Řídící struktury

PVA4 Programování a vývoj aplikací

Adam Fišer

Obsah

Úvod

- Podmínky
  - if
  - switch Cykly
  - for
  - foreach
  - while

Podmínky


FREIHEIT
ERKÄMPFEN!
photos,
ge. Let's
anymore.
ns have to
utcome of
olicy.
RONAL
CHTE H
3.1.2021
if www
HAU4.d
V.i.s.d.P.
n:
witz/M
aue
ar 2021
er P
P

Podmínky

- podmínka je tvořena klíčovým slovem `if` a definicí podmínky
- složené závorky `{}` pravý `alt` + `B` nebo `N` obsahují příkazy, které se vykonají, pokud je podmínka splněna

```
1   if ( podmínka) {
2     // příkazy
3   }
```

- u jednořádkových příkazů můžeme složené závorky vynechat

```
1   if ($a > $b)
2     echo "a je větší než b";
```

Ternární operátor

- zkrácený zápis podmínky
- `podmínka ? hodnotaTrue : hodnotaFalse`

```
1   echo ($a > $b) ? "a je větší než b" : "a je menší než b";
```

Operátory

- `$a == $b` - (`=` `=`) `$a` má stejnou hodnotu jako `$b`
- `$a ≠ $b` - (`!` `=`) `$a` má jinou hodnotu než `$b`
- `$a > $b` `$a` - je větší než `$b`
- `$a < $b` `$a` - je menší `$b`
- `$a ⩽ $b` - (`<` `=`) `$a` je menší nebo rovno `$b`

Logické operátory

- `!` - negace, `!$x` proměnná `$x` není `true`
- `$a === $b` - (`=` `=` `=`) `$a` má stejnou hodnotu A typ jako `$b`
- `$a !== $b` - (`!` `=` `=`) `$a` má jinou hodnotu NEBO typ než `$b`
- logický operátor AND - musí být oba pravdivé
  - `$a and $b`
  - `$a && $b` (`AltGr` + `C`)
- logický operátor OR - alespoň jeden musí být pravdivý
  - `$a or $b`
  - `$a || $b` (`AltGr` + `W`)
- `$a ?? $b` - null koalesční operátor - musí být stejný datový typ. Pokud je `$a` null, vrátí `$b`
- `$a ?: $b` - ternární operátor - pokud je `$a` pravda, vrátí `$a`, jinak `$b`
- `$a xor $b` - logický operátor XOR - jeden z nich musí být pravdivý, ale ne oba

Operátory rovnosti

Pozor na datové typy!

- `0 == 0` je pravda
- `000 == 0` je nepravdivý
- `foo = 0` je pravda
- `000 = 0` je pravda
- `"0" = 0` je pravda
- Chcete-li zkontrolovat, zda jsou dva skaláry, které mají být číselné, stejné, použijte `=`, např. `5 = "5"` je pravda.
- Chcete-li zkontrolovat, zda jsou dvě proměnné typu „řetězec“ a jsou stejné posloupnosti znaků, použijte `==`, například `1.e6" == "1.0e6` je nepravda.

Příklad

```php
if ($var == "abc" || $var == "def" || ...)
{
    // Alespoň jedna podmínka musí být true
    echo "true";
}
```

```php
if ($var1 == "abc" && $var2 == "def" && ...)
{
    // Všechny podmínky musí být vyhodnoceny jako true
    echo "true";
}
```

Větvení podmínek `else`, `elseif`

```
if (podmínka) {
  // příkazy
} elseif (podmínka) {
  // příkazy
} else {
  // příkazy
}
```

```
if ($cislo > 0) {
    echo $cislo . ' je kladné';

} elseif ($cislo < 0) {
    // Vykoná se, pokud první podmínka je vyhodnocena jako false
    // a je splněna definovaná podmínka
   echo $cislo . 'je záporné';

} else {
   // Vykoná se, pokud není splněna žádná podmínka
   echo $cislo .'je nula';

}
```

`switch`

- `switch` je alternativou k `if` a `elseif`
- Používá se, pokud chceme porovnávat hodnoty, nikoliv podmínky
- `switch` porovnává hodnotu proměnné s hodnotami v `case`
- `break` ukončí `switch` blok
- `default` je volitelný blok, který se vykoná, pokud žádný `case` nebyl splněn

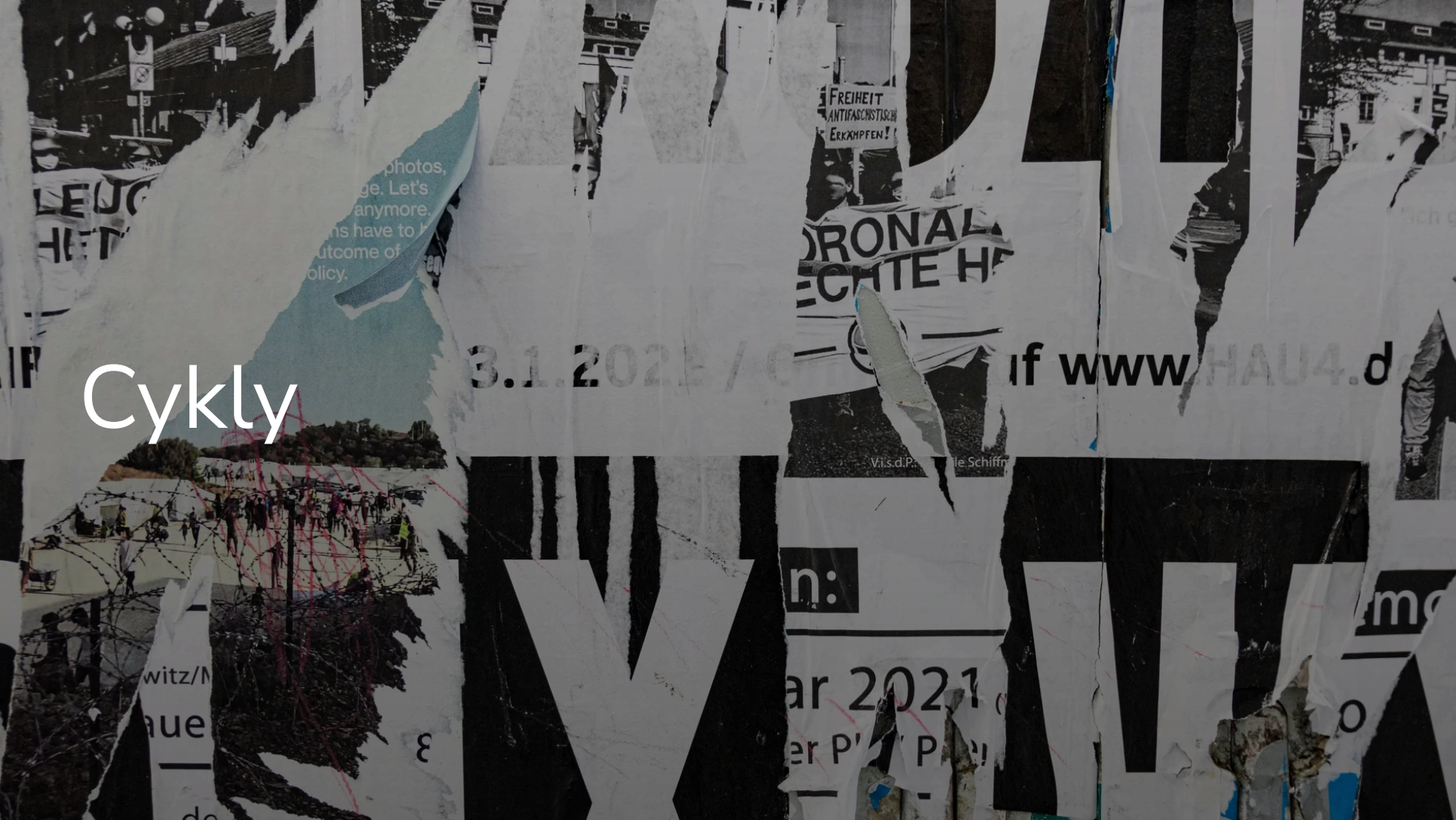
```
1   switch ($vstupniHodnota) {
2     case hodnota1:
3       // příkazy, pokud je splněna hodnota1
4       break;
5     case hodnota2:
6       // příkazy, pokud je splněna hodnota2
7       break;
8     default:
9       // příkazy, pokud nebyla splněna žádná hodnota
10  }
```

switch

```php
$den = "pondělí";

switch ($den) {
    case "pondělí":
        echo "Dnes je pondělí.";
        break;
    case "úterý":
        echo "Dnes je úterý.";
        break;
    case "středa":
        echo "Dnes je středa.";
        break;
    default:
        echo "Dnes nevím, který den je.";
}
```

Cykly


FREIHEIT
ERKÄMPFEN!
photos,
ge. Let's
anymore.
ns have to
utcome of
olicy.
ar 2021
witz/M
aue

Cyklus `for`

- `for` cyklus se skládá ze tří částí
  - inicializace proměnné
  - podmínka
  - iterace

```php
// for (inicializace čítače; testovací čítač; inkrement čítače)
for ($i = 0; $i < 10; $i++) {
  // příkazy
}
```

Cyklus `foreach`

- Smyčka bloku kódu pro každý prvek pole
- Umožňuje iterovat přes pole a získávat pouze hodnoty prvků v poli.
- foreach postupně projde všechny prvky v poli `$colors` a při každé iteraci bude proměnná `$value` obsahovat hodnotu aktuálního prvku v poli.

```
1  foreach ($pole as $hodnota) {
2    // příkazy
3  }
```

```
1  $colors = array("red", "green", "blue", "yellow");
2
3  foreach ($colors as $value) {
4      echo $value .'<br />';
5  }
```

Cyklus `foreach` s klíčem

- `as $key ⇒ $value` - pokud chceme získat klíč a hodnotu

```php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $key ⇒ $value) {
  echo 'Klíč: ' . $key . ', Hodnota: ' . $value . '<br />';
}
```

Cyklus `while`

- `while` cyklus se skládá z podmínky, která se vyhodnocuje před každou iterací
- Pokud je podmínka vyhodnocena jako `true`, cyklus se opakuje
- Pokud je podmínka vyhodnocena jako `false`, cyklus se ukončí
- Pozor na zacyklení!

```
while (podmínka) {
  // příkazy
}
```

```
$i = 1;

while ($i ⩽ 10) {
  echo $i . '<br />';
    $i++;
}
```

Cyklus `do ... while`

- `do ... while` cyklus se skládá z podmínky, která se vyhodnocuje po každé iteraci
- Cyklus se vykoná alespoň jednou, i když je podmínka `false`

```
do {
  // příkazy
} while (podmínka);
```

```
$i = 20;
do {
    echo $i . '<br />';
    $i++;
} while ($i ⩽ 10);

// Výstup: 20, ikdyž podmínka je false
```

Funkce

Funkce vs Procedura

- Funkce je blok kódu, který můžeme znovu použít
- může přijímat parametry a vrací hodnotu
- Procedura je blok kódu, který může být znovu použit, ale nevrací hodnotu

Interní funkce

- PHP má více než 1000 vestavěných funkcí a navíc si můžete vytvořit vlastní funkce.
- Kompletní přehled funkcí je uveden na php.net
- Každou funkci lze volat přímo ze skriptu a provést specifickou úlohu.

Uživatelsky definovalé funkce

- Funkce můžete definovat pomocí klíčového slova `function`
- Nesputí se, dokud nejsou zavolány tzn. nejsou spouštěny automaticky ani při načtení stránky
- Název funkce musí začínat písmenem nebo podtržítkem, na velikosti písmen nezáleží
- Návratová hodnota přes `return`
- Datový typ návratové hodnoty je uveden za dvojtečkou

```
1  function nazevFunkce(): datovyTyp {
2    // příkazy
3
4    // návratová hodnota
5    return $vysledek;
6  }
```

Volání funkce

- Funkci zavoláme pomocí jejího názvu a závorek
- Pokud funkce přijímá parametry, musíme je uvést v závorkách

```
1   nazevFunkce();
```

Ukázka

```
1   function pozdrav(): string {
2     return "Ahoj!";
3   }
4
5   echo pozdrav();
```

Parametry funkce

- Parametry jsou hodnoty, které funkce přijímá a jsou nutné pro vykonání funkce
- jsou uvedeny v závorkách a odděleny čárkou ve tvaru `datovyTyp parametr1, datovyTyp parametr2, ...`
- Mají pouze lokální platnost
- Můžeme nastavit výchozí hodnotu parametru přiřazení hodnoty za `=`

```
1  function nazevFunkce(string $parametr1, int $parametr2 = null): string {
2    // příkazy
3  }
```

Příklad

```php
// zapnutí vyžadování dodržování kontroly datových typů
// standardně se uvádí hned za <?php
declare(strict_types=1);
```

```php
function writeMessage(string $name, string $msg = "nic nenapsal"): string {
  $output = $name . ' napsal: ' . $msg;
  return $output;
}
```

```php
// volání funkce
echo writeMessage('Honza', 'První zpráva');             # Honza napsal: První zpráva
echo writeMessage('David', 'Druhá zpráva');             # David napsal: Druhá zpráva
echo writeMessage('Ilona', 'Jiná zpráva do výstupu');   # Ilona napsal: Jiná zpráva do výstupu
echo writeMessage('Petr');                              # Petr napsal: nic nenapsal
```

Děkuji za pozornost

Otázky?

Repository / Prezentace