PHP pro SPŠ a VOŠ Kladno

Jaroslav Holeček
holecek@spskladno.cz

Poslední úprava: 3. května 2020

Obsah

1 Obecné

1.1 Kód, který píše kód

Kromě jiného budeme PHP využívat k tomu, abychom nemuseli ručně vypisovat dlouhý a opakující se HTML kód - třeba velkou tabulku.
Napíšeme si proto PHP skript, který po spuštění napíše tento HTML kód „za nás“. Obvykle budeme pomocí PHP vytvářet pouze část HTML kódu a část napíšeme ručně.

1.2 Jak to funguje

Při použití PHP nám do hry vstupuje další prvek - a tím je server, který spouští naše PHP skripty. Obyčejný prohlížeč neumí PHP zpracovat.
Na této škole máme server Xeon na adrese xeon.spskladno.cz
Xeon má přístup k Vaší složce „public_html“ na disku H: a v prohlížeči si můžete přes Xeon tuto složku zobrazit na adrese: xeon.spskladno.cz/ vas_login (musíte na složce nastavit příslušná přístupová práva)
A jak to celé funguje?

1. PHP skript (soubor s naším php kódem) máme uložený na našem počítači (případně je uložený na serveru).
2. Tento skript předáme serveru a ten zpracuje veškeré PHP příkazy. Zbylý obsah souboru (mimo bloky <?php ...?>) pouze opíše.
3. Z výsledků příkazů v php blocích a textu mimo tyto bloky vznikne na serveru nový soubor - tentokrát již obyčejný textový (obvykle html)
4. Výsledný textový (html) soubor server odešle do našeho prohlížeče, který nám ho zobrazí.

1.3 Rozdělení kódu

PHP kód nemusíme mít v souboru pohromadě v jednom bloku „<?php ...?>“. Takových bloků můžeme mít v souboru libovolné množství - a všechny „se počítají“ jako jeden kód.
Můžeme tedy zapsat do PHP jen ty části, které chceme měnit a ty, které zůstávají stále stejné mohou být pevně napsané v html.

```
1  <html>
2  <body>
3
4  <?php
5  $default_value=15;
6  ?>
7
8  <form action="soubor_pro_zpracovani.php" method="post">
9  Počet řádků: <input type="number" name="radky" value=<?php echo $default_value; ?><br>
10 Počet sloupců: <input type="number" name="sloupce" value=<?php echo $default_value; ?>><br>
11 <input type="submit" value="Zobraz tabulku">
12 </form>
13
14
15 </body>
16 </html>
```

řádky 5, 9, 10: Tři bloky php. Tyto bloky jsou součást jednoho kódu - tedy to, co uděláme v jednom bloku, můžeme využít v jiném (zde např. uložená proměnná)

Zdrojový kód 1: Rozdělení kódu

1.4 Využití dat po obnovení stránky

Pokud obnovím stránku (znovu načtu, refresh) ztratím veškeré proměnné (jejich hodnoty), které jsem doposud v kódu použil.
Při každém načtení stránky - tedy i po obnovení - se vše provádí odznovu - od prvního řádku a nezústává nic zachováno (uloženo).
Pokud potřebuji využít data, která jsem získal v předchozím zobrazení stránky, musím je z původní stránky odeslat na stránku, kterou zobrazuji nyní (i když je to stejná stránka - soubor). Také si mohu uložit data mimo stránku - do PC nebo na server
Odeslat data mohu např. pomocí metody „post“ (využíváme ve formulářích). Uložit data mohu pomocí cookies, případně session.

1.5 Proměnné

Proměnné v kódu si můžeme představit jako pojmenované „krabičky“, do kterých si můžeme v průběhu programu ukládat různé informace.
Veškeré proměnné v PHP musí začínat $ dolarem.
PHP je „loosely typed language“. Pro nás to znamená především to, že nmusíme při vytváření proměnné zapisovat, jakého typu budou data, která do ni budeme ukládat. Dokonce se datový typ může v průběhu progrmau měnit - POZOR na to!

2 Datové typy

V hardwaru počítače neexistují zvláštní místa pro ukládání čísel, nápisů, obrázků, písniček apd..
Vše je uloženo ve stejné mechanické součástce - paměti (operační, cash, pevný disk).
Z pohledu programu a programátora si můžeme představit, že je vše uloženo ve formě 0 a 1 - např. v ASCII znamená 1000001: "A", 1110111: "w", 0111000: "8".
Zda jsou v počítači uloženy znaky ("Aw8"), čísla (420), nebo třeba písnička či obrázek pozná program podle datového typu.
Datový typ je tedy označení, **co znamenají** uložená data (**jedničky a nuly**) - zda je to číslo, znak, obrázek, ...
;

2.1 Odkazy

`https://www.w3schools.com/pHP/php_datatypes.asp`

2.2 Datové typy v PHP

V nemusíme zapisovat, jaký datový typ má být v proměnné uložený - dokonce se může v průběhu programu datový typ proměnné měnit.
PHP přiřadí datový typ automaticky - podle toho, co uložíme do proměnné.
Zjistit, jakého datového typu je nějaká proměnná můžeme pomocí funkce „var_dump()“.

```
1 <?php
2 $x = 8;
3 var_dump(x);
4
5 $x = "Ahoj";
6 var_dump(x);
7 ?>
```

řádek 2: Uloží do proměnné $x hodnotu 8 a rovnou přiřadí datový typ integer

řádek 5: Přepíše hodnoty proměnné $x na „Ahoj“ a změní datový typ na string

řádek 3, 6: Jakého datového typu a jaká hotnota je uložena v proměnné nám vrátí funkce var_dump()

Zdrojový kód 2: Type

2.2.1 Integer - int

Integer označuje **celé číslo**.
Jeho zkratka je „int“.

```
1 <?php
2 $pocet_mesicu = 12;
3 $bod_mrazu = 0;
4 $teplota_venku = -30 ;
5 ?>
```

řádek 2: Integer může být kladné celé číslo, záporné celé číslo nebo nula.

Zdrojový kód 3: Integer

Operace s int S celými čísly můžeme provádět:

\+ Sčítání

\- Odčítání

* Násobení

/ Dělení - výsledek je **float - desetinné číslo**

// Celočíselné dělení - výsledek je celé číslo - znáte ze 3. třídy ZŠ

% Zbytek po celočíselném dělení - znáte ze 3. třídy ZŠ

2.2.2 Float - float

Float označuje **desetinné číslo**, číslo s desetinnou čárkou (plovoucí desetinnou čárkou - odtud slovo „float“).
Takové číslo můžeme do proměnné uložit přímo („pozor - s desetinnou tečkou“). Také ho můžeme získat jako výsledek dělení.
Jeho zkratka je „float“.

```
1  <?php
2  $pocet_odpracovanych_hodin = 14.5;
3  var_dump(pocet_odpracovanych_hodin);
4
5  $pocet_mesicu = 12;
6  $rocni_plat = 1200;
7  $mesicni_plat = $rocni_plat / $pocet_mesicu;
8
9  var_dump($pocet_mesicu);
10 var_dump($rocni_plat);
11 var_dump($mesicni_plat);
12 ?>
```

řádek 2: Desetinné číslo (Float) zapisujeme s **desetinnou tečkou**

řádek 7: Pokud vydělíme dvě celá čísla, získáme desetinné číslo - i když jdou vydělit beze zbytku.

Zdrojový kód 4: Float

Operace s float S desetinnými čísly můžeme provádět:

\+ Sčítání

\- Odčítání

* Násobení

/ Dělení

2.2.3 Boolean - bool

Boolean označuje **logickou hodnotu - Pravda, Nepravda (true, false)**.
V proměnné tohoto datového typu je tedy uloženo buď „true“ (Pravda, Platí), nebo „false“ (Nepravda, Neplatí).
Výsledek s tímto datovým typem vrací např. operátory porovnání: <, >, =<, =>, ===, !== .
Boolean se využije mimo jiné všude, kde se pracuje s podmínkami - if 11, while 17.
Jeho zkratka je „bool“.

```php
<?php
$mam_brigadu = true;
var_dump($mam_brigadu);

$vysledek = 20 < 10;
var_dump($vysledek);
?>
```

Zdrojový kód 5: Boolean

řádek 2: Uložení boolean přímo.

řádek 2: Boolean jako výsledek porovnání dvou čísel.

Operace s bool S boolean hodnotami můžeme provádět:

or Logické sčítání - aby byl výsledek true, stačí aby byla jedna z hodnot true

and Logické násobení - aby byl výsledek true, všechny hodnoty musí být true

not Logická inverze - převrací hodnotu

2.2.4 String - str

String označuje **Řetězec znaků - nápis**. Zapisujeme je do uvozovek - jednoduchých, nebo dvojitých - výběr mezi jednoduchými a dvojitými nám umožní zapsat uvozovku i jako součást řetězce. **Pozor** při práci s čísly - je rozdíl mezi 15 (číslo - int) a "15"(řetězec - str). Číslo označuje počet - dá se sčitat, dělit, násobit s ostatními čísly tak , jak jste zvyklí z normálního počítání. Oproti tomu Řetězec je „obrázek“ (jak vypadá písmenko, číslo atp.) a tedy operace, které znáte (+,*) nebudou dělat to, co znáte z počítání (co to znamená sečíst dva obrázky?). PHP umí automaticky převést string na číslo, pokud je třeba - ale nedá nám nijak vědět, že to dělá.
PHP pracuje s datovými typy velmi volně je tedy třeba dát si pozor na to, co v kódu provádíme. Jeho zkratka je „str“.

```php
<?php
$jmeno = "František";
$prijmeni = 'Dobrota';
var_dump($jmeno);

$cele_jmeno = $jmeno . " " . $prijmeni
var_dump($cele_jmeno);
?>
```

Zdrojový kód 6: Float

řádek 2: Uložení string pomocí uvozovek.

řádek 6: Spojení tří nápisů do jednoho.

Operace se str S řetězcem lze provádět nepřeberné množství operací: `https://www.w3schools.com/pHP/php_ref_string.asp`
Jistě musíte znát:

. „tečka“ Zřetězení - spojí dva nápisy za sebe.

explode(odelovac, $promenna) Rozdělení - roztrhá nápis na části v místech daných odelovacem.

implode(spojovac, $promenna) Spojení - spojí všechny části v $promenna (všechny musí být string) do jednoho. Mezi každé dva spojí „spojovac“ - musí být string.

Pokud budete chtít s řetězcem cokoliv udělat - vždy se nejprve podívejte, zda již taková funkce neexistuje - téměř jistě ano a vy si ušetříte spoustu práce.

2.3 Pole - Array

Obdobně jako v jiných jazycích, máme i v PHP pole. Pole slouží k uložení více hodnot pod jedním názvem (pod jednou proměnnou).
Protože máme více hodnot pojmenované stejným názvem (proměnnou), pořtebujeme nějaký způsob, jak určit, kterou hodnotu z těchto mnoha uložených chceme používat.
Jednotlivým uloženým hodnotám říkáme prvky pole.

2.3.1 Indexované pole

V tomto typu pole máme prvky seřazené za sebou a očíslované (indexované). K jednotlivým prvkům pak přistupujeme právě pomocí indexu.
Očíslování (indexy) začínají od 0.

```
1  <html>
2  <body>
3  <?php
4  $kamaradi = array("Adam", "Bořek", "Cyril");
5  echo "Moji kamarádi jsou:<br>";
6  echo $kamaradi[0] . ", " . $kamaradi[1] . " a " . $kamaradi[2] . ".";
7  ?>
8  </body>
9  </html>
```

řádek 4: Vytvoření indexovaného pole

řádek 6: K jednotlivým prvků přistupujeme pomocí indexu v hranatých závorkách.

Zdrojový kód 7: Indexované pole

Obvykle máme v poli uloženo mnoho prvků a předem neznáme jejich počet. Proto s polem praujeme pomocí for cyklu.

```
1  <html>
2  <body>
3  <?php
4  $kamaradi = array("Adam", "Bořek", "Cyril");
5  echo "Moji kamarádi jsou:<br>";
6  $pocet_kamaradu = count($kamaradi)
7  for($i = 0; $i < $pocet_kamaradu; $i++ ){
8          echo $kamaradi[$i] . " ";
9  }
10 ?>
11 </body>
12 </html>
```

řádek 6: Uložíme si počet prvků v poli

řádek 7: Pomocí for cyklu projdeme všechna čísla od 0 do posledního indexu a použijeme je pro přístup k prvkům pole.

Zdrojový kód 8: Indexované pole - for

2.3.2 Asociativní pole

V tomto typu pole máme v prvku vždy dvě „krabičky“ - jedné říkáme key (klíč) a druhé value (hodnota). Uloženou hodnotu v tomto poli pak získáváme pomocí klíče.

```
1  <html>
2  <body>
3  <?php
4  $vek_kamaradi = array("Adam"=>16, "Bořek"=>18, "Cyril"=>17);
5  echo "Moji kamarádi jsou staří:<br>";
6  echo $vek_kamaradi["Adam"] . ", " . $vek_kamaradi["Bořek"] . " a " . $vek_kamaradi["Cyril"];
7  ?>
8  </body>
9  </html>
```

řádek 4: Vytvoření asociativního pole - ke každému key (zde jména) přiřazujeme value (zde čísla).

řádek 6: K jednotlivým prvků přistupujeme pomocí klíče.

Zdrojový kód 9: Asociativní pole

I v asociativním poli je vhodné procházet prvky pomocí for cyklu. Ten ale vypadá trochu jinak a říkáme mu „foreach“.

```
1  <html>
2  <body>
3  <?php
4  $vek_kamaradi = array("Adam"=>16, "Bořek"=>18, "Cyril"=>17);
5  echo "Moji kamarádi jsou staří:<br>";
6  foreach($vek_kamaradi as $jmeno => $vek){
7          echo $jmeno . ": " . $vek . "<br>";
8  }
9  ?>
10 </body>
11 </html>
```

řádek 6: Při procházení všech prvků řekneme, které pole chceme procházet (zde $vek_kamaradi) a uložíme si vždy key i value (zde do proměnných $jmeno a $vek)

Zdrojový kód 10: Asociatovní pole - foreach

3 Řídící struktury

Každý program běží postupně po řádcích od 1. řádku dále a vykonává příkazy přesně v tom pořadí, jak jdou za sebou.
Někdy (velmi často) ale chceme, aby se některé řádky (přikazy) přeskočily - neprovedly se. Někdy také chceme, aby se některé přikazy provedly opakovaně - vícekrát.
K tomuto slouží takzvané řídící struktury - speciální příkazy, které řídí, který řádek (příkaz) se provede jako další. Říkáme, že řídí běh programu.
To, že je nějaký řádek uvnitř řídící struktury, takzvaně v těle (např. je to řádek, který chceme přeskočit) určujeme pomocí složených závorek.

3.1 Odkazy

Operátory
`https://www.w3schools.com/pHP/php_operators.asp`
Podmínka - Větvení
`https://www.w3schools.com/pHP/php_if_else.asp`
Cykly
`https://www.w3schools.com/pHP/php_looping.asp`

3.2 Podmínka - Větvení

Můžeme si představit, že program běží po jednotlivých větvých - proto tomu říkáme větvení.
Stejně jako u větví na stromě, se v nějakém místě programu můžeme vydat dvěma směry (po dvou různých větvích). Na rozdíl od větví na stromě se ale jednotlivé větve v programu po ukončení této struktury opět spojí do jedné.

3.2.1 if

Nejjednodušší podmínka „if“ nám poslouží v případě, že chceme některé příkazy provést jen někdy - jen pokud je splněna daná podmínka. Pokud podmínka splněna není, přikazy se přeskočí.

```
1 <?php
2 if($uzivatel === "Antonín"){
3         echo "Nazdar, Tondo,";
4 }
5 ?>
```

Zdrojový kód 11: if

řádek 2: Řádek uvnitř těla (mezi složenými závorkami) se porvede jen pokud je podmínka splněna (true).

Za klíčové slovo „if“ píšeme podmínku - cokoliv, o čem umí php rozhodnout, zda je to splněné, nebo ne - zda je to *true*, nebo *false*.
Můžeme zde přímo zapsat *true*, nebo *false*, často zde píšeme porovnání: $<$, $>$, $===$, $!==$, $<=$, $>=$.

```
1 <?php
2 if($kredit <== 0){
3         echo "Nemáš dost kreditu";
4 }
5 ?>
```

Zdrojový kód 12: if - porovnání

řádek 2: Porovnání (operátor „<==“) vrací true nebo false.

3.2.2 if-else

Často chceme aby se při splnění podmínky vykonaly některé příkazy a při nesplnění podmínky se vykonaly jiné. Chceme tedy pomocí podmínky vybrat jednu ze dvou skupin příkazů (řádků).
K tomu slouží konstrukce „if-else“. Při splnění podmínky (true) se (stejně jako v obyčejném „if“) provedou příkazy v části po „if“. Při nesplnění podmínky (false) se provedou příkazy v části po „else“. Jedna z těchto dvou skupin příkazů se tedy provede vždy.

```
1 <?php
2 if($skore_1 < $skore_2){
3         echo "Vyhrál hráč 2";
4 }else{
5         echo "Vyhrál hráč 1";
6 }
7 ?>
```

Zdrojový kód 13: if - else

řádek 3: Pokud je splněna podmínka, provede se tento blok

řádek 5: Pokud podmínka není spléna, provede se tento blok

Kontrolní otázka: Kdo vyhrál, pokud mají stejné skóre?

3.2.3 elseif

Opravíme kód 14. Někdy potřebujeme vybrat ne mezi dvěma příkazy, ale mezi více. K tomu slouží konstrukce „elseif“ - dovolí nám vložit další „větev“. Takových větví může být libovolné množství a **program se vydá** vždy jen tou z nich, u které je **podmínka splněna jako první** (kontrolováno od **shora dolů**). Pokud není splněna žádná z podmínek, vydá se program větví „else“.

řádek 3, 5, 7: Provede se jen jeden z těchto bloků.

```
1 <?php
2 if($skore_1 < $skore_2){
3         echo "Vyhrál hráč 2";
4 }elseif ($skore_1 > $skore_2){
5         echo "Vyhrál hráč 1";
6 }else{
7         echo "Remíza";
8 }
9 ?>
```

Zdrojový kód 14: if - else

3.2.4 Switch

Pokud chceme program rozvětvit do více větví. Můžeme také použít „switch“.
Můžeme ho použít, pokud se chceme konkrétní větví vydat ve chvíli, kdy se zadaná proměnná přesně rovná určené hodnotě (nelze tedy použít pro větší, menší).

```php
<?php
$color = "red";

switch ($color) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

řádek 4: Určíme, podle hodnoty které proměnné se má program větvit.

řádek 5: Jednotlivé hodnoty, pro které chceme vytvořit speciální větev zapíšeme do „case“

řádek 7: Pozor! Každou větem musíme ukončit příkazem „break;“

řádek 14: Můžeme vytvořit i deafaultní vetěv - pokud hodnota nebude odpovídat ani jedné z těch, které jsme připravili

Zdrojový kód 15: Switch

3.2.5 Vnoření

Podmínky (stejně jako jiné řídící struktury) můžeme takzvaně **vnořovat** - tedy **vkládat jednu do druhé**.

```php
<?php
if($jazyk === "en"){
        if($uzivatel = "Antonín"){
                echo "Hello, Tonda";
        }
}else{
        if($uzivatel === "Antonín"){
                echo "Ahoj, Tondo";
        }
}
?>
```

řádek 3: Tento blok se vyhodnotí jen v případě, že je jazyk nastaven na „en“.

řádek 8: Pokud jazyk není nastaven na „en“ a také je uzivatel „Antonín“, provede se tento řádek.

Zdrojový kód 16: Vnoření

3.3 Cykly

Velmi často chceme, aby program provedl výpočet vícekrát. Buď proto, že tímto opakováním získáme požadovaný výsledek, nebo chceme stejnou operaci provést s více „objekty“ (např. zobrazit všechny produkty z obchodu).
K tomuto opakování slouží takzvané cykly.

3.3.1 While

Cyklus „while“ opakuje příkazy, **dokud je splněna** zadaná **podmínka**. Nejprve zkontroluje, zda je podmínka splněna - pokud ano, **vykoná všechny zadané příkazy** a poté zkontroluje podmínku znovu - a tak stále dokola, dokud při kontrole podmínky nezjistí, že podmínka již splněna není. Ve chvíli, kdy podmínka splněná není (ať už hned napoprvé, nebo kdykoliv později), přeskočí všechny zadané příkazy a program pokračuje dále za tímto „while“ cyklem.

```php
<?php
$x = 0;

while($x <= 100) {
    echo "Už jsme napočítali do: $x <br>";
    $x+=10;
}
?>
```

Zdrojový kód 17: While

řádek 4: Tělo while-cyklu (příkazy ve složených závorkách) se bude provádět stále dokola, dokud pude tato podmínka spněna
Podmínka se kontroluje hned na začátku a poté pokaždé po prvedení všech příkazů v těle.

řádek 6: U cyklů je velmi důležité zajistit, aby někdy skončili. Tedy, aby někdy podmínka slněna nebyla.

3.3.2 For

For cyklus je v zásadě stejný jako předchozí while. Pouze má přehlednější zápis (a proto se také v takových chvílích používá), pokud máme danný počet průběhů, které má for-cyklus vykonat.
Hlavička (první řádek) for-cyklu se skládá ze tří „přihrádek“ oddělených středníkem.

1. přihrádka se provede pouze jednou a nejčastěji zde zadáváme počáteční hodnotu počítadla kol
2. přihrádka je stejná, jako ve while-cyklu - je zde zapsaná podmínka, která určuje, kdy se cyklus zastaví - zastaví se poté, co podmínka není splněna
   Podmínka se kontroluje na začátku (tedy hned po provedení první přihrádky) a poté pokaždé po provedení všech příkazů v těla a ve třetí přihrádce
3. přihrádke se provede vždy po vykonání všech příkazů v těle for-cyklu

Ukázkový kód z příkladu 17 můžeme přepsat pomocí for následovně:

```php
<?php
for($x = 0; $x <= 100; $x++) {
    echo "Už jsme napočítali do: $x <br>";
}
?>
```

Zdrojový kód 18: For

řádek 2: První přihrádka $x=0 se provede pouze jednou, hned při doběhnutí programu na tento řádek.
Druhá přihrádka $x<=100 se kontroluje vždy před vstupem do těla cyklu (a do těla cyklu progrma vstoupí pouze pokud je podmínka splněna)
Třetí přihrádka $x++ se provede vždy po dokončení všech příkazů v těle cyklu.

3.3.3 Foreach

Cyklus „foreach“ používáme v php k procházení pole. Při procházení pole pomocí „foreach“ máme uložen **jeden** (u asociativního pole klíč a hodnotu, u indexovaného pole jen hodnotu) prvek z pole při každém průchodu.
Při prvním průchodu tělem cyklu (příkazy vesložených závorkách) máme uložen první prvek, při druhém průchodu druhý prvek atd.
Můžeme tak nějakou operaci (jeden nebo více příkazů) provést s každým prvkem pole zvlášť - a provést ji postupně se všemi prvky.

```php
<?php
$barvy = array("red", "green", "blue", "yellow");

foreach ($barvy as $jedna_barva) {
  echo "Moje oblíbená barva je: $jedna_barva <br>";
}

$barvy = array("red"=>80, "green"=>60, "blue"=>90, "yellow"=>70);

foreach ($barvy as $jedna_barva=>$procent) {
  echo "Moje oblíbená barva je: $jedna_barva a mám ji rád na $procent procent<br>";
}
?>
```

Zdrojový kód 19: Foreach

řádek 4: V proměnné „$jedna_barva“ jsou postupně (po jedné) uloženy všechny barvy z pole „$barvy“

řádek 10: U asociatvního pole si můžeme uložit zvlášť klíč a zvlášť hodnotu.

3.3.4 Break

Jakýkoliv cyklus (případně switch) můžeme také ukončit kdykoliv v jeho těle (mezi příkazy, které jsou v něm napsané). Slouží k tomu příkaz **„break“**. Ve chvíli kdy **program** dorazí na řádek s tímto příkazem, **okamžitě skočí za cyklus**, ve kterém je tento příkaz zapsaný.

```php
<?php
$barvy = array("red", "green", "blue", "yellow");

foreach ($barvy as $jedna_barva) {
  if($jedna_barva === "blue"){
    break;
  }
  echo "Moje oblíbená barva je: $jedna_barva <br>";
}
?>
```

Zdrojový kód 20: Break

řádek 6: Ve chvíli, kdy dorazí program na tento řádek (tedy musí být spněna podmínka v if), okamžitě se přeruší průběh foreach-cyklu a program pokračuje za jeho tělem (za zavírací složenou závorkou)

3.3.5 Continue

Pomocí předchozího „break“ se ukončil celý průběh cyklu. Můžeme také ukončit průběh pouze jednoho „kolečka“ (aktuálního průchod). Cyklus pak bude pokračovat v dalším kolečku - od jeho prvního řádku.

```php
<?php
$barvy = array("red", "green", "blue", "yellow");

foreach ($barvy as $jedna_barva) {
  if($jedna_barva === "blue"){
    continue;
  }
  echo "Moje oblíbená barva je: $jedna_barva <br>";
}
?>
```

Zdrojový kód 21: Continue

řádek 6: Ve chvíli, kdy dorazí program na tento řádek (tedy musí být splněna podmínka v if), přeruší se průběh jednoho kolečka foreach-cyklu a program pokračuje dále ve foreach-cyklu, ale s další hodnotou v proměnné „$jedna_barva“

4 Formulář

Pomocí PHP můžeme zpracovat data odeslaná přes formulář. Samotný formulář je vytvořený pomocí HTML.
Pro nastavení souboru, který bude zpracovávat data po jejich odeslání, nastavíme u formuláře atribut „action“. Soubor, jehož název zapíšeme do attributu „action“ se spustí po klikutí na tlačítko ve formuláři - a bude mít k dispozici data zadaná do formuláře.
Můžeme spustit jakýkoliv soubor (uložený na serveru) - klidně znovu ten, ve kterém máme náš formulář.

```
1  <html>
2  <body>
3
4  <form action="soubor_pro_zpracovani.php" method="post">
5  Počet řádků: <input type="number" name="radky"><br>
6  Počet sloupců: <input type="number" name="sloupce"><br>
7  <input type="submit" value="Zobraz tabulku">
8  </form>
9
10 </body>
11 </html>
```

Zdrojový kód 22: Formulář v HTML

řádek 4: Atribut „action“ určuje, který soubor se spustí po odeslání dat
Atribut „method“ určuje, jakým způsobem se data odešlou

řádek 5,6 Atribut „name“ musíme vyplnit, abychom mohli získat data z tohoto konkrétního políčka - určeného právě jeho „name“.

4.1 Metoda GET

Pokud pro odesílání dat zvolíme metodu „GET“, budou přenášená data snadno čitelná pro kohokoliv. Jsou přenášena pomocí URL (můžete si například vyzkoušet vyhledat něco pomocí vašeho oblíbeného webového vyhledávače - po vyhledání se podívejte na adresní řádek vašeho prohlížeče.)
Metoda „GET“ tedy není vhodná pro přenášení tajných informací, jako jsou hesla apd.
Po odeslání dat pomocí metody „GET“ budou data uložena v asociativním poli „$_GET“. Toto pole je globální proměnná a můžeme jej tedy používat kdekoliv v kódu.
K informacím zadaným do formuláře se dostaneme pomocí „$_GET["nazev_policka_ve_formulari"]“.
„nazev_policka_ve_formulari“ nastavíme políčku pomocí atributu „name“.
Použití této metody nastavíme v atributu formuláře „method“.

```
1  <html>
2  <body>
3
4  <form action="soubor_pro_zpracovani.php" method="get">
5  Počet řádků: <input type="number" name="radky"><br>
6  Počet sloupců: <input type="number" name="sloupce"><br>
7  <input type="submit" value="Zobraz tabulku">
8  </form>
9
10 <?php
11 $zadane_radky = $_GET["radky"];
12 $zadane_sloupce = $_GET["sloupce"];
13 ?>
14
15 </body>
16 </html>
```

řádek 4: Atribut „method“ nastavený na "get"určuje použití metody „GET“.

řádek 11, 12: Po odeslání dat je můžeme použít pomocí $_GET[] a „name“ daného políčka.

Zdrojový kód 23: Metoda GET

4.2 Metoda POST

U metody „POST“ se data přenáší pomocí HTTP POST a nejsou tedy snadno čitelná pro kohokoliv. Navíc také můžeme poslat libovolný objem dat, což u metody „GET“ nelze (nicméně zde ve škole nás to příliš netrápí).
Po odeslání dat pomocí metody „POST“ budou data uložena v asociativním poli „$_POST“. Toto pole je globální proměnná a můžeme jej tedy používat kdekoliv v kódu.
K informacím zadaným do formuláře se dostaneme pomocí „$_POST["nazev_policka_ve_formulari"]“. „nazev_policka_ve_formulari“ nastavíme políčku pomocí atributu „name“.

```
1  <html>
2  <body>
3
4  <form action="soubor_pro_zpracovani.php" method="post">
5  Počet řádků: <input type="number" name="radky"><br>
6  Počet sloupců: <input type="number" name="sloupce"><br>
7  <input type="submit" value="Zobraz tabulku">
8  </form>
9
10 <?php
11 $zadane_radky = $_POST["radky"];
12 $zadane_sloupce = $_POST["sloupce"];
13 ?>
14
15 </body>
16 </html>
```

řádek 4: Atribut „method“ nastavený na "post"určuje použití metody „POST“.

řádek 11, 12: Po odesání dat je můžeme použít pomocí $_POST[] a „name“ daného políčka.

Zdrojový kód 24: Metoda POST

5 Funkce

Funkce (někdy také nazývané metody) slouží k významnému zjednodušení a zpřehlednění kódu. Funkci si můžete představit jako „krabičku“, která umí dělat něco užitečného a kterou mám uloženou v paměti. Tuto „krabičku“ můžu v programu použít kolikrát chci.
Často chceme stejný proces (několik příkazů) spustit na několika místech v programu. Funkce nám umožní tento proces (několik příkazů) zapsat pouze jednou - pojmenovat ho - a poté ho spustit jen zadáním jeho jména. Nemusíme tak stále dokola psát stejný kód na všech místech, kde ho chceme spustit.
Pokud někde v programu píšete **podruhé** stejnou část kódu, již je to chvíle, kdy je čas na **použití funkce**.

Pro ukázku použití funkcí vytvoříme funkce, které budou zdravit naše kamarády a učitele

5.1 Vytvoření funkce

Předtím, než můžeme funkci používat, ji musíme samozřejmě vytvořit - definovat. Definicí funkce ji pouze uložíme do paměti. V paměti funkce čeká do té doby, dokud ji nezavoláme (nespustíme). Funkci vytvoříme zapsáním klíčového slova „**function**“. Za ním následuje **název funkce** - jak chceme funkci volat. Dále jsou **kulaté závorky** - později do závorek zapíšeme takzvané argumenty, ale i když žádné argumenty psát nechceme, kulaté závorky zde být musí. Tomuto prvnímu řádku se říká **hlavička funkce**. Následuje tělo funkce, které je zapsáno do složených závorek. Do složených závorek zapíšeme všechny příkazy, které chceme mít uvnitř funkce - tedy ty, které se provedou, až funkci zavoláme (spustíme). Těmto řádkům říkáme „tělo funkce“.

```
1  <?php
2  function pozdrav(){
3          echo("Nazdar");
4          echo("Nazdar");
5          echo("Nazdar");
6  }
7
8  echo("Franta");
9  pozdrav();
10
11 echo("Lojza");
12 pozdrav();
13
14 echo("Mařenka");
15 pozdrav();
16 ?>
```

Zdrojový kód 25: Definice funkce

řádek 2: Definice (vytvoření, připravení) funkce: Klíčové slovo „function“, název funkce, kulaté závorky
Dále začne tělo funkce otevírací složenou závorkou

řádky 3-5: Tyto řádky (příkazy) se provedou po zavolání funkce - jsou uvnitř funkce - jsou odsazené. Říkáme jim „tělo funkce“
Funkce končí zavírací složenou závorkou

řádek 8: Program pokračuje na dalším řádku - již nepatří do funkce.

řádky 9, 12, 15: Voláme funkci. Na všech těchto řádcích skočí program do volané funkce (tedy na řádek 2) a provede všechny příkazy uvnitř funkce.

5.2 Argumenty

U funkce sice chceme, aby prováděla stále stejné příkazy, ale byly bychom rádi, aby uměla tyto stejné příkazy provést na různých datech (vstupech). Například pozdravit společně se jménem - a toto jméno bude pokaždé jiné (podle toho, koho zrovna zadravíme).
Abychom mohli dostat nějakou informaci (třeba jméno) dovnitř do funkce, použijeme argumenty

funkce.

```
1  <?php
2  function pozdrav($jmeno){
3          echo("Nazdar $jmeno");
4          echo("Nazdar $jmeno");
5          echo("Nazdar $jmeno");
6  }
7
8
9  pozdrav("Franta");
10
11 pozdrav("Lojza");
12
13 pozdrav("Mařenka");
14 ?>
```

Zdrojový kód 26: Funkce s argumentem

řádek 2: Definice funkce s tím, že jí při volání předáme jeden argument - jméno, které má pozdravit: Argumenty (zde je pouze jeden, ale může jich být více) píšeme do kulatých závorek.

řádek 9, 11, 13: Protože jsem vytvořili funkci s argumentem - musíme jí nějakou hodnotu tohoto argumentu předat.

Argumentů můžeme funkci předat více, mohou být jakýchkoliv datových typů a mohou se ve funkci použít libovolně - tedy může být jeden argument string, druhý integer, atd. Více argumentů píšeme do kulatých závorek a **odělujeme je čárkou**.

Upravíme funkci tak, abychom jí mohli říct (předat argument), kolikrát má daného člověka pozdravit

```
1  <?php
2  function pozdrav($jmeno, $pocet){
3          for($i = 0; $i < $pocet; $i++){
4                  echo("Po $i: Nazdar $jmeno<br>");
5          }
6  }
7
8
9  pozdrav("Franta", 1);
10 pozdrav("Lojza", 3);
11 pozdrav("Marenka", 10);
```

Zdrojový kód 27: Funkce s dvěma argumenty

řádek 2: Definice funkce s dvěma argumenty.

řádek 9, 10, 11: Protože jsme vytvořili funkci s dvěma argumenty - musíme jí při volání předat dvě hodnoty.

5.3 Návratová hodnota - return

Funkce mohou také spočítat výsledek (tak, jak znáte z matematiky - např. funkce 2x+1 spočítá pro vstup 1 výsledek 3, pro vstup 4 výsledek 9, pro vstup 7 výsledek 15 atd.).
Často u funkce chceme, aby nám výsledek, který spočítá, takzvaně „vrátila“ na místo (řádek) programu, odkud jsme funkci zavolali. V tomto místě (kde jsme funkci zavolali) typicky výsledek použijeme k dalším výpočtům, ale můžem s ním dělat cokoli chceme (nic, uložit ho, dále s ním počítat).
Že je již výpočet u konce (došli jsme k výsledku, který chceme vrátit) a chceme tedy funkci ukončit a vrátit výsledek, zepíšeme ve funkci pomocí klíčového slova „return“. Po tomto klíčovém slově se již ve funkci neprovedou žádné příkazy. Pokud za slovo „return“ zapíšeme co má funkce vrátit,

přenese se tato hodnota do místa, odkud jsme funkci zavolali.

```
1  <?php
2  function pozdrav($jmeno, $pocet){          !
3          for($i = 0; $i < $pocet; $i++){
4                  echo("Po $i: Nazdar $jmeno<br>");
5          }
6
7          return strlen(jmeno);
8          echo("Toto se jiz nevypise :-( ");
9  }
10
11
12 $delka_jmena_1 = pozdrav("Jan", 1);
13 echo("$delka_jmena_1");
14
15 echo(pozdrav("Vladislav", 3));
16
17 $delka_jmena_2 = pozdrav("Cecilie", 10);
18
19 $soucet_delky_jmen = $delka_jmena_1 + $delka_jmena_2;
20 echo("Pocet znaku na pozvance: $soucet_delky_jmen")
21 ?>
```

Zdrojový kód 28: Funkce s return

- řádek 7: Klíčové slovo „return“ a za ním hodnota, která se má vrátit - zde např. délka jména, které jsme funkci předali jako argument (např. u "Jan"je vráceno 3, u "Cecilie"je vráceno 7). Žádný další řádek už se neprovede.
- řádek 12: Vrácenou hodnotu si můžeme uložit do proměnné (zde „$delka_jmena_1“).
- řádek 15: Vrácenou hodnotu můžeme také přímo použít (zde zobrazit na stránce).
- řádek 19: Uložené hodnoty můžeme samozřejmě kdykoliv použít.

6 Session a cookies

Často si chceme na našich stránkách udržet libovolnou informaci i při přechodu na jinou stránku (v rámci naší webové prezentace). Například se chceme příhlásit pouze jednou a rádi bychom, aby toto přihlášení vydrželo celou dobu, co na stránkách pracujeme. Případně si nastavíme barvu pozadí a chceme mít stejné pozadí po celou dobu.
Je zřejmé, že si takovou informaci musíme někam uložit. Musíme ji uložit na místo, kde zůstane uchována i při zavření stránky - a otevření jiné stránky, nebo i té samé.
Možnosti kam uložit informace budeme využívat dvě:

Na server pomocí **session**

Do PC pomocí **cookies**

6.1 Session

Pomocí session si můžeme uložit informace na server, který spouští naše php skripty.
Nejprve navážeme se serverem trvalé spojení (session) - toto spojení je platné (server pozná, že jsme novou stránku otevřeli opět my) do doby, dokud nezavřeme prohlížeč (velmi komplikovaným způsobem jde přerušit i pomocí php kódu). Při zavření pouze jedné karty, zůstane spojení navázané.
V průběhu spojení si informace ukládáme do asociativního pole $_SESSION .

```
1  <?php
2  session_start()
3  ?>
4  <html>
5  <body>
6
7  <?php
8  $_SESSION["username"] = "MojeJmeno";
9  $_SESSION["font_size"] = 50;
10
11 echo '<p style="font-size: $_SESSION[\"font_size\"];">';
12 echo 'Přihlášen uživatel: $_SESSION["username"] </p>';
13
14 unset($_SESSION['username']);
15 session_destroy();
16 ?>
17
18 </body>
19 </html>
```

řádek 2: Navázání spojení (session) se serverem.
Musí být první příkaz na stránce.

řádek 8: Uložení informací.

řádek 11: Použití informací.

řádek 14: Smazání jedné proměnné.

řádek 15: Smazání všech dat.

Zdrojový kód 29: Session

6.2 Cookies

Cookies jsou informace, které si stránka ukládá do PC učivatele.
Není zcela samozřejmé, že si stránka (která především slouží ke čtení) ukládá cokoliv na uživatelské zařízení - a zasahuje tak do něj (zabírá paměť). Proto s tímto zásahem do svého zařízení musí uživatel souhlasit - to je důvod, proč jste se již jistě setkali s tím, že jste na stránce museli odškrtnout souhlas s cookies.

Informace **ukládáme a měníme** pomocí funkce „**setcookie()**“.
Čteme je z asociativního **pole $_COOKIE**.

```php
<?php
setcookie("username", "MojeJmeno");
setcookie("font-size", 50);

/*Tento řádek zkuste zakomentovat*/
setcookie("username", "", time() - 3600);
?>
<html>
<body>

<?php
echo '<p style="font-size: $_COOKIE[\"font_size\"];">';
echo 'Přihlášen uživatel: $_COOKIE["username"] </p>';
?>

</body>
</html>
```

Zdrojový kód 30: Cookie

řádky 2, 3: Vytvoření (uložení) cookie. Funkce „setcookie()“ se musí objevit před <html> tagem

řádek 12: Použití informací.

řádek 6: Smazání jednoho z cookie. Nastavíme mu dobu platnosti na čas (v sekundách) v minulosti.
Zde jen ilustrativní přiklad - obvykle by tento řádek byl na jiné stránce.

7 Databáze

Zdrojové kódy převzaty z `https://www.w3schools.com/php/php_mysql_intro.asp`, kde najdete i další ukázky.
Pokud chceme, aby naše webové stránky zpracovávaly větší množství informací (např. více uživatelů, různé zboží v e-shopu) je vhodné tyto informace uložit v databázi.
Zde ve škole můžeme využít školní databázi na serveru `dbs.spskladno.cz` (MySQL)- pro přístup přes prohlížeč použijte `http://dbs.spskladno.cz/myadmin/` . Přístupové údaje získáte od svých vyučujících.
Možností jak pracovat s databází pomocí php je více a mimo jiné záleží na typu databáze. Zde si ukážeme jednu z možností.

7.1 Přístupové údaje

Pro připojení k databázi (přesněji k databázovému serveru) je potřeba zadat údaje:

adresa serveru ip, nebo url adresa - dbs.spskladno.cz

jméno získáte od vyučujících

heslo získáte od vyučujících

pro připojení ke konkrétní databázi navíc zadáme název databáze:

název databáze zde na školním databázovém serveru závisí na účtu, pod kterým se k serveru přihlásíme

Pomocí výše uvedených údajů vytvoříme spojení našeho php skriptu s databází. Přes toto spojení můžeme zasílat do databáze požadavky pomocí jazyku SQL.

```
1  <?php
2  $servername = "dbs.spskladno.cz";
3  $username = "dotaz_na_vyucujiciho";
4  $password = "dotaz_na_vyucujiciho";
5  $dbname = "vyuka__";
6
7  $conn = new mysqli($servername, $username, $password, $dbname);
8
9  if ($conn->connect_error) {
10     exit("Spojení se nezdařilo. Chyba: " . $conn->connect_error);
11 }
12
13 $conn->close();
14 ?>
```

řádek 2: Uložení přístupových údajů.
řádek 7: Vytvoření spojení
řádek 9: Kontrola, zda se spojení podařilo. Pokud ne, ukončí se skript a vypíše se chybová hláška.
řádek 13: Ukončení spojení

Zdrojový kód 31: Připojení k databázi

7.2 Dotazy bez výsledku

U některých dotazů do databáze nezískáváme žádný výsledek (např. vytvoření tabulky, vložení dat do tabulky). U takových dotazů můžeme zjistit, zda dotaz proběhl v pořádku, případně se něco pokazilo (např. nemáme přístup, špatně zapsaný dotaz) a můžeme vypsat chybovou hlášku, kterou nám zaslal zpět databázový server.

```php
<?php
$servername = "dbs.spskladno.cz";
$username = "dotaz_na_vyucujiciho";
$password = "dotaz_na_vyucujiciho";
$dbname = "vyuka__";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    exit("Spojení se nezdařilo. Chyba: " . $conn->connect_error);
}

$sql_create = "CREATE TABLE Uzivatel (
id INT PRIMARY KEY,
jmeno TEXT NOT NULL,
heslo TEXT NOT NULL
)";

$sql_insert = "INSERT INTO Uzivatel (jmeno, heslo) VALUES ('jarda', '123'), ('adam', '456')";

$result = $conn->query($sql_create);
if ($result === TRUE) {
    echo "Tabulka vytvořena.";
} else {
    echo "Tabulku se nepodařilo vytvořit. Chyba: " . $conn->error;
}

$result = $conn->query($sql_insert);
if ($result === TRUE) {
    echo "Data vložena.";
} else {
    echo "Data se nepodařilo vložit. Chyba: " . $conn->error;
}

$conn->close();
?>
```

řádek 17, 19: Příprava dotazů pro vytvoření tabulky a vložení dvou uživatelů.
Ve skutečné databázi se hesla nikdy neukládají v této „otevřené“ formě.

řádek 21, 28: Zaslání dotazu do databáze.

řádek 22, 29: Kontrola, zda dotazy proběhly v pořádku.
Pokud ne, ukončí se skript a vypíše se chybová hláška.

Zdrojový kód 32: Tvorba tabulky a vložení dat

7.3 Dotazy s výsledkem

U některých dotazů získáváme z databáze informaci (proto databázi máme . . .).
Výsledkem dotazu může být velmi mnoho řádků (např. e-shop má v nabídce mnoho triček) a bylo by značně náročné, přenášet celý výsledek najednou.
V následující ukázce si můžeme představit, že po odeslání dotazu se v databázi **dočasně vytvoří** náš **výsledek** a do našeho php skriptu **se přenese ukazatel** na tyto data (nepřenáší se tedy data samotná, ale pouze informace, kde v databázi jsou). **Část dat** (typicky jeden řádek) **se přenese** až **po zavolání funkce** „fetch_assoc()“. Díky tomu můžeme z databáze stahovat jen tolik dat, kolik potřebujeme.
Data jsou po přenesení **uložena v asociativním poli**, kde **název** jednotlivých **položek** (klíč) je **název sloupečku** v databázi.

```php
<?php
$servername = "dbs.spskladno.cz";
$username = "dotaz_na_vyucujiciho";
$password = "dotaz_na_vyucujiciho";
$dbname = "vyuka__";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    exit("Spojení se nezdařilo. Chyba: " . $conn->connect_error);
}

$sql_create = "CREATE TABLE Uzivatel (
id INT PRIMARY KEY,
jmeno TEXT NOT NULL,
heslo TEXT NOT NULL
)";

$sql_insert = "INSERT INTO Uzivatel (jmeno, heslo) VALUES ('jarda', '123'), ('adam', '456')";

$result = $conn->query($sql_create);
if ($result === TRUE) {
    echo "Tabulka vytvořena.";
} else {
    echo "Tabulku se nepodařilo vytvořit. Chyba: " . $conn->error;
}

$result = $conn->query($sql_insert);
if ($result === TRUE) {
    echo "Data vložena.";
} else {
    echo "Data se nepodařilo vložit. Chyba: " . $conn->error;
}

$sql_select = "SELECT id, jmeno, heslo FROM Uzivatel";
$result = $conn->query($sql_select);

if ($result->num_rows > 0) {

  while($row = $result->fetch_assoc()) {
    echo "id: ". $row["id"]. " - Jméno: ". $row["jmeno"]. " - Heslo: ". $row["heslo"]. "<br>";
    }

} else {
    echo "Ve výsledku nejsou žádné řádky.";
}

$conn->close();
?>
```

řádek 35: Příprava dotazu pro získání dat.

řádek 38: Zjištění počtu řádků ve výsledku.

řádek 40: Stažení jednoho řádku z výsledku a kontrola, zda jsou v něm data (po stažení všech řádků již budou stažená data prázdná)
Toto se opakuje, dokud není stažený řádek prázdný (již ve výsledku žádný není).

řádek 41: Ze staženého řádku čteme data jako z asociativního pole.

Zdrojový kód 33: Select - dotaz s výsledkem

8 Užitečné

8.1 Proměnná $_SERVER

Proměnná „$_SERVER“ je globální proměnná (tedy k ní máme přístup kdekoliv v programu), ze které můžeme získat spoustu užitečných informací: `https://www.php.net/manual/en/reserved.variables.server.php`.

8.1.1 ’REQUEST_METHOD’

V „$_SERVER[’REQUEST_METHOD’]“ lze zjistit, pomocí jaké metody jsme načetli stránku. Pro nás to bude obzlvášště užitečné při zpracování dat z formuláře. Můžeme zjistit, zda jsme stránku zobrazili bez odeslání dat (a tedy nemáme co zpracovávat), nebo po odeslání dat přes formulář (a tedy můžeme tyto data používat).

```
1  <html>
2  <body>
3
4  <form action="soubor_pro_zpracovani.php" method="post">
5  Počet řádků: <input type="number" name="radky"><br>
6  Počet sloupců: <input type="number" name="sloupce"><br>
7  <input type="submit" value="Zobraz tabulku">
8  </form>
9
10 <?php
11 if ($_SERVER["REQUEST_METHOD"] == "POST"){
12         $zadane_radky = $_POST["radky"];
13         $zadane_sloupce = $_POST["sloupce"];
14 }else{
15         echo "Zatím jsi nic neposlal.<br>";
16         echo "Zadej počet řádků a sloupců, klikni na tlačítko a uvidíš ty divy...";
17 }
18
19 ?>
20
21 </body>
22 </html>
```

Zdrojový kód 34: Metoda POST

- řádek 4: Nastavení metody „POST“.
- řádek 11: Kontrola, zda na stránku přišly data pomocí metody „POST“.
- řádek 12-13: Pokud byla použita metoda „POST“, můžeme data zpracovat.
- řádek 15-16: Pokud nebyla použita metoda „POST“, uděláme něco jiného.

8.2 Funkce

8.2.1 empty(), isset()

Funkce „empty($promenna)“ vrátí hodnotu „True“, pokud je proměnná, kterou testujeme (vložíme do kulatých závorek), „prázdná“. „Prázdná“ znamená, že neexistuje, nebo má hodnotu:
0; **0.0**; **"0"**; **""**; **NULL**; **FALSE**; **array()**
Obdobným způsobem pracuje funkce „isset($promenna)“. Ta naopak vrátí hodnotu „True“, pokud proměnná **existuje** a je v ní uložena hodnota **ruzná od NULL**.

```
1  <html>
2  <body>
3
4  <form action="soubor_pro_zpracovani.php" method="post">
5  Počet řádků: <input type="number" name="radky"><br>
6  Počet sloupců: <input type="number" name="sloupce"><br>
7  <input type="submit" value="Zobraz tabulku">
8  </form>
9
10 <?php
11 if (empty($_POST["radky"])){
12         echo "Nezadal jsi počet řádků!";
13 }else{
14         $zadane_radky = $_POST["radky"];
15 }
16
17 if (isset($_POST["sloupce"])){
18         $zadane_sloupce = $_POST["sloupce"];
19 }else{
20         echo "Nezadal jsi počet sloupců!";
21 }
22 ?>
23
24 </body>
25 </html>
```

Zdrojový kód 35: empty()

- řádek 11: Kontrola, zda byly zadány řádky
- řádek 12: Pokud nebyly zadány - proměnná je prázdná - vypíšeme hlášku.
- řádek 14: Pokud byly zadány - proměnná není prázdná - můžeme ji použít.
- řádek 17: Kontrola, zda byly zadány sloupce
- řádek 18: Pokud byly zadány - proměnná je nastavena - můžeme ji použít.
- řádek 20: Pokud nebyly zadány - hodnota proměnné je NULL, nebo neexistuje - vypíšeme hlášku.