměti, databáze…)**.

Vytváření pole

Pole se vytváří za pomoci **konstrukce array**. Příklad:

```
<?php
  $cisla = array(1,2,3);
  // Vytvoření pole, do kterého budou uložena tři čísla: jednička, dvojka a
trojka.
  var_dump($cisla);
  // Vypsání celého pole. Výpis bude vypadat takto: // array(3) { [0]=>
int(1) [1]=> int(2) [2]=> int(3) }
?>
```

V poli můžeme používat i různé kombinace datových typů, např. místo dvojky můžete vložit text „dvojka“ (vyzkoušejte si to).

Práce se záznamy v poli

Se záznamy v poli se dá různě pracovat.

1. Sečtení počtu záznamů v poli

Pro sečtení celkového počtu záznamů v poli existuje jednoduchá **funkce count**. Příklad:

```
<?php
  $pole = array(10,20,30,40);
  // Vytvoření pole, do kterého budou uložena čtyři čísla
  $pocet = count($pole);
  // Funkce count zde zjišťuje počet údajů v poli $pole. Protože pole $pole obsahuje čtyři údaje, funkce count vrátí čtyřku. Proměnná $pocet tedy bude obsahovat čtyřku.
   echo $pocet;
 // Vypíše: 4
?>
```

Tato funkce se velmi často používá. Příklad – vypsání celkového počtu záznamů v návštěvní knize, celkový počet záznamů ve fotogalerii apod.

2. Hledání v poli

Aby bylo možné se záznamy v poli pracovat, je nutné, aby byl každý záznam nějak identifikovatelný (každý záznam má své pojmenování). To se děje vlastně automaticky a to tak, že každý ze záznamů má podle toho jak byl do pole uložen své pořadí. V PHP tomu říkáme **index.** Pořadí či index je v PHP (ale i v jiných prog. jazycích) **počítáno od nuly**. Tedy první záznam má index nula druhý jedna, třetí dva atd… Příklad:

```
$pole = array(10,20,30,40);
//desítka má index nula, dvacítka má index jedna, třicítka má index dva…
```

Pro práci s indexi používá PHP hranatých závorek. Např. – [2] *(klávesová zkratka pravý Alt+F a G)*

Příklad:

```
<?php
  $pole = array(10,20,30,40);
  // Vytvoření pole, do kterého budou uložena čtyři čísla
  var_dump($pole[1]);
  // Vypíše: int(20),
?>
```

3. Změny záznamů v poli

Velmi jednoduchým způsobem můžeme také jednotlivé záznamy v poli měnit. Příklad:

```
<?php
  $pole = array(1,2,3,4,5);
  // Vytvoření pole, ve kterém bude uloženo pět čísel. Jednička má index nula, dvojka má index jedna, trojka má index dva, čtyřka mí index tři a pětka má index čtyři.
```

```
  var_dump($pole);
 // Vypíše celé array(5) { [0]=> int(1) [1]=> int(2) [2]=> int(3) [3]=>
int(4) [4]=> int(5) }
 echo '<br>';
 //další záznamy budou na novém řádku
 $pole[3] = 10;
 // Změna prvku v poli, který má index tři. Na tomto indexu byla původně
čtyřka, teď jsme tam uložili namísto ní desítku. Pole teď vlastně obsahuje
čísla: 1,2,3,10,5.
  var_dump($pole);
  // Vypíše array(5) { [0]=> int(1) [1]=> int(2) [2]=> int(3) [3]=> int(10)
[4]=> int(5) }
?>
```

4. Přidávání nových prvků do pole

Pokud do pole potřebujeme přidat nový záznam, máme k dispozici dva způsoby – přidání nového záznamu automaticky, tedy na konec pole nebo tak, že rovnou napíšeme, jáký index záznamu dáváme. Příklad, kdy vytváříme nový index 2 a do něj vkládáme hodnotu 30:

```
<?php
  $pole = array(1 => 20, 3 => 40, 6 => 70);
  // Vytvoření pole, ve kterém budou uložena tři čísla
  echo $pole[1];
  // Vypíše: 20, protože index 1 má číslo 20
 $pole [2] = 30;
 // Vytvoření záznamu s indexem 2 a vložení čísla 30 do tohoto indexu
 var_dump ($pole);
 // Vypíše: 20array(4) { [1]=> int(20) [3]=> int(40) [6]=> int(70) [2]=>
int(30) }
?>
```

Všimněte si, že nový záznam je vypsán až jako poslední i když to neodpovídá číselné posloupnosti.

Druhým způsobem je vkládání nového záznamu bez udání indexu. **Stačí vložit jen prázdné hranaté závorky**:

```
<?php
$pole = array(1 => 20, 3 => 40, 6 => 70);
  // Vytvoření pole, ve kterém budou uložena tři čísla
echo $pole[1];
// Vypíše: 20, protože index 1 má číslo 20
$pole [] = 30;
// Vytvoření záznamu s indexem 7 a vložení čísla 30 do tohoto indexu
var_dump ($pole);
// Vypíše: 20array(4) { [1]=> int(20) [3]=> int(40) [6]=> int(70) [7]=>
int(30) }
?>
```

V tomto případě se PHP podívá na největší existující index a nově vkládanému

přiřadí index o číslo větší. Tedy v tomto případě 7.

5. Ovlivňování indexů

Indexy v poli můžeme i my sami ovlivnit a to zápisem:

index => hodnota

PHP neklade žádná omezení, indexy můžete tvořit i napřeskáčku. Např.:

```
<?php
$pole = array(7 => 20, 6 => 30);
// Vytvoření pole, ve kterém budou uložena dvě čísla, a to dvacítka a
třicítka.
// Index 5 má dvacítka a index 6 má třicítka.
echo $pole[6];
// Vypíše: 30
?>
```

6. Řetězce jako indexy

Indexy můžeme mít i v podobě textových řetězců. Příklad:

```
<?php
$pole = array('jednička' => 1, 'dvojka' => 2);
// Vytvoření pole, ve kterém budou uložena dvě čísla, a to jednička a dvojka.
// Index 'jednička' má uloženo číslo 1 a index 'dvojka' má uloženo číslo 2.
echo $pole['jednička'];
// Vypíše: 1
?>
```

Mezi indexem pro který je použito celé číslo a indexem, pro nějž je použit řetězec není v principu (z hlediska práce s ním) žádný rozdíl.

7. Procházení polí

Pro procházení polí se používá většinou funkce foreach. Existují ještě jiné funkce, ale ty se používají jen velmi zřídka (např. next() a prev()). Funkce (nebo také konstrukce) **foreach je vlastně cyklus**, který prochází všemi prvky pole a pro každý z nich provede nějakou činnost. Např. jej vypíše. Zjednodušeně to vypadá takto:

```
foreach (pole as proměnná_pro_každý_prvek_pole)
  příkaz_který_se_provede_pro_každý_prvek_pole;
```

Ve skutečnosti to vypadá následovně:

```
<?php
$pole = array(1,2,3);
// Vytvoření pole, které obsahuje tři čísla.
foreach ($pole as $hodnota)
echo 'Nalezen prvek s hodnotou: ',$hodnota,'<br>';
// Konstrukce foreach, která prochází pole $pole a hodnotu každého prvku
```

```
tohoto pole vypíše.
?>
```

Této konstrukci říkáme procházení bez indexu. Další variantou je procházení s indexem. Ke stávajícímu příkladu vlastně jen přidáme požadavek na výpis indexu:

```
<?php
 $pole = array(1,2,3);
 // Vytvoření pole, které obsahuje tři čísla.
 foreach ($pole as $index => $hodnota)
 echo 'Nalezen prvek s indexem: ',$index,', hodnota: ',$hodnota,'<br>';
// konstrukce foreach, která prochází pole $pole a vypíše jak index daného prvku, tak jeho hodnotu
?>
```

Pozn.: proměnná $index může mít jakýkoliv název, tedy ne zrovna „index“. To stejné platí pro proměnnou „hodnota“.

8. Prověření existence pole

Abychom zjistili, jestli v nějakém poli existuje určitý prvek, použijeme funkci ***in_array()***.

Příklad:

```
<?php
$jmena = array("Petr", "Jan", "Gustav", "Pavel");

if (in_array("Jan", $jmena))
  {
  echo "Jan je mezi jmény";
  }
else
  {
  echo "Jan mezi jmény není";
  }
?>
```

9. Přidání a odebrání prvku pole

Přidání na začátek pole

Jak je popsáno výše, pokud přidáme do pole nový prvek, vždy se zařadí na konec – tedy zařadí se jako poslední. Pokud ale chceme, aby se zařadil jako první, můžeme k tomu využít funkci ***array_unshift()***. Příklad:

```
<?php
$jmena = array("Adam","Bára","Tomas");
echo ($jmena[0]." ".$jmena[1]." ".$jmena[2]);
echo "<br>";
array_unshift($jmena,"Karel");
echo ($jmena[0]." ".$jmena[1]." ".$jmena[2]);
```

```
?>
```

V tomto příkladu vidíte, že na počátku má pozici 0 v poli jméno Adam. Po přidání nového jména Karel se toto jméno zařadí funkcí array_unshift na začátek, tedy místo jména Adam. Ostatní prvky pole se posunou.

Odebrání prvního prvku pole

Pokud použijeme funkci ***array_shift***, pak se naopak první prvek v poli smaže a ostatní prvky pole se posunou o pozici vpřed:

```
<?php
$jmena = array("Adam","Bára","Tomas");
echo ($jmena[0]." ".$jmena[1]." ".$jmena[2]);
echo "<br>";
array_shift($jmena);
echo ($jmena[0]." ".$jmena[1]);
//pozice 2 není vypsána, protože neexistuje a skript by vypsal chybu
?>
```

Přidání na konec pole

Velmi podobně se používá funkce ***array_push***:

```
<?php
$jmena = array(1 => "Adam", 3 => "Dan", 6 => "Karel");
print_r($jmena);
echo "<br>";
array_push($jmena,"Hana");
print_r($jmena);
?>
```

Odebrání prvku z konce pole

Pokud potřebujeme odebrat poslední prvek pole, použijeme funkci ***array_pop***:

```
<?php
$jmena = array(1 => "Adam", 3 => "Dan", 6 => "Karel");
print_r($jmena);
echo "<br>";
array_pop($jmena);
print_r($jmena);
?>
```

10. Přidání a odebrání více prvků pole

Rozšíření pole

Pomocí funkce ***array_pad*** můžeme prvky pole rozšířit jak na začátku (záporné hodnoty), tak na konci (kladné hodnoty). Tato funkce má však 3 parametry. Syntaxe funkce:

```
array_pad(pole, množství, prvky)
```

Jako množství se udává **celkový počet prvků, které bude pole tvořit po přidání**. Parametr prvky je hodnota, která bude do celkového počtu vytvořena na zbývajících indexech pole. Příklad:

```
<?php
$jmena=array("Adam","Dan","Karel");
print_r(array_pad($jmena,6,"Vaclav"));
?>
```

Výsledkem bude, že celkový počet prvků v poli bude 6 a poslední 3 budou mít stejnou hodnotu – Václav.

Načtení prvků

Hromadně můžeme načíst prvky pole za pomoci funkce ***array_slice***. Tato funkce má 4 parametry:

```
array_slice(pole, pozice, délka, zachování indexů)
```

Pozice znamená číslo indexu, od kterého se mají začít načítat hodnoty prvků. Délka je nepovinný údaj. Pokud se uvede, znamená to počet indexů, které se mají od pozice načíst. V případě, že není uvedena, načtou se všechny prvky až do konce. Zachování indexů je parametr, který má dvě hodnoty – false nebo true. Pokud použijeme ano, tedy true, zachovají se původní hodnoty indexů daných prvků. Hodnota false je defaultní a je zbytečné ji udávat. Znamená to tedy, že nově vytvořený seznam prvků má novou (svoji) indexaci:

```
<?php
$jmena=array("Adam","Dan","Karel","Bára","Josef", "Tomáš");
print_r(array_slice($jmena,2,2));
//načetli jsme prvky od pozice 2 v počtu 2. Indexy jsou vytvořeny znovu.
echo "<br>";
print_r(array_slice($jmena,2,2,true));
//načetli jsme prvky od pozice 2 v počtu 2. Indexy se zachovaly z původního číslování pole.
?>
```

Nahrazení prvků

Pro nahrazování prvků pole se používá funkce ***array_splice***. Základní syntaxe:

```
array_splice(pole, pozice, délka, nahrazující prvek)
```

Nahrazení je provedeno tak, že se od pozice XY (parametr pozice – např. od pozice 2) v délce XY (délka je počet indexů – např. 3) nahradí prvky původního pole nahrazujícími prvky (většinou definované v proměnné).

```
<?php
$jmena=array("Adam","Dan","Karel","Bára","Josef", "Tomáš");
print_r($jmena);
echo "<br>";
$jmena2=array("David","Petr","Jan");
array_splice($jmena,2,3,$jmena2);
```

```
//Jména Karel a Bára budou nahrazeny novými jmény David, Petr a Jan
//Jedná se o pozice od č.2 v počtu 3. Nová jména jsou načítána z proměnné
jmena2
print_r($jmena);
?>
```

Odstranění prvků

Odstranění prvků se provede stejnou funkcí, jen vynecháme nahrazující prvek:

```
<?php
$jmena=array("Adam","Dan","Karel","Bára","Josef", "Tomáš");
print_r($jmena);
echo "<br>";
print_r(array_splice($jmena,2,3));
echo "<br>Výsledek po odstranění: ";
print_r($jmena);
?>
```

V tomto případě budou od pozice 2 (včetně) odstraněny 3 prvky. Zbytek zůstává beze změn, jen zachované prvky změní svůj index (v příkladu dostane jméno Tomáš nově index č. 2).

Převedení pojmenovaných indexů na číselné

Pokud máme pole, kde máme jednotlivé indexy pojmenované (asociativní pole), můžeme si je jednoduše převést na defaultní – číselné hodnoty funkcí ***array_values***:

```
<?php
$seznam=array("Jméno"=>"Petr","Věk"=>"41","Bydliště"=>"Znojmo");
print_r(array_values($seznam));
//Výsledek - Array ( [0] => Petr [1] => 41 [2] => Znojmo )
?>
```

Převedení pojmenování indexů na jednotlivé prvky

Pokud jsou jednotlivé indexy pojmenované a tyto názvy potřebujeme dostat do pole jako jednotlivé prvky, použijeme funkci ***array_keys***:

```
<?php
$auta=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander");
print_r(array_keys($auta));
//Výsledkem je Array ( [0] => Volvo [1] => BMW [2] => Toyota )
?>
```

Tato funkce má ještě další dva volitelné parametry – hodnota (value) a kontrola typu (strict). První příklad ukazuje hledání hodnoty indexu čísla „15“ bez kontroly typu:

```
<?php
$cisla=array(15,25,35,45,"15");
print_r(array_keys($cisla,"15"));
```

```
//Výsledek je Array ( [0] => 0 [1] => 4 )
//Skript našel číslo 15 na pozici 0 a na pozici 4
?>
```

Druhý příklad ukazuje hledání hodnoty indexu čísla „15“ s kontrolou datového typu. A protože hledaný výraz „15“, tedy číslo, uvedené v uvozovkách není číslem, ale textovým řetězcem, není nalezeno na pozici 0, ale jen na pozici 4:

```
<?php
$cisla=array(15,25,35,45,"15");
print_r(array_keys($cisla,"15",true));
//Výsledek je Array ( [0] => 4 )
//Skript našel výraz "15" pouze na pozici 4. Nově má index 0
?>
```

Záměna hodnot a indexů

Funkcí ***array_flip*** zaměníme hodnoty prvků a indexy:

```
<?php
$pole1=array("a"=>"červená","b"=>"zelená","c"=>"modrá","d"=>"žlutá");
$zmena=array_flip($pole1);
print_r($zmena);
//Výsledek - Array ( [červená] => a [zelená] => b [modrá] => c [žlutá] => d )
?>
```

Obrácené pořadí prvků

Funkce ***array_reverse*** slouží k obrácení pořadí prvků v poli:

```
<?php
$auta=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota");
print_r(array_reverse($auta));
//Výsledek - Array ( [c] => Toyota [b] => BMW [a] => Volvo )
?>
```

Sdružení (sloučení) polí

Na sloučení polí používáme funkci ***array_merge***. Pozor u asociativních (pojmenovaných) indexů – zde mohou nastat chyby.

```
<?php
$pole1=array("červená","zelená");
$pole2=array("modrá","žlutá");
print_r(array_merge($pole1,$pole2));
//Výsledek - Array ( [0] => červená [1] => zelená [2] => modrá [3] => žlutá )
?>
```

PHP disponuje mnoha dalšími funkcemi na práci v poli. Další informace o **polích v PHP** najdete např. na webu itnetwork.cz nebo např. na w3schools.com

10. Cykly v PHP

Cykly jsou konstrukce, které umožňují vykonávat stejné nebo podobné příkazy vícekrát po sobě. Cykly se používají velmi často a nedá se bez nich skoro napsat žádný rozumný skript. Součástí cyklu je i podmínka, která vyhodnocuje, kdy má již provádění cyklu skončit. Někdy potřebujeme, aby se určité cykly opakovaly po určitý počet opakování, třeba desetkrát, Jindy můžeme chtít, aby se cyklus opakoval po určitý čas, další variantou může být např. požadavek, aby cyklus neustále vypisoval určená data z databáze, než se vypíšou všechny apod. Proto existují v PHP pro různé varianty cyklů různé konstrukce:

- cyklus while
- cyklus do while
- cyklus for
- konstrukce switch
- cyklus foreach (viz kapitola Pole v PHP)

Cyklus while

Je to nejjednodušší varianta cyklu. Základní tvar je:

```
while (podmínka)
 příkaz_který_se_bude_cyklicky_opakovat;
```

Princip:

1. Otestuje se podmínka. Pokud není pravdivá, cyklus while končí a PHP pokračuje za cyklem while
2. Pokud je podmínka pravdivá, vykoná se příkaz a cyklus se opakuje.

Jinak řečeno, smyslem cyklu while je neustále dokola opakovat příkaz, dokud platí podmínka. Příkaz se však nemusí provést ani jednou. To v případě, že hned na začátku po testování je podmínka nepravdivá. Příklad cyklu while:

```
<?php
 $i = 1;
 // Proměnné $i je přiřazeno číslo 1
 while ($i <= 5)
 {
 echo $i,' ';
 // Vypiš číslo uložené v proměnné $i a pak vypiš mezeru, aby čísla nebyla
nalepená hned na sobě.
  ++$i;
 // Inkrementace proměnné $i, tedy zvětšení čísla o jedničku.
 }
?>
```

Tento příklad vypisuje čísla od jedné do pěti.

Příkaz break v cyklech

Existuje možnost či potřeba přerušit provádění cyklu v určitém okamžiku. K tomuto účelu existuje příkaz break. Jakmile PHP narazí na tento příkaz, okamžitě cyklus předčasně ukončí:

```
<?php
 $i = 1;
 // Proměnné $i je přiřazeno číslo 1
 while ($i <= 5)
 {
 if ($i > 2)
 break;
 // Pokud bude proměnná $i větší než dvě, vykonej příkaz break
  echo $i,' ';
 // Vypiš číslo uložené v proměnné $i a pak vypiš mezeru, aby čísla nebyla nalepená hned na sobě.
  ++$i;
 // Inkrementace proměnné $i, tedy zvětšení čísla o jedničku.
 }
?>
```

Cyklus do while
Je velmi podobný cyklu while. Jediný rozdíl je v umístění podmínky, která se testuje.

```
do
 příkaz_který_se_bude_cyklicky_opakovat;
while (podmínka);
```

Princip je tedy jednoduchý:

1. Vykoná se příkaz
2. Otestuje se podmínka. Pokud je pravdivá, pokračuje cyklus znovu od začátku

Na rozdíl od cyklu while se cyklus do while vykoná vždy alespoň jednou.

Příklad:

```
<?php
 $i = 1;
 // Proměnné $i je přiřazeno číslo 1
  do
 {
 echo $i,' ';
 // Vypiš číslo uložené v proměnné $i a pak vypiš mezeru, aby čísla nebyla nalepená hned na sobě.
  ++$i;
 // Inkrementace proměnné $i, tedy zvětšení čísla o jedničku.
 }
 while ($i <= 2);
```

```
?>
```

Cyklus for
Cyklus for je nejsložitějším cyklem v PHP, přesto pro svou užitečnost patří k nejpoužívanějším cyklům vůbec. Používáme ho především tam, kde víme, kolikrát chceme cyklus opakovat. Jeho základní tvar vypadá následovně:

```
for (počáteční_příkaz; podmínka; příkaz_který_se_vykoná_po_každém_cyklu)
 příkaz_který_se_bude_cyklicky_opakovat;
```

Jinak také:

```
for(inicializace proměnné; podmínka; operace) {skript}
```

Cyklus probíhá následovně:

1. Nejdříve se vykoná počáteční příkaz
2. Otestuje se podmínka. Pokud je pravdivá, pokračuje se následujícím bodem 3. pokud není pravdivá, cyklus skončí a PHP pokračuje za cyklem for
3. Vykoná se „příkaz, který se bude cyklicky opakovat“ a hned poté se vykoná „příkaz, který se vykoná po každém cyklu“. Pokračuje se znovu bodem 2.

Kód cyklu for (i když je nejsložitější) má velmi jednoduchý kód. Příklad:

```
<?php
 for ($i=1; $i<=5; ++$i)
 echo $i, ' ';
?>
```

Popis:

1. Nejdříve se vykoná počáteční příkaz (inicializace proměnné). Je to první příkaz v závorce. V tomto případě to je přiřazení proměnné $i na číslo 1.
2. Pak se otestuje podmínka, v tomto případě se testuje, zda je proměnná $i menší nebo rovna číslu 5. Pokud je podmínka pravdivá, pokračuje se vykonáním příkazu, který se bude cyklicky opakovat (skript), v tomto případě tedy vypsáním proměnné $i.
3. Dále se pokračuje příkazem, který se vykoná po každém cyklu (operace), v tomto případě je to inkrementace – zvětšení hodnoty o jednu.
4. Následuje opakování cyklu, tedy otestuje se podmínka atd… Jakmile je testovaná podmínka nepravdivá, okamžitě cyklus končí a PHP pokračuje až za cyklem for.
5. Jak již bylo zmíněno, tento cyklus se především využívá v případě, že víte, kolikrát se má daná operace provádět. Např. zákazník v e-shopu může stáhnout tři soubory zdarma. Další už budou placené… apod.

Konstrukce switch
Konstrukce switch slouží k testování (porovnáváním) hodnoty proměnné se zadanými hodnotami. Příklad:

```
<?php
```

```
 $zeme = 'cz';
 // Textový řetězec (v tomto případě zkratka cz) je přiřazena do proměnné
$zeme
 switch ($zeme) {
 case 'cz':
 echo 'Česká republika';
 case 'sk':
 echo 'Slovenská republika';
 case 'pl':
 echo 'Polsko';
 }
?>
```

Vysvětlení:

Textový řetězec (v tomto případě zkratka cz) je přiřazena do proměnné $zeme. Konstrukce switch začne porovnávat obsah proměnné $zeme s jednotlivými hodnotami uvedenými u větví case. V tomto případě je to postupně cz, sk a pl. Jakmile PHP narazí na shodnou hodnotu, vykoná příkaz v dané větvi case. V našem případě je to vždy vypsání daného textového řetězce. Protože v proměnné máme hodnotu ‘cz‘, ve výsledku se vypíše Česká republika.

Protože uvedený příklad funguje tak, že i po vyhledání shodné hodnoty nadále PHP pokračuje v porovnávání a hledání hodnot dalších větví case, **musí se doplnit konstrukce switch o příkaz break**. Příklad:

```
<?php
 $zeme = 'cz';
 // Textový řetězec (v tomto případě zkratka cz) je přiřazena do proměnné
$zeme
 switch ($zeme) {
   case 'cz':
     echo 'Česká republika';
     break;
   case 'sk':
     echo 'Slovenská republika';
     break;
   case 'pl':
     echo 'Polsko';
     break;
   }
?>
```

Jakmile PHP v konstrukci switch narazí na break, další příkazy se již nevykonají a PHP se tak „nezdržuje“ s testováním dalších větví case a pokračuje až za konstrukcí switch.

Další variantou je, že PHP při testování nenalezne žádnou shodnou hodnotu ve větvích case. Pro tento případ se vkládá další větev s názvem default. Příklad:

```
<?php
```

```
 $zeme = 'cz';
 // Textový řetězec (v tomto případě zkratka cz) je přiřazena do proměnné
$zeme
 switch ($zeme) {
 case 'cz':
 echo 'Česká republika';
 break;
 case 'sk':
 echo 'Slovenská republika';
 break;
 case 'pl':
 echo 'Polsko';
 break;
 default:
 echo 'Nejste z České republiky, ze Slovenska ani z Polska';
 }
?>
```

Ještě můžeme konstrukci switch „vylepšit“ o vykonání příkazu pro určitou skupinu. Příklad:

```
<?php
 $zeme = 'cz';
 // Textový řetězec (v tomto případě zkratka cz) je přiřazena do proměnné
$zeme
 switch ($zeme) {
    case 'cz':
    case 'sk':
    case 'pl':
 echo 'Střední Evropa';
 break;
// Země patřící do Střední Evropy
    case 'gb':
    case 'fr':
 echo 'Západní Evropa';
 break;
// Země patřící do Západní Evropy
default:
 echo 'Nejste ze Střední ani Západní Evropy';
 break;
 }
?>
```

---

9. Logické operace v PHP

Často se stává, že potřebujeme kombinovat více podmínek dohromady. Pro tyto

účely existují v PHP tzv. **logické operace**. **Logická operace vyhodnocuje vždy hodnotu ano či ne, resp. true nebo false, nebo také jedničku či nulu**.

Logická negace (!)

Jedná se o nejjednodušší operaci, tedy obrácení podmínky. Provádí vlastně jen to, že z pravdivé podmínky udělá nepravdivou a naopak. V PHP se logická negace zapisuje tak, že podmínce přiřadíme znak vykřičník – !

```
<?php
var_dump(10 > 20);
// Tato podmínka je nepravdivá. Vypíše: bool(false)
echo '<br>';
// Zajistí, aby se další výpis vypsal na další řádku.
var_dump(!(10 > 20));
// Podmínka je pravdivá, protože je použita operace negace, která otočila pravdivost podmínky.
// Vypíše: bool(true)
?>
```

Podmínku, kterou chcete obrátit (negovat) je vždy dobré umisťovat do závorek.

Logické operace and a or

Tyto logické operace slouží ke spojení dvou podmínek tak, aby se výsledek tvářil jako jedna podmínka. Obě operace se liší tím, jak vypočítavají výslednou hodnotu podmínky v závislosti na obou původních podmínkách, ze kterých jsou složeny.

Logickou operaci **and lze jinak také vyjádřit slovním spojením „a zároveň“**. Tato operace považuje dvě podmínky za pravdivé jedině tehdy, pokud jsou obě pravdivé. Oproti tomu logické operaci **or (nebo) stačí, aby alespoň jedna ze dvou podmínek platila**.

Logická operace **and**			Logická operace **or**		
Podmínka 1	**Podmínka 2**	**Výsledek**	**Podmínka 1**	**Podmínka 2**	**Výsledek**
pravda	pravda	pravda	pravda	pravda	pravda
pravda	nepravda	nepravda	pravda	nepravda	pravda
nepravda	pravda	nepravda	nepravda	pravda	pravda
nepravda	nepravda	nepravda	nepravda	nepravda	nepravda

Příklad s logickou operací **and**:

```
<?php
$znamka = 1;
// Zde zadáme školní známku. Vyzkoušejte si sem zadávat různá čísla.
var_dump( $znamka >= 1 and $znamka <= 5 );
// Podmínka je pravdivá, pokud je v proměnné $znamka uložena platná školní známka.
?>
```

Pokud budete do proměnné místo současné jedničky zadávat známky jedna až pět,

bude stále výsledkem pravda. Jakmile zadáte známku nižší než jedna nebo vyšší než pět, stane se minimálně jedna ze dvou porovnávaných podmínek nepravdivou a tím pádem celý výsledek funkce var_dump bude nepravda.

Příklad s logickou operací **or**:

```
<?php
$a = 2;
var_dump($a > 10 or $a == 2);
// Výsledek je pravdivá podmínka, protože alespoň jedna podmínka je pravdivá.
Vypíše: bool(true)
?>
```

Dvojí zápis logické operace and a or

Kromě klasických slov and a or můžete v PHP použít také zápis **&&** (pravý Alt+C) a **||** (pravý Alt+W, programátoři tento znak nazývají „roura“). Tak že např. tyto dva zápisy jsou totožné:

```
$a > $b and $a != 3
//stejné jako:
$a > $b && $a != 3
```

I když jsou obojí zápisy možné, je mezi mini nepatrný rozdíl a to v tzv. prioritě (přednosti). **Slova and a or mají tzv. nižší – příznivou prioritu**, obvykle odpovídající programátorovým záměrům. Obecně je doporučováno raději používat právě tyto slovní vyjádření.

Neúplné vyhodnocování logických výrazů pomocí and a or

PHP se obecně snaží postupovat co nejúsporněji. **Ctí zásadu „zdravé lenosti“**. Právě v případě logické operace and je to velmi zřejmé. Jak již bylo napsáno, je u logické operace and potřeba k pravdivému výsledku, aby obě dvě části podmínky byly pravdivé. V každém jiném případě bude výsledek nepravdivý. Vysvětlení je v následujícím příkladu:

```
<?php
$a = 1;
$b = 2;
$c = 20;
$d = 20;
var_dump ($a > $b and $c == $d);
// Vypíše: bool(false)
?>
```

PHP postupuje tak, že jakmile zjistí, že je první část podmínky nepravdivá (není pravda, že proměnná a je větší než proměnná b), nemusí se již vůbec zabývat tím, co je ve druhé části podmínky (tedy to, že je proměnná c rovna proměnné d). Touto částí se již PHP vůbec nezabývá. Tím se skripty nepatrně zrychlují. V našem případě to nebude zcela jistě patrné, ovšem u složitějších programů to může hrát významnou roli. Tento způsob postupu se nevyužívá jen kvůli rychlosti zpracování, ale také z jiných důvodů, kdy je např. na proměnné navázán další složitější skript apod. (viz dále).

8. Operátory porovnávání hodnot v PHP

Operátorem se nazývá zápis matematické operace. Při použití se ptáme, zda je jedno číslo menší než druhé nebo např. zda se tyto čísla rovnají apod.

Základní operátory porovnávání hodnot:

Příklad	Kdy je podmínka pravdivá
$a == $b	Hodnota proměnné a je rovna hodnotě proměnné b
$a != $b $a <> $b	Hodnota proměnné a není rovna hodnotě proměnné b
$a < $b	Hodnota proměnné a je menší než hodnota proměnné b
$a <= $b	Hodnota proměnné a je menší nebo rovna hodnotě proměnné b
$a > $b	Hodnota proměnné a je větší než hodnota proměnné b
$a >= $b	Hodnota proměnné a je větší nebo rovna hodnotě proměnné b

Příklad:

```
<?php
$a = 100;
// Zkuste zadat i jiné číslo.
if ($a == 100)
//pokud je proměnná rovna číslu 100, pak se vypíše následný text
echo 'Proměnná $a je rovna číslu 100.';
?>
```

Datový typ boolean

Když PHP pracuje s podmínkami, zajímá ho vlastně jen jedno: zda je podmínka pravdivá nebo nepravdivá. Používá k tomu hodnoty true – pravda a false – nepravda. Tomuto typu dat říkáme logický datový typ – v PHP boolean. Pro ilustraci jeden příklad, kdy se hodnota vypíše:

```
<?php
$a = (3 != 4);
// Do proměnné a se uloží výsledek vyhodnocení podmínky, zda se číslo 3
nerovná číslu 4
var_dump($a);
// Vypíše bool(true)
?>
```

Jak již bylo napsáno, funkce var_dump vypisuje informaci o typu dat i její hodnotu (diagnostická funkce). Tak že v tomto případě je to typ dat bool (takto se označuje logický datový typ) a hodnota, tedy true (neboť je pravda, že 3 není rovno 4).

Operátory porovnávání hodnot s kontrolou typu
Kromě základních operátorů, které byly vyjmenovány v předchozí kapitole, PHP

používá ještě dva speciální nejen pro porovnání hodnoty, ale zároveň také pro porovnání typu těchto dat:

`$a === $b`	Hodnota proměnné a je rovna hodnotě proměnné b a navíc jsou stejného typu
`$a !== $b`	Hodnota proměnné a není rovna hodnotě proměnné b nebo nejsou stejného typu

Rozdíl mezi běžným porovnáváním a porovnáváním s kontrolou typu dat je vidět na dalším příkladu:

```
<?php
var_dump(1.0 == 1);
// Podmínka je pravdivá, protože reálné číslo 1.0 se rovná celému číslu 1.
// Vypíše se bool(true)
echo '<br>';
// Zajistí, aby se další výpis vypsal na další řádku.
var_dump(1.0 === 1);
// Podmínka je nepravdivá, protože 1.0 a 1 nejsou shodné typy.
// Číslo 1.0 patří do typu reálných čísel a číslo 1 patří do typu celých čísel.
?>
```

Porovnávání řetězců

Porovnávání čísel není nic složitého. **Ovšem pokud potřebujeme porovnat řetězce (tedy text), je situace o něco složitější.** PHP k tomuto účelu využívá tzv. tabulku ASCII (American Standard Code for Information Interchange). Každému znaku je přiřazen nějaký číselný kód (např. A má přiřazeno 65, B má 66, atd…). Kompletní ASCII tabulku najdete např. na adrese:

http://cs.wikipedia.org/wiki/ASCII

Postup porovnávání řetězců:

1. Začnou se porovnávat znaky obou řetězců od začátku
2. Nejdříve se porovnají první znaky obou řetězců, pak druhé, třetí atd.
3. Jakmile se narazí na první rozdíl nebo jeden z řetězců skončí, je rozhodnuto.
4. Pokud se při porovnávání řetězců narazí na konec jednoho z nich, je ten kratší menší než ten delší.

Podle tabulky ASCII platí, že libovolné velké písmeno je menší než libovolné malé písmenko.

Ukázky porovnávání řetězců:

První řetězec	**Druhý řetězec**	**Výsledek porovnání**
‚abc‘	‚abx‘	**První řetězec je menší než druhý**
‚XX‘	‚xx‘	**První řetězec je menší než druhý**
‚sova‘	‚sova‘	**Oba řetězce jsou shodné**
‚xy‘	‚xyz‘	**První řetězec je menší než druhý**

Složitější porovnávání:

Následující příklad ukazuje porovnávání hodnot s kontrolou typu:

```
<?php
var_dump ('xy' === 'xyz');
// Vypíše bool(false)
?>
```

Výsledkem je výpis bool(false)

Porovnávání kombinovaných řetězců:

Řetězce se porovnávají poněkud složitěji. PHP se totiž nejdříve podívá, jestli jdou oba řetězce převést na číslo. Pokud ano, porovná je jako čísla. Pokud minimálně jeden nejde převést na číslo, porovná je jako texty. Příklad:

```
<?php
var_dump('30' == '+30');
// Podmínka je pravdivá, protože PHP porovná řetězce jako čísla, vypíše se tedy bool(true)
echo '<br>';
// Zajistí, aby se další výpis vypsal na další řádek.
var_dump('30x' == '+30x');
// Podmínka není pravdivá, protože PHP porovná řetězce jako texty, vypíše se tedy bool(false)
?>
```

Ukázky porovnávání složitějších řetězců:

Pozn.: ***řetězce jsou „kousky“ textu, řetězec poznáme podle uvozovek nebo apostrofů.***

Podmínka	**Výsledek podmínky**
,+10.0´ == ’10‘	**pravdivá (porovnávají se jako čísla)**
,2‘ == ,2a‘	**nepravdivá (porovnávají se jako řetězce)**
,abc‘ != ,xyz‘	**pravdivá (porovnávají se jako řetězce)**
,8‘ == ,+8‘	**pravdivá (porovnávají se jako čísla)**

Porovnávání řetězce s číslem

Pokud PHP porovnává řetězec s číslem, chová se jednoznačně tak, že převede řetězec na číslo a pak provádí porovnání. Např.:

```
<?php
var_dump('100' > 40);
// Výsledkem je pravda, protože PHP zde porovnává jakoby se jednalo o dvě čísla.
// Tedy zjišťuje, zda je číslo 100 větší, než číslo 40. Vypíše se tedy bool(true)
?>
```

Převádění řetězce na číslo provádí PHP tak, že se „podívá“ na první znak. Pokud zde nenajde číslo, výsledkem je číslo nula (0). Výjimku tvoří znaky + a -. Ty fungují jako v matematice.

Ilustrační příklady převodu řetězce na číslo:

Řetězec před převodem	Výsledek po převodu PHP na číslo
„10slonů“	**10**
‚postel‘	**0**
‚1.2 m‘	**1.2**
‚333 stříbrných stříkaček‘	**333**
„9e2“	**900 (zápis 9e2 je zápis reálného čísla ve vědecké notaci)**

Nejvíce se převod řetězce na čísla projeví při matematických výpočtech:

Zápis	Výsledek
1 + „1 pes“	**2**
‚1.2 kg‘ – ‚0.8 kg‘	**0.4**
8 * „lev“	**0**
0 + „3e4xxx“	**30000 (zápis 3e4 je zápis reálného čísla ve vědecké notaci)**

Další ukázky porovnávání řetězců s čísly:

Podmínka	Výsledek podmínky
‚+1‘ > 0	**pravdivá** (řetězec ‚+1‘ se převede na jedničku)
’23 psů‘ == 23	**pravdivá** (řetězec ’23psů‘ se převede na číslo 23)
‚-2abc‘ > 10	**nepravdivá** (řetězec ‚-2abc‘ se převede na číslo -2)
0 != ‚cihla‘	**nepravdivá** (řetězec ‚cihla‘ se převede na nulu)

---

7. Podmínky v PHP

Činnost většiny programů (a to nejen v PHP) je řízena tzv. podmínkou. **Podmínka je jednoduše řečeno zápis, který PHP vyhodnotí a zjistí, zda je pravdivý nebo ne**. Další program pak může pokračovat a to jak pro stav, kdy je podmínka pravdivá (je splněna) či naopak nepravdivá (není splněna).

Podmíněný příkaz if

Podmínky v PHP jsou řešeny pomocí **příkazu if**. Dělíme jej na úplný a neúplný.

Úplný podmíněný příkaz if

Úplný podmíněný příkaz je takový, který řeší nejen stav, kdy je daná podmínka

splněna, ale i stav, kdy podmínka není splněna. Celá konstrukce příkazu if vypadá zjednodušeně takto:

```
if (podmínka)
příkaz který se má vykonat pokud je podmínka pravdivá
else
příkaz který se má vykonat pokud podmínka není pravdivá
```

Jak je z příkladu patrno, podmínka, která se má vyhodnotit, je za slůvkem if (volný překlad „pokud“) v kulatých závorkách. Může to být jakýkoliv výraz, kterému PHP rozumí. Může to být nějaká hodnota, třeba číslo, nebo řetězec, proměnná nebo matematická operace, apod. Po vyhodnocení podmínky s výsledkem „pravda“ se vykoná nějaký další krok, umístěný hned na dalším řádku. Pokud však výsledkem vyhodnocení podmínky je nepravda (podmínka není splněna), vykoná se to, co následuje za slovem else (volný překlad „jinak“). Příklad:

```
<?php
 if (10 > 2)
 echo 'Podmínka je pravdivá';
 else
 echo 'Podmínka není pravdivá';
?>
```

V tomto případě se po spuštění vypíše text Podmínka je pravdivá, protože je pravda, že číslo 10 je větší než 2. Pokud si zkusíte zaměnit obě čísla (tedy – 2 > 10), vypíše se text Podmínka není pravdivá.

Pozn.: všimněte si, že se za závorkami nepíše středník!

Neúplný podmíněný příkaz if

Velmi často se v praxi stává, že nepotřebujeme celou funkčnost konstrukce if. V mnoha případech nám stačí, aby se v případě pravdivé podmínky něco vykonalo a v případě nepravdivosti nevykonalo nic. V tomto případě můžeme zcela vynechat část konstrukce else.

```
<?php
 echo 'Toto je kontrola na pravdivost. Pokud bude výsledkem nepravda, nic se nevypíše.';
 // Tento text se vypíše vždy. V tomto případě pouze informuje o činnosti programu

 echo '<br />';
 // zde se zařídí, že se vše následující vypíše na nový řádek

 if (3 > 2)
 //toto je samotná podmínka, která se vyhodnocuje

 echo 'Podmínka je pravdivá';
 //pokud bude pravdivá, tak se tento text vypíše. Pokud ne, nic dalšího se nestane
?>
```

V uvedeném příkladu se to, co je součástí konstrukce if (**všimněte si, že konstrukce if končí až se středníkem!**), tzn. text „Podmínka je pravdivá“ vypíše, protože podmínka byla splněna tedy je pravdivá. Pokud však tato podmínka pravdivá nebude, není zde definován příkaz else a tak se nevykoná nic. Zkuste si nyní vyzkoušet tento příklad tak, že zaměníte obě číslice 3 a 2.

Operátory v podmínce

O operátorech bude zmínka v následující kapitole. Tady je jeden příklad:

```
<?php
$a = 10;
$b = 0;
if ($b != 0)
// Nebude-li proměnná b rovna nule, pak proveď následující výpočet
{
    $vysledek = $a / $b;
    echo('Výsledek dělení: ' . $vysledek);
}
if ($b == 0)
    echo('Nulou nelze dělit!');
?>
```

Zobrazení času s kombinovanou podmínkou:

```
<?php
$presnycas = date('H:i');
$hodina = date('H');
echo ('Na serveru je přesně: '.$presnycas.'</br>Tak že ');
if ($hodina>0 and $hodina<6)
{
echo 'je po půlnoci.';
}
else if ($hodina>6 and $hodina<12)
{
echo 'je dopoledne.';
}
else if ($hodina>12 and $hodina<18)
{
echo 'je odpoledne.';
}
else if ($hodina>18)
{
echo 'je večer.';
}
?>
```

Ukázka načtení obsahu pokud je splněna podmínka (správné heslo):

```
<form action="" method="post">
    Zadej heslo: <input type="password" name="heslo"><br />
```

```
    <input type="submit" value="Odeslat heslo" />
</form>
<?php
$heslo = $_POST['heslo'];
echo "Následující obsah se ukáže jedině pokud znáš správné heslo.";
if ($heslo == 'tajne-heslo')
 echo '<h1>Tady je tajný obsah stránky</h1>';
else
 echo '<p>Neznáš heslo, neuvidíš tajný obsah...</p>';
?>
```

Funkce die

V souvislosti s příkazem if se velmi často používá funkce die, která slouží k okamžitému skončení skriptu PHP. Před tím, než se skript PHP ukončí, se ještě naposledy vypíše nějaká zpráva, která je umístěna uvnitř funkce die:

```
die('Ukončení programu...');
```

Často se tato funkce používá jako reakce na nějakou chybu (příklady později).

---

6. Čísla v PHP

Čísla v PHP

PHP podporuje čísla ve dvojí podobě – **celá** (dat. typ *integer*) a **reálná** čísla (dat. typ *float*).

Celá čísla

Celá čísla jsou jakákoliv čísla, která nemají desetinnou čárku. Příkladem celých čísel může být např. -7, 9, 145, 2 458, -128 523 atd. Celá jsou ovšem v PHP **pouze v rozsahu od -2 147 483 648 do +2 147 483 647**. Jakmile se bude jednat o číslo mimo tento rozsah, bude se automaticky jednat o číslo reálné.

Zápis celého čísla – 3 (jakmile nenapíšete desetinnou tečku v zápisu čísla, jedná se vždy o číslo celé)

POZOR: pokud zapíšete číslo takto – 0x nebo 0X bude to považováno za číslo zapsané v šestnáctkové soustavě. Pokud zapíšete číslo takto – 017, bude to považováno za číslo zapsané v osmičkové soustavě.

Reálná čísla

Reálná čísla jsou čísla, která mohou, ale nemusí obsahovat desetinnou čárku. Na rozdíl od matematiky mají reálná čísla v PHP omezenou přesnost (tedy při výpočtech se zaokrouhlují). V manuálu pro PHP se uvádí, že mají přesnost na 16 číslic. Příkladem pro reálná čísla je např. 9,15 nebo 159,1235 atd.

Zápis reálného čísla – 3.0 (pro desetinnou čárku se v zápisu PHP používá tečka)

Zápis reálného čísla ve vědecké notaci – číslo, které má za sebou písmeno malé „e“ nebo velké „E“, za kterým pokračuje číslo zvané exponent.

```
<?php
echo 12e3;
//vypíše se 12000
?>
```

Proč se vůbec tedy používají celá i reálná čísla, když by vlastně stačila čísla reálná?

- Výpočty s celými čísly bývají o něco rychlejší (tato vlastnost se projeví jen u velmi složitých výpočtů)
- Celá čísla jsou zcela přesná, nezaokrouhlují se, u reálných dochází k zaokrouhlování, tedy nepřesnostem

Matematické výpočty s čísly

PHP umí jak základní, tak složitější matematické operace. Pro začátek jeden z příkladů:

```
<?php
echo 12 + 6
//vypíše 18
?>
```

Jak už bylo zmíněno, mezi základní datové typy patří také čísla. PHP podporuje dva základní číselné typy a to celá a reálná čísla. Základní vlastností je to, že se s nimi dá počítat. Nabídka matematických schopností PHP jazyka nevybočuje z toho, co nabízí většina programovacích jazyků. Zápis matematických operací vychází z jazyka C.

V PHP můžete běžně sčítat, odčítat, násobit a dělit. Základní operace vypadají následovně:

`+` sčítání
`–` odčítání
`*` násobení
`/` dělení
`%` zbytek po dělení

Příklad základních matematických operací:

```
<?php
echo 10 * 2 + 4;
// Vypíše: 24
?>
```

Dělení čísel

Dělení čísel má svá přesná pravidla. Pokud dělíme dvě čísla a výsledkem bude jiné číslo než celé, bude výsledným datovým typem tohoto čísla reálné číslo, tedy číslo s desetinnou čárkou:

```
<?php
$a = 10 / 5;
var_dump($a);
echo "</br>";
// Výsledkem je celé číslo, vypíše int(2)
$b = 10 / 3;
var_dump($b);
// Výsledkem je reálné číslo, vypíše float(3.3333333333333)
?>
```

Pozn.: int a float jsou výrazy pro vyjádření datového typu (int = celé číslo, float = reálné číslo). Více o označování datových typů později.

Operace inkrementace a dekrementace

Jedná se o speciální matematické operace, které zvětšují, respektive zmenšují hodnotu proměnné
o jednu. Princip je v podstatě jednoduchý – **inkrementace (++) způsobí zvětšení** hodnoty proměnné
o jednu a naopak **dekrementace (–) způsobí zmenšení** hodnoty proměnné o jednu. Zápis může vypadat takto:

$x++ **ale také** ++$x

Oba zápisy jsou možné, ale každý se chová poněkud jinak. V prvním případě – $x++ vydá takovýto výraz nejprve stav proměnné x a potom k ní přičte jedničku. Tedy vydá číslo 3. Ve druhém případě – ++$x se nejprve k proměnné x přičte jednička a pak výraz vydá stav této proměnné, tedy číslo 4. Viz následující příklad:

```
<?php
$x = 3;
$y = $x++;
echo $y;
//Vypíše 3
echo "</br>";
echo $x;
//Vypíše 4
?>
```

Přiřazování a jeho další možnosti

Přiřazování zde již bylo probíráno z počátku. Ovšem existují další varianty než jen obyčejné přiřazení hodnot do proměnné.

1. Přiřazení hodnoty do několika proměnných naráz:
   $a = $b = $c = 25;
   Po takovém přiřazení se do všech tří proměnných přiřadí hodnota 25
2. Kombinované přiřazení:

$a = ($b = 3) + 4;
V tomto případě se nejdříve přiřadí hodnota 3 proměnné $b, následně se tato hodnota přičte k číslu 4 a tento výsledek se přiřadí proměnné $a.

3. Přiřazení s operací:
$a = $a + 3;
Zde se proměnné $a přičte číslo 3. PHP však umožňuje zkrácený zápis:
$a += 3;
4. Následující tabulka obsahuje několik příkladů normální a zkrácených zápisů přiřazení proměnné s operací:

Normální zápis	Zkrácený zápis
$a = $a + 33	$a += 33
$a = $a – 5	$a -= 5
$a = $a * 10	$a *= 10
$a = $a / 8	$a /= 8
$a = $a % 3	$a %= 3

---

5. Proměnná v PHP

Proměnné patří mezi základní stavební kameny každého programovacího jazyka.

Definice – proměnná je název pro nějaké místo, do kterého pak můžeme dosadit libovolnou hodnotu.

Příklad – vytváříte program, ve kterém potřebujete několikrát pracovat s číslem 4000000. Jednoduchým přiřazením tohoto čísla do proměnné (např. s názvem x), již pak v samotném běhu programu používáte pouze název této proměnné:

```
<?php
$x = 4000000;
//proměnné x jsme přiřadili číslo 4000000
echo $x * 2;
//výsledkem bude číslo 8000000 – tedy 4 miliony krát dvě
?>
```

V PHP se proměnná značí symbolem **dolaru – $** (pravý Alt a ů), za kterým následuje jméno proměnné (např. $pokus představuje proměnnou s názvem pokus). Vkládání hodnot do proměnné se nazývá **přiřazování hodnot proměnné**.

Do proměnné samozřejmě nemusíme vždy přiřazovat pouze číslo. Může to být i libovolný text neboli řetězec:

```
<?php
$adresa = 'Palackého 125, Nová Paka, 542 12';
```

```
//vytvořili jsme proměnnou s názvem adresa, která obsahuje delší text
echo $adresa;
?>
```

Proměnné se mohou **během programu měnit**. Tedy např.:

```
<?php
$adresa = 'Palackého 125, Nová Paka, 542 12';
echo $adresa;
echo '<br />';
//vypíše adresu a zařídí, aby se vše následující vypsalo na nový řádek
$adresa = 'Sokolovská 28, Praha 8, 111 50';
echo $adresa;
echo '<br />';
//vypíše adresu a zařídí, aby se vše následující vypsalo na nový řádek
$adresa = 'Kamenická 85, Písek 5, 245 20';
echo $adresa;
echo '<br />';
//vypíše adresu a zařídí, aby se vše následující vypsalo na nový řádek
?>
```

Tento příklad vypíše jednotlivé adresy vždy na nový řádek:

```
Palackého 125, Nová Paka, 542 12
Sokolovská 28, Praha 8, 111 50
Kamenická 85, Písek 5, 245 20
```

Názvy proměnných

Každá proměnná má svůj název. Můžeme používat **písmena a také znak podtržítko (_)**. Ovšem název **nesmí začínat číslicí!**

PHP je programovacím jazykem, který je tzv. **„Case sensitive“**, neboli je citlivý na velká či malá písmena. Můžete tedy používat jak malá tak velká písmena, ale pamatujte, že je v každém z nich rozdíl.

České znaky lze také používat, ale velmi důrazně se to nedoporučuje. Obecně lze říci, že můžete používat všechny znaky, které mají ASCII hodnotu od 127 do 255 (ASCII je standardizovaný způsob, který přiděluje každému znaku určité číslo – hodnotu ASCII).

Používejte pokud možno názvy, které alespoň trochu symbolizují to, co daný skript bude vykonávat nebo daná proměnná obsahuje. V programu se pak budete později lépe orientovat.

Když proměnná neexistuje

Pokud se pokusíte v PHP spustit skript s proměnnou, která neexistuje, mohou nastat dvě varianty možností:

1. buď je PHP nastaveno tak, aby vypsalo chybové hlášení (o těchto nastaveních později..)
2. je PHP nastaveno tak, že nevypisuje chybové hlášení – pak se nic

nevypíše

```
<?php
$pokus=1;
//vytvořili jsme proměnnou s názvem pokus, která obsahuje číslo 1
echo $pkus;
//zde je napsáno místo pokus pkus, tedy takováto proměnná neexistuje
?>
```

Pokud je PHP nastaveno na vypisování chybových hlášení v tomto případě vypíše:

Notice: Undefined variable: c in **C:\wamp\www\index.php** on line **4**

(cesta je samozřejmě dle daného umístění souboru)

Zrušení proměnné

Pokud jednou proměnné přiřadíte určitou hodnotu, bude tato proměnná existovat až do konce skriptu PHP. To sice nemusí, ale také může vadit. Pokud tedy potřebujeme určitou proměnnou zrušit, použijeme tzv. kontrukci **unset**, za kterou se do závorek vypíše jméno proměnné, kterou potřebujeme zrušit.

```
unset ($a)
```

Příklad:

```
<?php
$a = 1;
// Proměnná $a teď obsahuje jako hodnotu číslo 1
echo $a;
// Vypíše hodnotu proměnné $a tedy 1
unset($a);
// konstrukce unset zruší proměnnou $a. Tedy nyní již proměnná $a už
neexistuje.
echo $a;
// Nyní (pokud je tak PHP nastaveno) vypíše stránka chybu
?>
```

Proměnné uvnitř řetězce

Následující příklad demonstruje užití vytvořené **proměnné uvnitř** nějakého textu – řetězce.

```
<?php
$jmeno = 'Alík';
// Vytvoříme proměnnou $jmeno a dáme jí jako hodnotu řetězec 'Alík'
echo "Můj pes se jmenuje $jmeno";
// Vypíše: Můj pes se jmenuje Alík
?>
```

Další příklad ukazuje, jak je možné spojit proměnnou s určitým textem:

PHP je benevolentní k umístění znaku dolaru – funguje i varianta, kdy dolar není uvnitř závorek:

```
echo "Máme doma ${jmeno}a";
```

Spojování řetězců

Spojování řetězců je vlastně **spojování kusů textu.** Pro spojování textu se v PHP používá tečka (.), vložená mezi tyto řetězce. Např:

```
<?php
echo 'a'.'b'.'c';
// vypíše se abc
?>
```

Podobně můžete **spojovat proměnné**:

```
<?php
  $znak = 'x';
  echo $znak . $znak . $znak;
  //Vypíše xxx
?>
```

Další variantou může být **připojování** nových hodnot k již vytvořené proměnné:

```
<?php
$a = '1';
  // Proměnná $a obsahuje text: 1
  $a .= '2';
  // Proměnná $a nyní připojí text: 2
  $a .= '3';
  // Proměnná $a nyní připojí text: 3
  echo $a;
  // Nakonec se vypíše: 123
?>
```

Operace s jednotlivými znaky

Každý řetězec má určitý počet znaků. PHP si jednotlivé znaky umí očíslovat. Příklad:

```
$a = abcd;
//Pořadí znaků je tedy a=0, b=1, c=2, d=3 atd.
```

První znak má pořadí 0, druhý 1, třetí 2 atd. Tedy oproti klasickému číslování se u PHP ale i u jiných programovacích jakzyků **začíná nulou**. Pokud potřebujeme nahradit daný znak z určité proměnné, vypadá to následovně:

```
<?php
  $a = '12345';
 $a[2] = 'x';
 // znak na pozici 2 v řetězci je přepsán znakem x.
 echo $a;
```

```
  // Vypíše: 12x45
?>
```

Požadovaná pozice se vkládá do hranatých závorek **[** a **]** (klávesa pravý Alt+F a pravý Alt+G). Nicméně může se používat také složených závorek – **{** a **}** (klávesa pravý Alt+B a pravý Alt+N). Oba zápisy fungují stejně. Ovšem je mezinárodně dohodnuto, že verze PHP 6 bude podporovat již jen první způsob zápisu.

Pokud znak neexistuje

Pokud program požaduje vypsání znaku na pozici, kde nic není (přesněji řečeno, která neexistuje), nevypisuje se nic nebo chybové hlášení (záleží na nastavení výpisu chyb PHP serveru).

```
<?php
  $a = 'ABCDE';
  echo $a[5];
  // Pokus o výpis znaku na pozici 5.
  // Nevypíše nic nebo chybové hlášení (dle nastavení PHP)
?>
```

Připsání znaku v případě, že neexistuje znak na dané pozici

Pokud se pokusíte změnit znak na neexistující pozici, dojde k připsání tohoto znaku do řetězce:

```
<?php
$a = 'abcd';
$a[4] =  'x';
// Písmeno x se dopíše na 4, tedy v tomto případě neexistující pozici.
echo $a;
// Vypíše: abcdx
?>
```

Vyzkoušejte ještě variantu, kdy nebude daná pozice navazovat na řetězec. Např. zadáte pozici 8:

```
<?php
$a = 'abcd';
$a[8] =  'x';
// Písmeno x chceme vypsat na 8. pozici. 4. až 7. ale neexistuje.
echo $a;
// Vypíše: abcd x. Řetězec prostě vytvoří mezeru
?>
```

Určení pořadí znaku pomocí proměnné

O něco složitěji vypadá následující příklad. Namísto přímého uvedení pořadového čísla znaku je použita proměnná:

```
<?php
$a = 'ABCDE';
```

```
$poradi = 3;
echo $a[$poradi];
// Vypíše znak na pozici 3, tedy D
?>
```

---

4. Řetězce v PHP

Řetězce jsou vlastně „kousky textu“. Nebo-li mohou obsahovat cokoliv.

Pro vypsání nějakého textu (řetězce) používáme nejčastěji příkaz echo a to třemi způsoby:

1. řetězec uzavřený do **jednoduchých uvozovek – apostrofů**
2. řetězec uzavřený do **dvojitých uvozovek**
3. použít tzv. **syntaxi heredoc** v případě dlouhých textů

První způsob (apostrofy):

```
<?php
echo 'Toto je příklad řetězce.';
?>
```

Druhý způsob (uvozovky):

```
<?php
echo "Toto je příklad řetězce.";
?>
```

Důležité rozdíly mezi těmito dvěma způsoby zápisu jsou:

- do uvozovek můžeme psát **jméno proměnné (viz dále) přímo**.
- do apostrofů musíme proměnnou zapsat tak, že ji **„obalíme“ tečkami (tzn. konkadence):**

  ```
  <?php
  $body = 15;
  echo 'Pro získání oprávnění musíte mít ' .$body. ' bodů.';
  ?> 
  ```

- **do uvozovek se dají psát speciální znaky jako je např. \n – odřádkování nebo \t – tabulátor a další** (POZOR, prohlížeče neznají konstrukci **\n**, a tak nový řádek vidíme pouze ve zdrojovém kódu).
- **do uvozovek můžeme psát i další uvozovky, ale musí použít tzv. escape sekvenci** (viz níže). Použití uvozovek se tím pádem méně hodí na výpis HTML kódu, protože je složitější.

Třetí způsob:

V případě, že potřebujete vypsat delší text, můžete použít tzv. **syntaxi heredoc.** Syntaxe heredoc začíná trojicí znaků <<<, za kterou následuje nějaký text nazývaný **identifikátor** (v našem případě MMM). Stejný identifikátor musíme použít na ukončení takovéhoto řetězce. Za úvodním identifikátorem nesmí začínat text. Ten musí být umístěn na novém řádku. Ukončovací identifikátor musí být na samostatném řádku, před ním nesmí být ani mezera, za ním musí být středník a nesmí se objevit v daném „dlouhém“ textu.

```
<?php
$text = <<<MMM
Toto je příklad dlouhého řetězce na
více řádků v heredoc syntaxi. Mohou to být
dlouhé věty..... Lorem ipsum dolor sit amet
Consectetuer lobortis lorem amet rutrum
Non Aenean Ut pellentesque condimentum
Accumsan laoreet ut turpis elit
Augue vitae Duis nec Nunc
Eget elit feugiat quis Ut
Vitae lobortis faucibus Aliquam wisi
Nam Donec sociis dictum commodo
Pede feugiat amet convallis metus
Tincidunt parturient quam pretium sapien
Nullam lorem libero vestibulum risus
Et metus sed purus nec
dipiscing est laoreet consectetuer enim
Id sed diam tellus ipsum
MMM;
echo $text;
?>
```

Escape sekvence

Pokud do textu **potřebujete zapsat uvozovky či apostrof**, musíte před takovýto znak vložit tzv. **escape sekvenci**:

```
<?php
echo "Tady bude uvozovka \" a text dále pokračuje.";
// Vypíše: Tady bude uvozovka " a text dále pokračuje
?>
```

PHP v tomto případě ví, že pokud se před uvozovkami či před apostrofem nachází zpětné lomítko, nekončí řetězec, ale je třeba následný znak vypsat.

Pozn.: *o escape sekvencích bude dále ještě zmíňka*

Psaní HTML tagů do řetězců

V rámci psaní textových řetězců je možné také přímo do tohoto textu začlenit i jakékoliv HTML a CSS tagy. Příklad:

```
<?php
```

```
echo 'Tady bude <span style="color: red;">červený text</span>.
<br>
Další text na novém řádku.';
?>
```

Zjištění délky řetězce – strlen

V případě, že potřebujeme zjistit délku řetězce (počet znaků) využívá PHP k této operaci funkci **strlen,** v případě kódování UTF-8 v kombinaci s diakritikou se používá **mb_strlen:**

```
<?php
echo mb_strlen('Řidič');
// Vypíše se číslo 5, protože slovo Řidič má 5 znaků
?>
```

Za funkcí strlen jsou v závorkách předány tzv. **parametry funkce**. V tomto případě je to pouze řetězec ‚ahoj'. Zápis funkce se odborně nazývá **volání funkce.** Výsledkem je vypsání čísla 4, jinak řečeno **funkce vrací hodnotu**.

Funkci strlen můžeme využít i pro vypsání délky řetězce, pocházejícího z proměnné:

```
<?php
$a = 'Miloslav Boudný';
// Přiřazení řetězce do proměnné a.
echo mb_strlen($a);
// Vypíše délku řetězce v proměnné $a, tedy číslo 15.
?>
```

Odstranění bílých mezer – trim

Další praktickou vestavěnou funkcí je **trim**.

Funkce trim odstraní tzv. bílé znaky ze začátku a z konce řetězce.

Za bílé znaky jsou považovány **mezery, tabulátory, konce řádků a další znaky**. Lze využít např. u zadávání textových polí do formuláře a jejich kontrolu.

Defaultně funkce trim odstraní tyto znaky:

- (ASCII 32 (0x20)), obyčejná mezera.
- (ASCII 9 (0x09)), tabulátor.
- (ASCII 10 (0x0A)), nová řádka (line feed).
- (ASCII 13 (0x0D)), návrat vozíku (carriage return).
- (ASCII 0 (0x00)), znak NUL.
- „x0B“ (ASCII 11 (0x0B)), vertikální tabulátor.

Malá ukázka:

```
<?php
$text1 = " Karel ";
```

```
$text2 = "      Svoboda  ";
echo "Vaše jméno a příjmení je ".$text1." ".$text2;
echo "<br>";
echo "Vaše jméno a příjmení je ".trim($text1)." ".trim($text2);
// Je třeba si zobrazit zdrojový kód, jinak neuvidíte rozdíl.
?>
```

Funkce trim má také možnost zadání parametru

V následujícím příkladu je např. odstraněno písmeno K a S. Jedná se o znaky, které musí být na začátku.

```
<?php
$text1 = "Karel";
$text2 = "Svoboda";
echo "Vaše jméno a příjmení je ".trim($text1,"K")." ".trim($text2,"S");
?>
```

Více informací k funkci trim

Příbuzné funkce:

- ltrim() – odstraní bílé mezery zleva
- rtrim() – odstraní bílé mezery zprava

Získání části řetězce – substr

Další často používanou funkcí je funkce **SubStr**. Ta slouží k získání části řetězce. **Má 3 parametry – řetězec, pozici 1. znaku a délku** (oddělené čárkou):

substr($řetězec, pozice 1. znaku, délka)

Vrací část řetězce určené délky od daného znaku. Pokud je pozice 1. znaku záporné číslo, bude se pozice počítat od konce. Pokud je záporná délka, bude řetězec tolik znaků od konce končit. Není-li délka vůbec zadána, funkce vrátí vše od prvního znaku do konce řetězce. Tady je několik příkladů:

```
<?php
$text = "Caste hrani pocitacovach her nici krcni pater.";
echo substr($text, 6)."<br>"; // vrací znaky od 6 pozice - hrani pocitacovach
her nici krcni pater.
echo substr($text, -6)."<br>"; // vrací znaky od 6 pozice zleva (od konce)-
pater.
echo substr($text, 6, 5)."<br>"; // vrací znaky od 6. pozice v počtu 5
dalších znaků - hrani
echo substr($text, 6, -12)."<br>"; // vrací znaky od 6. pozice a z tohoto
ještě odstaní 12 znaků od konce - hrani pocitacovach her nici
echo substr($text, -6, 5)."<br>"; // vrací znaky od 6 pozice od konce v počtu
5 znaků - pater
echo substr($text, -6, -3)."<br>"; // vrací znaky od 6 pozice od konce a
poslední 3 znaky ještě taky odstraní - pat
?>
```

Nahrazení textového řetězce – str_replace

Asi nejpoužívanější funkcí je pak funkce **str_replace**. Obsahuje tři parametry – CO nahradit, ČÍM nahradit, KDE nahradit (oddělené čárkou):

str_replace(,CO‘,’ČÍM‘,’KDE‘);

Příklad:

```
<?php
$text = str_replace('ing.','Ing.', 'Tituly magistr - Mgr. a inženýr - ing. jsou na stejné úrovni. Píšou se vždy s velkým písmenem. Dříve se titul inženýr psal malými písmeny.');
echo $text;
// pokud se v textu objeví "ing.", nahradí se textem "Ing.".
?>
```

Zjištění pozice textu – strpos

Další podobnou funkcí je pak funkce **strpos** (pro UTF-8 opět s prefixem mb_). Jedná se o zjištění pozice, na které se nachází hledaný text, přesněji řečeno **za kterou začíná hledaný text**:

strpos($text, „Hledaný text“);

```
<?php
$text = "Není to PHP výborný jazyk?";
echo (mb_strpos($text, "PHP"));
// vrací hodnotu 8 - pozice, za kterou začíná text "PHP"
?>
```

Počítají se samozřejmě i mezery.

Pozor na diakritiku!

Pozor na diakritické znaky. Minimálně u funkcí strlen a strpos není tzv. ***„multibyte safe„***, což znamená, že „neumí“ počítat diakritické znaky správně. Proto se používá jejich novější podoba – **mb_strlen, mb_strpos**, která již tuto diakritiku počítá správně.

Více o Multibyte Safe.

3. Podrobné výpisy v PHP

Pro výpisy a nejen ty podrobné využíváme především tyto tři způsoby:

1. příkaz **echo**
2. funkce **print_r**
3. funkce **var_dump**

Příkaz echo

Nejčastější metoda pro výpis dat. Dobře postačí na jednoduché datové typy a to čísla a řetězce.

Funkce print_r

Je to daleko univerzálnější řešení, dokáže vypsat jakoukoliv hodnotu a to i tam, kde příkaz echo nestačí.

Rozdíly mezi echo a print_r:

- **zápis** – print_r je funkce a proto se musí hodnota vypisovat do závorek (tak jako u každé jiné funkce), zatímco u příkazu echo se závorky používat nemusejí, ale mohou.
- **počet vypsaných hodnot** – echo umí vypsat více hodnot najednou, kdežto funkce print_r umí vypsat pouze jednu
- **výpis složitějších dat** – echo umí pouze řetězec a číslo. Print_r umí cokoliv

Funkce var_dump

Další možností pro výpis dat je funkce **var_dump**. Tato funkce se liší tím, že kromě samotných dat vypíše i typ, o který se jedná:

```
<?php
var_dump("Adam");
//vypíše: string(4) "Adam"
?>
```

Slovo string je anglický název pro řetězec a PHP jej používá kdykoliv chce říci, že pracuje s řetězcem. Za slovem string je v závorce uvedena číslice 4, čímž je řečeno, že obsahuje 4 znaky. Pak následuje v uvozovkách již vlastní řetězec, tedy Adam.

Proto této funkci říkáme **diagnostická**, jejímž účelem je především podat maximální informace o nějakých datech v PHP. Pro běžné výpisy se využívají první dva předchozí způsoby.

Vyzkoušejte např. tyto další příklady. Vložte za slovo Adam jednu či více mezer a sledujte, jaký je mezi nimi rozdíl a to včetně zobrazení zdrojového kódu:

```
<?php
var_dump("Adam     ");
// ale vypíše:
string(9) "Adam "
?>
```

---

2. Datové typy v PHP

Co je to datový typ – je to typ dat, které bude daná funkce, příkaz, skript apod. zpracovávat. V první řadě je třeba umět data vypisovat.

V PHP existuje **8 základních datových typů, které dělíme na**:

Skalární datové typy

Obsahují pouze **jeden prvek informace** (např. jedno číslo, jeden text).

- ***Boolean*** – logický datový typ. Jedná se o hodnotu *pravda/nepravda*, v zápisu PHP se zadává *TRUE* nebo *FALSE* (na velikosti písmen nezáleží).
- ***Integer*** – celočíselný typ. Jsou to celá kladná i záporná čísla. Rozsah hodnot závisí na platformě, kde PHP běží; obvykle se hodnota pohybuje někde okolo dvou miliard (32-bit signed).
- ***Float*** – desetinné číslo. Rozsah tohoto typu proměnné také závisí na platformě, kde PHP běží obvykle je k dispozici 1,8e308 hodnot (64-bit signed).
- ***String*** – řetězec (text). Řetězce jsou v PHP kódovány 1 bajtem, takže mohou obsahovat 256 znaků.

Složené datové typy

Na rozdíl od předchozích typů mohou obsahovat složená data **více prvků téhož typu** (např. 5 jmen jakožto 5 textových řetězců).

- ***Array*** – pole. Jedná se o složený typ, který může obsahovat i ostatní datové typy.
- ***Object*** – objekt. Objektová proměnná může obsahovat různé prvky PHP, např. proměnné, funkce a konstanty.

Speciální datové typy

- ***Resource*** *(v PHP 8 již není podporován)* – obvykle se používá k interakci s externím zdrojem dat (např. databází). Pokud např. potřebujeme otevřít soubor(fopen), pak funkce vrací prostředek, který slouží k výběru zadaného souboru.
- ***Null*** – „nic“ (prázdná hodnota). Jedná se o zvláštní datový typ, který nemá žádnou hodnotu. Hodí se třeba tehdy, pokud není nějaká proměnná inicializována nebo pokud byla smazána funkcí unset(). Je také možné libovolné proměnné přiřadit hodnotu null dodatečně.

Zatímco *string*, *integer* a *float* můžete jednoduše vypsat příkazem print, u ostatních typů to není možné. Tam je to možné vypisovat mnoha jinými způsoby, k nejuniverzálnějším patří funkce ***var_dump()*** a ***var_export()***.

V PHP verzi 8 jsou další, nové datové typy nebo naopak některé jsou potlačeny.

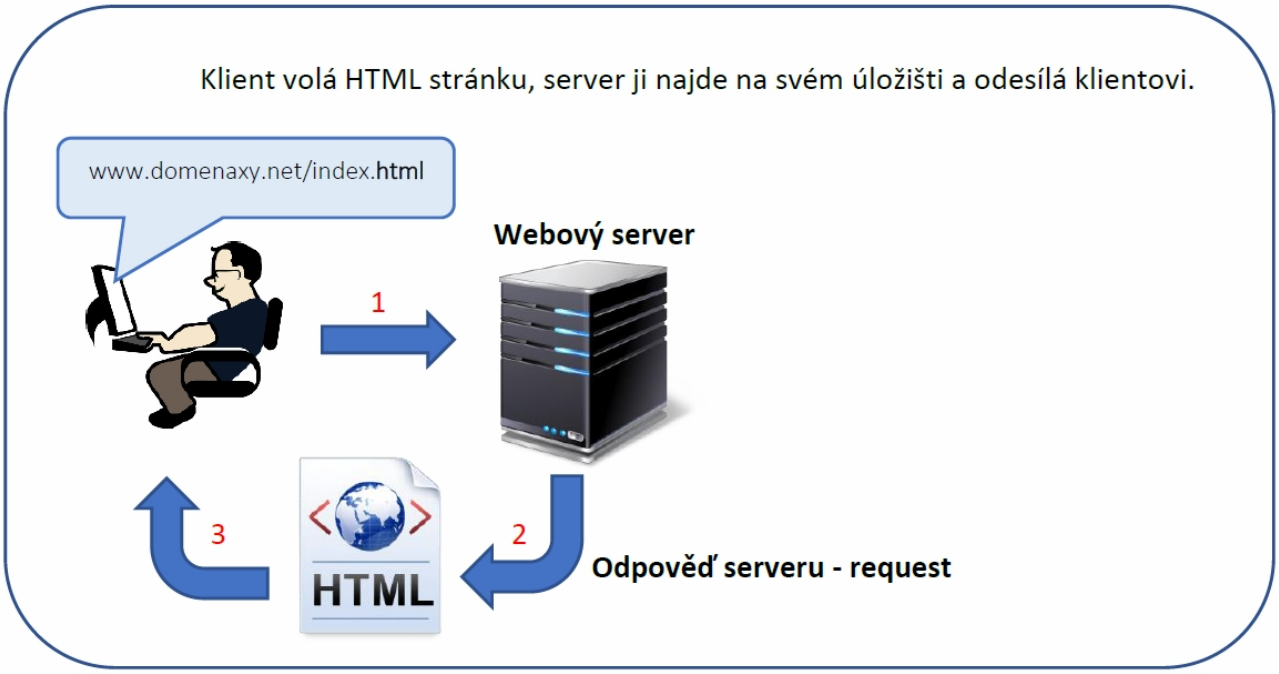
---

1. Úvod do PHP

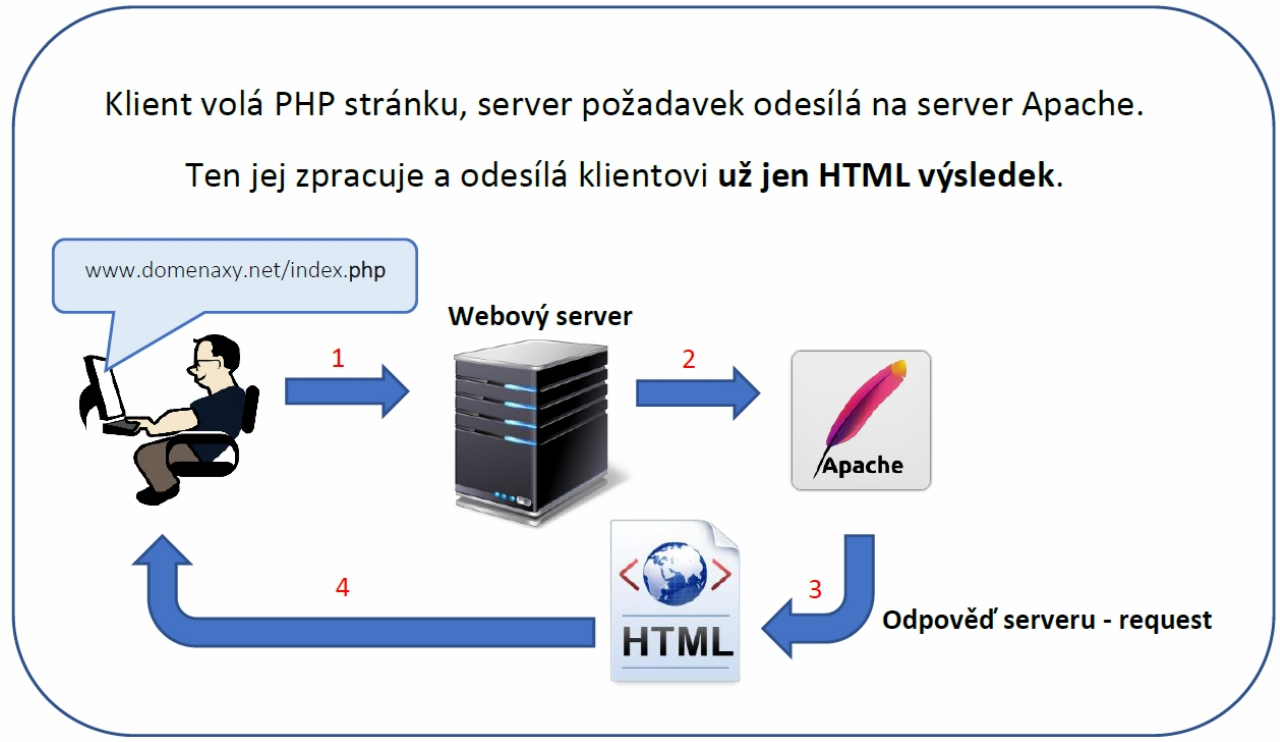
Definice PHP – je to serverový skriptovací programovací jazyk („PHP: Hypertextový preprocesor“, původně *Personal Home Page*), určený především pro programování dynamických internetových stránek. Nejčastěji se začleňuje přímo do struktury jazyka HTML, XHTML či WML (tvorba on-line dokumentů pro mobilní zařízení).

Základní rysy PHP:

- **Skripty jsou prováděny na straně serveru** – k uživateli je přenášen až výsledek jejich činnosti v HTML podobě. Princip je následující:
  Klasická HTML stránka



PHP stránka



- Syntaxe jazyka je **inspirována několika programovacími jazyky (Perl, C, Pascal a Java).**
- **PHP je nezávislý na platformě** (Linux, Windows…).
- **PHP podporuje přístup k většině databázových systémů** (databází, mj. **MySQL, Oracle, PostgreSQL, Microsoft SQL**) a celou řadu internetových protokolů (HTTP, SMTP, SNMP, FTP, IMAP, POP3, LDAP…).

PHP je vedle ASP(X) jedním ze dvou nejrozšířenějších skriptovacích jazyků pro web. Oblíbeným se stal především díky jednoduchosti použití, bohaté zásobě funkcí a tomu, že kombinuje vlastnosti více programovacích jazyků a nechává tak vývojáři částečnou svobodu v syntaxi.

Historie jazyka PHP

Jeho počátky spadají do roku 1994. Tehdy se pan Rasmus Lerdorf rozhodl vytvořit jednoduchý systém pro počítání přístupu ke svým stránkám; bylo to napsáno v PERLu. Za nějakou dobu byl systém přepsán do jazyka C,, protože perlovský kód dost zatěžoval server. Sada těchto skriptů byla ještě později téhož roku vydána pod názvem „Personal Home Page Tools“, zkráceně PHP. Koncem roku 1998 byla již k dispozici verze PHP 3.0, která byla mnohem rychlejší (a vybavenější) než „dvojka“ a která byla k dispozici rovněž pod operačními systémy Windows.

Několik zajímavých statistik:

- weby využívající PHP – http://news.netcraft.com
- jak se využívají verze PHP – http://w3techs.com
- využití programovacích jazyků na webu – http://w3techs.com

Proč ano a proč ne

Proč je PHP tak oblíbené? K tomu vede celá řada věcí:

- PHP je relativně jednoduché na pochopení
- PHP má syntaxi velmi podobnou jazyku C a je tedy většině vývojářů dost blízký
- PHP podporuje širokou řadu souvisejících technologií, formátů a standardů
- je to otevřený projekt s rozsáhlou podporou komunity
- dají se najít kvanta již hotového kódu k okamžitému použití nebo funkční PHP aplikace. Podstatná část z hotového kódu je šířena pod nějakou svobodnou licencí a dá se použít ve vlastních projektech
- PHP si dobře rozumí s webovým serverem Apache (aby ne, vždyť je to sesterský projekt spravovaný Apache software foundation)
- PHP snadno komunikuje s databázemi, jako je MySQL, PostgreSQL a řada dalších
- PHP je multiplatformní a lze jej provozovat s většinou webových serverů a na většině dnes existujících operačních systémů
- PHP podporuje mnoho existujících poskytovatelů webhostingových služeb

Má PHP také svoje nevýhody?

Víceméně ne; pokud budete chtít napsat dynamický web, bude PHP prakticky vždy dobrou volbou. S některými věcmi byste ale přesto měli při použití PHP počítat:

- PHP je interpretovaný, ne kompilovaný jazyk
- kdokoli má přímý přístup k serveru, může nahlédnout do vašich PHP skriptů
- Podpora objektového programování není v PHP na moc dobré úrovni.
- protože je PHP aktivně vyvíjen, v budoucích verzích jazyka se mohou některé funkce změnit nebo se mohou chovat jinak než dosud.

Co se v PHP dá napsat ?

Lakonická odpověď by zněla: **Téměř všechno, co nějak souvisí s dynamickým webem a/nebo s databázemi**. Mezi nejčastější aplikace psané v PHP patří například:

- internetové obchody
- podnikové informační systémy (ať už intranetové nebo internetové)
- diskusní fóra
- redakční systémy
- firemní prezentace
- dynamické osobní stránky
- weboví poštovní či databázoví klienti
- vyhledávače a katalogy
- drobnosti typu počitadla, ankety a mnoho dalších

(Zdroj: http://www.linuxsoft.cz/, https://cs.wikipedia.org)

Pro běh a testování PHP je třeba, aby na daném počítači byl nainstalován **minimálně PHP** a **webový server (např. Apache)**. Pro plnohodnotné využití je však většinou **doplněn o databázový systém MySQL**. Pro kombinaci s operačním systémem Linux se vžila zkratka **LAMP**, pro Windows systémy pak **WAMP**. Existuje

mnoho „balíčků“, kterými zmíněné tři komponenty velmi jednoduše nainstalujeme současně (Laragon, XAMPP, PHP Triad, EasyPHP, WAMP a další).

Pro editaci a tvorbu můžeme použít jakýkoliv textový editor. Většinou se však využívá některých editorů, které jsou vybaveny určitými vylepšeními, např.:

- zvýraznění barevné syntaxe,
- možnost otevření více souborů najednou,
- vyhledávání a nahrazování textu,
- možnost zobrazení čísla řádků,
- možnost otevření libovolně velkého souboru apod.

Takovýmto editorem je např. **PSPad**. Mezi další patří – rkEdit, emacs, HTML-Kid, jEDIT, Vim a mnoho dalších.

Soubory PHP mají v drtivé většině koncovku *.php. Dříve se také užívalo i jiných koncovek např. *.php3, *.php4, *.php5, *.phtml.

Chceme-li tedy zapsat kód PHP do webových stránek, můžeme to udělat celkem 4. způsoby. Nicméně ne všechny fungují na každém serveru. Způsob, který se používá v drtivé většině je způsob č. 1.

Známe tyto způsoby zápisu PHP:

1. **způsob (tzv. velké oddělovače) – nejčastější způsob**

```
<?php
echo 'Nějaký libovolný text';
?>
```

2. **způsob**

```
<?
echo 'Nějaký libovolný text';
?>
```

3. **způsob (musí být podporováno serverem a jeho nastavením)**

```
<script language="php">
echo 'Nějaký libovolný text';
</script>
```

4. **způsob – ASP styl (musí být podporováno serverem a jeho nastavením)**

```
<%
echo 'Nějaký libovolný text';
%>
```

Psaní komentářů v PHP:

1. způsob

```
// komentář na jeden řádek
```

2. způsob

```
# komentář na jeden řádek
```

3. způsob

```
/* komentář
 na více
řádků */
```

Jak vložit video na pozadí stránek

Pro pozadí stránek můžeme použít i video. Dnes nejčastěji uložené na Youtube.com. Použití těchto videí je však značně problematické. proto je vhodnější si potřebné video stáhnout a používat vlastní adresu. Funkční stránka může vypadat např. takto:

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
video {
  object-fit: cover;
  width: 100vw;
  height: 100vh;
  position: fixed;
  top: 0;
  left: 0;
}
html, body {
  height: 100%;
}
html {
  font-size: 150%;
  line-height: 1.4;
}
body {
  margin: 0;
}
.viewport-header {
  position: relative;
  height: 50vh;
  text-align: center;
  display: flex;
  align-items: center;
  justify-content: center;
```

```
}
h1 {
  color: #ac9272;
  text-transform: uppercase;
  letter-spacing: 2vw;
  line-height: 1.2;
  font-size: 3vw;
  text-align: center;
}
main {
  width: 80vw;
  left: 10%;
  overflow: auto;
  background: rgba(black, 0.66);
  color: white;
  position: relative;
  padding: 1rem;
  padding: 20px;
  border: 1px solid #ac9272;
  text-align:center;
  color: #ac9272;
  font-weight: bold;
}
</style>
</head>
<body>
<video src="https://oaznojmo.eu/demo/video-na-pozadi/videopozadi3.mp4"
autoplay loop playsinline muted></video>
<header class="viewport-header">
  <h1>Ukázka videa na pozadí</h1>
</header>
<main>oaznojmo.eu</main>
</body>
</html>
```

Výsledek takto vloženého videa najdete v této ukázce.

Základy HTML

Základní zdroje informací najdete např. na těchto stránkách:

- http://w3schools.com/html/
- http://jakpsatweb.cz/
- https://itnetwork.cz
- http://linuxsoft.cz/
- http://owebu.bloger.cz/

- http://polopate.jakpsatweb.cz/
- http://tutorials.cz
- http://tutorialy.com/
- http://tvorba-webu.cz/
- http://webtvorba.cz/
- http://webtvorba.howto.cz/rubriky/php/
- a mnoho dalších

Proč (ne)mít web zdarma