# Podmínky

Činnost řady příkazů v PHP je řízena tzv. podmínkou. Podmínka je zjednodušeně řečeno zápis, který PHP vyhodnotí a zjistí, zda je pravdivý nebo ne. Podmínky se v PHP používají zejména k řízení řady příkazů, mezi něž patří i podmíněný příkaz `if`. Téma podmínek je v PHP velmi důležité, bez nich se prakticky nedá napsat ani krátký skript PHP.

# Podmíněný příkaz if

Tato podkapitola je zaměřena na podmíněný příkaz `if`, což je jedna z nejdůležitějších konstrukcí v každém programovacím jazyce vůbec. To umožňuje vykonávání části programu jen tehdy, pokud je splněna určitá podmínka.

## Úplný podmíněný příkaz if

Celá konstrukce `if` má zhruba takovýto tvar:

```
if (podmínka)
  příkaz_který_se_vykoná_pokud_je_podmínka_pravdivá;
else
  příkaz_který_se_vykoná_pokud_není_podmínka_nepravdivá;
```

Konstrukce `if` pracuje tak, že se nejdříve podívá na podmínku. Podmínka je to, co je zapsané v kulatých závorkách za klíčovým slovem `if`. PHP s každou podmínkou provádí následující operaci: vyhodnotí ji a zjistí, jestli je nebo není pravdivá.

Podmínkou se rozumí v PHP cokoli, co PHP umí vyhodnotit v měřítku pravdivost (pravda vs nepravda). V zásadě to může být jakýkoli výraz, kterému PHP rozumí. Může to být nějaká hodnota, třeba číslo nebo řetězec, může to být proměnná nebo matematická operace.

Po vyhodnocení podmínky potom konstrukce `if` vykoná jeden ze dvou příkazů. Pokud byla podmínka vyhodnocena jako pravdivá, vykoná se první příkaz, který je zapsaný za klíčovým slovem `if` (přesněji řečeno příkaz, který je zapsaný až za podmínkou). V případě, že podmínka není pravdivá, vykoná se druhý příkaz, který je zapsaný za slovem `else`.

## Příklady tvorby konstrukcí s if

Příklad tvorby konstrukcí s `if`:

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

```
    echo 'Podmínka není pravdivá';
?>
```

**Příklad:** Soubor kap04_01.php

Pokud spustíte výše uvedený příklad, dostanete vypsaný text: `Podmínka je pravdivá`. Ale asi bude zajímavější vysvětlit, co vlastně příklad dělá.

Protože se jedná vlastně o konstrukci `if`, PHP nejdříve vyhodnotí podmínku. Pokud se podíváte trochu výše na příklad, najdete tam jako zápis podmínky 10 > 2, jinak řečeno ptáme se na to, jestli je číslo 10 větší než číslo 2. Podmínku v našem příkladu PHP vyhodnotí jako pravdivou, protože je evidentní, že číslo 10 je větší než číslo 2. Protože podmínka je pravdivá, vykoná PHP první příkaz, což v tomto případě znamená, že se vypíše text: `Podmínka je pravdivá`.

Zkuste si úvodní příkaz obrátit, tzn. zkuste uvést podmínku, která by neplatila. Můžete to udělat třeba takto:

```
<?php
  if (1 > 2)
    echo 'Podmínka je pravdivá';
  else
    echo 'Podmínka není pravdivá';
?>
```

**Příklad:** Soubor kap04_02.php

Pokud si spustíte příklad, dostanete teď opačný výpis, protože se vypíše text: `Podmínka není pravdivá`.

Výše uvedený příklad zase pracuje tak, že PHP nejdříve provede vyhodnocení podmínky. Podmínka je v tomto případě zapsána jako `1 > 2`, tedy ptáme se, zda číslo 1 je větší než číslo 2. Podmínka není pravdivá, takže PHP provede druhý příkaz za slovem `else`, a to v našem příkladu znamená, že se vypíše text: `Podmínka není pravdivá`.

Pokud to shrneme, tak s konstrukcí `if` dostáváme do ruky nástroj, jenž umožní vykonávat některé příkazy jenom někdy v závislosti na určité podmínce.

## Neúplný podmíněný příkaz if

Velmi často se v praxi stává, že nepotřebujeme celou funkčnost konstrukce `if`, tak jak byla vysvětlena v podkapitole *Úplný podmíněný příkaz if*, kde to funguje tak, že při pravdivé podmínce se vykoná první příkaz a při nepravdivé podmínce druhý příkaz. V mnoha případech nám stačí, když se při pravdivosti podmínky vykoná příkaz a při nepravdivosti podmínky se nevykoná nic. V takovém případě můžeme vynechat celou část konstrukce od slova `else` a výsledkem je tzv. neúplný podmíněný příkaz `if`:

```
if (podmínka)
  příkaz_který_se_vykoná_pokud_je_podmínka_pravdivá;
```

A zde je příklad na neúplný podmíněný příkaz `if`:

```
<?php
  echo 'Toto je vypsáno ještě před konstrukcí if';
  // Výpis řetězce.
```

```
  echo '<br />';
  // Zařídí, aby se následující řetězec vypsal na další řádek

  if (3 > 2)
    echo 'Podmínka je pravdivá';
?>
```

**Příklad:** Soubor kap04_03.php

Příklad pracuje na stejném principu jako předchozí příklady na konstrukci `if`. Pouze se provádí výpis ještě před použitím konstrukce `if`, aby bylo vůbec něco vidět, pokud bude podmínka nepravdivá. Takže nejdříve se vypíše text: `Toto je vypsáno ještě před konstrukcí if`. Pak se vypíše značka `<br />`, která způsobí, že další text bude vypsán na další řádek. Podmínka v konstrukci `if` je zapsaná jako 3 > 2, tedy zjišťujeme, zda číslo 3 je větší než číslo 2. Protože tato podmínka je pravdivá, vykoná se příkaz a vypíše se `Podmínka je pravdivá`.

Zkuste příklad mírně upravit tak, že mu zadáte nepravdivou podmínku. To lze třeba takto:

```
<?php
  echo 'Toto je vypsáno ještě před konstrukcí if';
  // Výpis řetězce.

  echo '<br />';
  // Zařídí, aby se následující řetězec vypsal na další řádek

  if (3 > 9)
    echo 'Podmínka je pravdivá';
?>
```

**Příklad:** Soubor kap04_04.php

V tomto příkladu je podmínka změněna na zápis 3 > 9, tedy na porovnání, zda číslo 3 je větší než číslo 9. To je evidentně nepravdivá podmínka, v tomto případě tedy konstrukce `if` nevykoná nic. Veškeré, co tento příklad udělá, je to, že provede příkazy před konstrukcí `if`, tedy vypíše text: `Toto je vypsáno ještě před konstrukcí if`.

## Funkce die

V souvislosti s příkazem if se velmi často používá i funkce `die`, která slouží k okamžitému ukončení skriptu PHP, to znamená, že webový server přestane vykonávat další příkazy, které se ve skriptu PHP vyskytují. Před tím, než se skript PHP ukončí, se ještě naposledy vypíše nějaká zpráva, která je umístěna uvnitř funkce `die`. Takže třeba příkaz

```
die('Končím...');
```

způsobí okamžité ukončení skriptu PHP se zprávou `Končím...`

K čemu se funkce `die` používá? Hodně často k reakci na chyby. Pokud nastane nějaká chyba, kdy už nemá smysl pokračovat ve zbytku skriptu, použije se často právě funkce `die`. Tato funkce se bude hodně často používat v praktických příkladech, hlavně v kapitole o MySQL.

# Operátory porovnávání hodnot

Asi nejpřirozenějším způsobem používání podmínek v jazyce PHP je použití porovnávacích operátorů. Operátorem se nazývá zápis matematické operace. Při použití porovnávacích operátorů se jedná o takové běžné věci, kdy se ptáme, zda je jedno číslo menší než druhé nebo zda se dvě čísla rovnají apod. To je asi nejčastější způsob, jak budete psát podmínky.

Porovnávání bylo použito už v příkladech na podmíněný příkaz `if`. Nejspíše vám nebude dělat potíže jejich pochopení.

Nejdříve si tedy představme samotné možnosti porovnávání hodnot v PHP. Zde je přehledná tabulka (viz tabulka 4.1) základních porovnávacích operátorů (při čtení tabulky si představte, že máme dvě proměnné `$a` a `$b` a porovnáváme jejich hodnoty):

**Tabulka 4.1:** Základní operátory porovnávání hodnot

| Příklad: | Kdy je podmínka pravdivá: |
|---|---|
| $a == $b | Hodnota proměnné $a je rovna hodnotě proměnné $b |
| $a != $b | Hodnota proměnné $a není rovna hodnotě proměnné $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 |

Zkuste teď spustit třeba příklad na porovnání, zda hodnota v proměnné `$a` je rovna číslu 100:

```
<?php
  $a = 100;
  // Sem zkuste zadat i jiné číslo.

  if ($a == 100)
    echo 'Proměnná $a obsahuje číslo 100.';
?>
```

**Příklad:** Soubor kap04_05.php

Výše uvedený příklad po spuštění vypíše text: `Proměnná $a obsahuje číslo 100`. Pokud příklad upravíte a nastavíte proměnnou `$a` tak, aby jí bylo přiřazeno jiné číslo, nevypíše se nic.

# Datový typ boolean

Když PHP pracuje s podmínkami, zajímá ho vlastně jen jediná věc: zda je podmínka pravdivá nebo nepravdivá. PHP si dokonce pro tyto dva výsledky zavedl speciální hodnoty, které můžete přímo používat ve vašich skriptech PHP. Pokud je podmínka vyhodnocena jako nepravdivá, PHP použije hodnotu `false` (česky to znamená faleš nebo nepravda),

pokud je podmínka vyhodnocena jako pravdivá, PHP použije hodnotu `true` (česky to znamená pravý nebo pravda).

Pro ilustraci si zkuste spustit tento příklad:

```
<?php
  var_dump(1 != 1);
  // Vypíše: bool(false)
?>
```

**Příklad:** Soubor kap04_06.php

Pokud si tento příklad spustíte, dostanete tento výpis:

```
bool(false)
```

Co se vlastně stalo a jak tento příklad pracuje? Zjednodušeně by se dalo říci, že v příkladu je použita funkce `var_dump` pro výpis vyhodnocení podmínky. Podmínka je zapsaná jako parametr funkce `var_dump`. V příkladu je použit zápis podmínky 1 != 1, tedy zjišťujeme, jestli číslo 1 se nerovná číslu 1. Protože to evidentně není pravda, podmínka je vyhodnocena jako nepravdivá a vypíše se bool (false). Slovo `false` PHP používá, kdykoli mluví o nepravdivé podmínce.

Jak vidíte v příkladu, můžete použít funkci `var_dump` k tomu, abyste se dozvěděli, jak PHP vyhodnotí podmínku, tedy zda je podmínka pravdivá nebo nepravdivá. Pokud funkce `var_dump` vypíše

```
bool(false)
```

pak PHP podmínku vyhodnotil jako nepravdivou. Pokud naopak funkce `var_dump` vypíše

```
bool(true)
```

pak poznáte, že PHP vyhodnotil podmínku jako pravdivou. Můžete si tak vyzkoušet libovolnou podmínku jednoduše tím, že ji předáte funkci `var_dump`.

V tuto chvíli tedy víte, že PHP vyhodnocuje podmínky, a pokud je podmínka nepravdivá, označí to hodnotou `false`, pokud je podmínka pravdivá, označí to PHP hodnotou `true`. Ve skutečnosti jde PHP s podmínkami ještě mnohem dále. PHP povýšil práci s podmínkami na další typ dat, se kterými umí pracovat. Odborně se tomuto typu dat, se kterými podmínky pracují, říká logický datový typ.

Samotné PHP nazývá logický datový typ speciálním názvem jako tzv. `boolean` (v určitých případech PHP používá také název `bool`). Tím tedy máte informaci úplnou a už víte, co znamená to slovo `bool`, které vypisuje funkce `var_dump` ve výše uvedeném příkladu. Funkce `var_dump` vždycky vypisuje i jméno datového typu, takže při vypisování podmínek vždycky předřadí slovo `bool`.

Zkusíme teď trochu obtížnější věc. Pokud nebudete přesně rozumět, prostě přeskočte až na konec této podkapitoly a vraťte se sem, až budete trochu zběhlejší v PHP.

Protože v PHP se vyhodnocování podmínek děje jako práce s logickým datovým typem, můžete také používat všechno, co PHP nabízí pro práci s daty a datovými typy. Konkrétně jde o to, že například můžete podmínku přiřadit do proměnné, jak vidíte v tomto příkladu:

```
<?php
  $a = (3 != 4);
  // Do proměnné uložím výsledek vyhodnocení
  // podmínky, zda se číslo 3 nerovná číslu 4

  var_dump($a);
  // Vypíše bool(true)
?>
```

**Příklad:** Soubor kap04_07.php

# Operátory porovnávání hodnot s kontrolou typu

V PHP kromě základních operátorů pro porovnávání existují také dva speciální operátory, které porovnávají a zároveň kontrolují shodnost datového typu.

**Tabulka 4.2:** Operátory porovnávání hodnot s kontrolou typu

| Příklad | Kdy je podmínka pravdivá |
|---|---|
| $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 porovnáním a porovnáním s kontrolou typu lze snadno ukázat na 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 řádek.

  var_dump(1.0 === 1);
  // Podmínka je nepravdivá, protože nejsou shodné
  // typy. Číslo 1.0 patří do typu reálných čísel
  // a číslo 1 patří do typu celých čísel.
?>
```

**Příklad:** Soubor kap04_08.php

# Porovnávání řetězců

Porovnávání řetězců je trochu složitější oblast než porovnávání čísel. S porovnáváním čísel má každý zkušenosti a není potřeba dlouze vysvětlovat, které číslo je menší a jaké větší. Ovšem porovnávat se dají i řetězce a ve skriptech PHP se takové porovnávání řetězců používá velmi často.

## Tabulka ASCII

Pro porovnávání znaků, ale i v mnoha jiných situacích je užitečná tzv. tabulka znaků ASCII.

**Poznámka:** ASCII (American Standard Code for Information Interchange) je standard, který byl stvořen pro anglickou abecedu a jenž přiřazuje každému znaku nějaký číselný kód. Například písmenu A je přiřazen číselný kód 65. Tento standard je velice užitečný a široce využívaný.

ASCII je v podstatě způsob, jak každému znaku přiřadit jeho pořadové číslo, což můžete vidět v tabulce znaků ASCII (viz tabulka 4.3).

**Tabulka 4.3:** Tabulka znaků ASCII

| Znak | Kód |
|---|---|
| mezera | 32 |
| @ | 64 |
| ` | 96 |
| ! | 33 |
| A | 65 |
| a | 97 |
| " | 34 |
| B | 66 |
| b | 98 |
| # | 35 |
| C | 67 |
| c | 99 |
| $ | 36 |
| D | 68 |
| d | 100 |
| % | 37 |
| E | 69 |
| e | 101 |
| & | 38 |
| F | 70 |
| f | 102 |
| ' | 39 |

| Znak | Kód |
|---|---|
| G | 71 |
| g | 103 |
| ( | 40 |
| H | 72 |
| h | 104 |
| ) | 41 |
| I | 73 |
| i | 105 |
| * | 42 |
| J | 74 |
| j | 106 |
| + | 43 |
| K | 75 |
| k | 107 |
| , | 44 |
| L | 76 |
| l | 108 |
| - | 45 |
| M | 77 |
| m | 109 |
| . | 46 |
| N | 78 |
| n | 110 |
| / | 47 |
| O | 79 |
| o | 111 |
| 0 | 48 |
| P | 80 |
| p | 112 |
| 1 | 49 |
| Q | 81 |
| q | 113 |
| 2 | 50 |
| R | 82 |
| r | 114 |
| 3 | 51 |
| S | 83 |

| Znak | Kód |
|---|---|
| s | 115 |
| 4 | 52 |
| T | 84 |
| t | 116 |
| 5 | 53 |
| U | 85 |
| u | 117 |
| 6 | 54 |
| V | 86 |
| v | 118 |
| 7 | 55 |
| W | 87 |
| w | 119 |
| 8 | 56 |
| X | 88 |
| x | 120 |
| 9 | 57 |
| Y | 89 |
| y | 121 |
| : | 58 |
| Z | 90 |
| z | 122 |
| ; | 59 |
| [ | 91 |
| { | 123 |
| < | 60 |
| \ | 92 |
| \| | 124 |
| = | 61 |
| ] | 93 |
| } | 125 |
| > | 62 |
| ^ | 94 |
| ~ | 126 |
| ? | 63 |
| _ | 95 |
| (del) | 127 |

V uvedené tabulce znaků ASCII můžete vidět nejdůležitější část z ní, tj. všechny znaky, které mají přiřazen kód od 32 do 127.

Podle ASCII se porovnávají i znaky. Za menší je považován ten znak, který má nižší pořadové číslo. Například znak `a` má pořadové číslo 97, zatímco znak `g` má pořadové číslo 103. Z toho vyplývá, že znak `a` je menší než znak `g`.

## Princip porovnávání řetězců

Pokud porovnáváte čísla, je celkem evidentní, jak to dopadne a které číslo má být větší apod. U řetězců je to trochu jinak. V zásadě se při porovnávání dvou řetězců používá tento postup:

1. Začnou se porovnávat znaky obou řetězců hezky od začátku.
2. Nejdříve se porovnají první znaky obou řetězců, pak druhé a tak dále.
3. Jakmile se narazí na první rozdíl (tedy když oba znaky nebudou stejné) 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.

Jakmile narazí PHP na místo, kde jsou znaky na stejné pozici v obou řetězcích rozdílné, pak rozhodují tyto znaky. Porovnává se podle tabulky znaků ASCII. Platí, že třeba znak „c“ je menší než znak „x“. Ale také platí, že libovolné velké písmeno je menší než libovolné malé písmeno. Který znak je menší a jaký je větší, se řídí podle tabulky ASCII (viz tabulka 4.3).

Příklady takového porovnávání řetězců jsou uvedeny v následující tabulce (viz tabulka 4.4).

**Tabulka 4.4:** Ukázky porovnávání řetězců

| První řetězec | Druhý řetězec | Výsledek porovnání |
|---|---|---|
| 'abc' | 'abx' | První řetězec je menší než druhý řetězec. |
| 'XX' | 'xx' | První řetězec je menší než druhý řetězec. |
| 'sova' | 'sova' | Oba řetězce jsou shodné. |
| 'xy' | 'xyz' | První řetězec je menší než druhý řetězec. |

## Porovnávání dvou řetězců v podání PHP

V podkapitole *Princip porovnávání řetězců* byl vysvětlen princip porovnávání dvou řetězců. Ovšem princip je jenom princip a v PHP je to maličko složitější. Pokud použijete porovnávání hodnot s kontrolou typu, tedy `===` nebo `!==`, pak se porovnávání řetězců chová naprosto přesně tak, jako bylo popsáno v části *Princip porovnávání řetězců*:

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

**Příklad:** Soubor kap04_09.php

Pokud spustíte výše uvedený příklad, dostanete výpis:

```
bool(false)
```

Tento výpis sděluje, že podmínka uvedená v příkladu je nepravdivá. Protože podmínka je zapsána jako `'xy' === 'xyz'`, zjišťuje se, jestli řetězec `xy` je rovný řetězci `xyz`. Každý řetězec je jiný, což značí, že se řetězce sobě nerovnají, a podmínka je tedy vyhodnocena jako nepravdivá. Proto tedy bude vypsáno `bool(false)` jako u všech nepravdivých podmínek.

Trochu jiná situace nastane, pokud použijete běžné porovnávání hodnot bez kontroly typu, tedy použijete == nebo != (případně <, <=, > nebo >=). Pak se chová PHP při porovnávání řetězců trochu jinak a dost zvláštně. Upřímně řečeno, nevím, co k tomu autory PHP vedlo, ale tato oblast chování PHP může vést k obtížím.

Jak se tedy PHP chová, pokud nastane ten případ, že porovnáváte dva řetězce běžným způsobem? PHP se totiž nejdříve podívá, jestli jdou oba řetězce převést na číslo. Pokud ano, porovná oba řetězce jako čísla. Pokud jeden nebo oba porovnávané řetězce na číslo převést nejdou, porovnává je PHP jako texty.

```
<?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)
?>
```

**Příklad:** Soubor kap04_10.php

**Poznámka:** Převod řetězce na číslo se v tomto případě provádí následovně. Řetězec se převede na číslo jenom tehdy, pokud celý řetězec představuje číslo. Tedy třeba řetězec `10` se převede na číslo 10, ale řetězec `10a` už se na číslo nepřevede, protože tento řetězec obsahuje víc než číslo.

Tabulka 4.5 přináší několik příkladů toho, jak se PHP chová při porovnávání řetězců:

**Tabulka 4.5:** Ukázky porovnávání řetězců v PHP

| Podmínka: | Výsledek podmínky: |
|---|---|
| '+10.0' == '10' | Podmínka je pravdivá (porovná se jako čísla) |
| '2' == '2mamuti' | Podmínka není pravdivá (porovná se jako řetězce) |
| 'abc' != 'xyz' | Podmínka je pravdivá (porovná se jako řetězce) |
| '8' == '+8' | Podmínka je pravdivá (porovná se jako čísla) |

# Porovnávání řetězce s číslem

Porovnávat v podmínkách se dá samozřejmě ledacos. Nemusíte porovnávat jenom čísla nebo jenom řetězce, ale můžete také porovnávat řetězec s číslem. V takovém případě se PHP chová jednoznačně, a to podle stylu všechno se bude porovnávat jako číslo. To občas vede k neočekávaným výsledkům.

Funguje to takto: Pokud porovnáváte řetězec s číslem, převede se řetězec na číslo a PHP pak dále postupuje, jako kdyby porovnával rovnou dvě čísla.

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

**Příklad:** Soubor kap04_11.php

Výše uvedený příklad demonstruje, jak to PHP provádí. Příklad obsahuje zápis podmínky `'100' > 40`. Protože se porovnává řetězec s číslem, snaží se PHP porovnávat čísla. Proto převede řetězec „100“ na číslo, což je v tomto případě jednoduché. Pak už PHP jen porovná, jestli je `100 > 40`, tedy jestli je číslo 100 větší než číslo 40. Protože to je pravda, PHP vyhodnotí podmínku jako pravdivou. Vypíše se tedy `bool(true)`, což funkce `var_dump` vypisuje při každé pravdivé podmínce.

Při převádění samotného řetězce na číslo se chová PHP tak, že se snaží na začátku řetězce najít číslo. Pokud ho najde, výsledkem převodu je právě ono číslo. Pokud na začátku řetězce žádné číslo nenajde, výsledkem převodu je nula. Další podrobnosti najdete v části *Automatický převod řetězce na číslo*.

Tabulka 4.6 uvádí pár příkladů:

**Tabulka 4.6:** Ukázky porovnávání řetězců s číslem

| Podmínka: | Výsledek podmínky: |
|---|---|
| '+1' > 0 | Podmínka je pravdivá (řetězec '+1' se převede na jedničku) |
| '23psů' == 23 | Podmínka je pravdivá (řetězec '23psů' se převede na číslo 23) |
| '-2abc' > 10 | Podmínka není pravdivá (řetězec '-2abc' se převede na číslo -2) |
| 10 != 'cihla' | Podmínka není pravdivá (řetězec 'cihla' se převede na nulu) |

# Logické operátory

Při práci s podmínkami se velmi často stává, že prostě jednoduché podmínky nestačí. Poměrně frekventovanou potřebou je kombinování více podmínek dohromady. Pro tyto účely existují v PHP logické operace.

## Logická negace

Nejjednodušší logickou operací je negace, tedy obrácení podmínky. Dělá pouze to, že z pravdivé podmínky udělá nepravdivou a naopak, tedy z nepravdivé podmínky udělá pravdivou. V PHP se logická operace negace zapisuje tak, že podmínce předřadíme znak vykřičník (!).

```
<?php
  var_dump(10 > 20);
  // Tato podmínka je nepravdivá.
  // Vypíše: bool(false)

  echo '<br>';
  // Zajistí, aby se další výpis vypsal
  // na další řádek.

  var_dump(!(10 > 20));
  // Tato podmínka je pravdivá, protože
  // je použita operace negace, která otočila
  // pravdivost podmínky.
  // Vypíše: bool(true)
?>
```

**Příklad:** Soubor kap04_12.php

**Upozornění:** Logická negace má v PHP velmi vysokou prioritu. Pokud se chcete vyhnout problémům, postačí, když si zapamatujete, že při použití logické negace je většinou nutné nešetřit závorkami. Vyplatí se uzavírat podmínku, kterou chcete obrátit, do závorek. Pokud tedy použijete následující zápis, nejspíše dostanete to, co očekáváte:

```
!(10 >= 0) // Výsledek je nepravda
```

Pokud ovšem napíšete totéž bez závorek, dostanete úplně jiný výsledek:

```
!10 >= 0 // Výsledek je pravda
```

## Logické operace and a or

Logické operace `and` a `or` slouží ke spojení dvou podmínek tak, aby se výsledek tvářil jako jedna podmínka. Obě operace se liší pouze tím, jak vypočítávají výslednou hodnotu podmínky v závislosti na obou původních podmínkách, ze kterých jsou složeny.

Zkusme si představit nejdříve logickou operaci `and`. Její činnost je možné vyjádřit slovním spojením *„a zároveň“*. Logická operace `and` vezme dvě podmínky a považuje je za pravdivé tehdy, jestliže jsou pravdivé obě zároveň. Pokud je první nebo druhá podmínka nepravdivá, případně obě, výsledkem logické operace `and` je také nepravda.

Jako jednoduchý ilustrační příklad logické operace `and` může posloužit třeba situace, kdy potřebujete zkontrolovat, zda celé číslo je platnou školní známkou. Ve škole se dávají známky od 1 do 5, tudíž potřebujeme zjistit, zda číslo leží v tomto intervalu. To však nelze zjistit jednoduchou podmínkou. Lze to však napsat dvěma jednoduchými podmínkami. První by zkontrolovala, zda je číslo větší nebo rovno jedné, druhá by zkontrolovala, zda je číslo menší nebo rovno pětce. Pokud by platily obě podmínky najednou, pak by číslo bylo platnou školní známkou. Takto by to vypadalo zapsáno v PHP:

```
<?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.
?>
```

**Příklad:** Soubor kap04_13.php

Činnost logické operace `or` je možné vyjádřit slovem *„nebo“*. Logická operace `or` vezme dvě podmínky a složí je do jednoho výsledku. Pokud je první nebo druhá podmínka pravdivá, výsledek bude brán jako pravda. Jinak řečeno, logické operaci `or` stačí, aby jedna nebo obě původní podmínky byly pravdivé k tomu, aby vyšla pravda. Nepravda vyjde jen tehdy, když obě původní podmínky jsou nepravdivé.

```
<?php
  $a = 2;

  var_dump($a > 10 or $a == 2);
  // Logická operace or složená ze dvou podmínek.
  // Výsledek je pravdivá podmínka, protože druhá
  // původní podmínka je pravdivá.
  // Vypíše: bool(true)
?>
```

**Příklad:** Soubor kap04_14.php

## Dvojí zápis pro logické operace and a or

PHP má dvojí zápis pro logické operace `and` a `or`. Kromě přímého použití slov `and` a `or` můžeme použít také zápis `&&` pro logickou operaci `and` a `||` pro logickou operaci `or`. Tento druhý způsob zápisu inspirovaný programovacím jazykem C uvidíte v různých skriptech PHP velmi často. Takže třeba tento zápis:

```
$a > $b and $a != 3
```

je naprosto totožný se zápisem:

```
$a > $b && $a != 3
```

Ačkoli v drtivé většině případů nebude žádný rozdíl mezi tím, zda použijete první způsob zápisu (tedy slova `and` či `or`) nebo druhý způsob (tedy zápis `&&` či `||`), přesto je mezi nimi malý rozdíl. Ten nespočívá v samotné činnosti, která je pokaždé stejná, ale v prioritě, tedy v tom, jaké operace má přednost, pokud jich použijete více dohromady. Nechci zde ale nyní zacházet do detailů. Proto vám pouze doporučím, abyste používali v zápisu pokud možno slovní tvary `and` a `or`, které mají příznivou nízkou prioritu, obvykle odpovídající programátorovým záměrům.

**Tip:** Ačkoli můžete v PHP používat dva různé zápisy pro logické operace, a to jednak **and** a **or**, jednak ekvivalentní **&&** a **||**, používejte raději první způsob, protože má lepe umístěnou prioritu. Přesto je nutné znát způsoby oba, protože v mnoha skriptech PHP se používá způsob druhý.

## Neúplné vyhodnocování logických výrazů pomocí and a or

Pokud PHP zpracovává podmínky vytvořené pomocí logických operací `and` a `or`, snaží se postupovat maximálně úsporně. Ctí zásadu zdravé lenosti. V mnoha případech není potřeba vyhodnotit celou podmínku, ale stačí pouze vyhodnotit jen určitou část podmínky, aby bylo jasné, zda je pravdivá nebo nepravdivá. PHP toho plně až nestydatě využívá, a tak vyhodnocuje jen tu část podmínky, kterou je nezbytně potřeba vyhodnotit.

Představte si následující podmínku:

```
1 > 3 and 3 <= 4
```

Tato podmínka používá logickou operaci `and`, o které je známo, že jejím výsledkem bude pravda jenom tehdy, když se skládá ze dvou pravdivých podmínek. Protože první podmínka (tedy 1 > 3) je nepravdivá (není pravda, že 1 je větší než 3), je už v této chvíli jasné, že výsledkem celého zápisu bude nepravda, a není vůbec nutné se druhou podmínkou jakkoli zabývat. Přesně tak to udělá i PHP a v tomto případě vůbec druhou podmínku nevyhodnotí.

Pokud se nad tím tedy zamyslíme, dojdeme k tomuto závěru: Je-li použita logická operace `and` a první podmínka je nepravdivá, jasně vyjde nepravda a PHP už nebude vyhodnocovat druhou podmínku. Je-li použita logická operace `or` a první podmínka bude pravdivá, jasně vyjde pravda a PHP také nebude vyhodnocovat druhou podmínku.

Tato lenost při vyhodnocování vám při použití „normálních“ podmínek za běžných okolností k ničemu užitečná nebude (snad až na to, že vám skripty PHP běží teoreticky nepatrně rychleji). Úplně jiné to ovšem je, pokud mají vaše podmínky nějaké vedlejší efekty. Například při vyhodnocování podmínek se může stát, že dojde ke změně nějaké proměnné. Pak je nutné s tímto efektem počítat. Až budete programovat skutečné skripty PHP nebo se budete dívat do cizích skriptů, zjistíte, že lenost ve vyhodnocování podmínek je ve skutečnosti využívána v PHP velmi často.

# Test

1. K čemu slouží v PHP příkaz if?
2. Jaký je rozdíl mezi úplným a neúplným příkazem if?
3. Pomocí jakého operátoru porovnáte dvě čísla na rovnost?
4. Jaký význam mají pro PHP hodnoty false a true?
5. Jaký je rozdíl při porovnávání pomocí == (2 rovnítka) a pomocí === (3 rovnítka)?
6. Co je to ASCII?