Přednáška 3: Pokročilé PHP

Funkce (1/6)

Funkce
Funkce je blok kódu, který **jednou nadefinujeme**, a pak jej můžeme **opakovaně volat** z libovolného místa skriptu. Funce může, ale nemusí obdržet při svém volání **parametry** a může, ale nemusí vracet **hodnotu**. Pokud ji nevrací, pak se jedná o obdobu procedury známé z jiných programovacích jazyků.

PHP obsahuje mnoho **předdefinových funkcí** (viz. min. přednáška a např. funkce isset(), empty() ad.)

Vlastní uživatelsky definované funkce
Funkce se deklarují příkazem function. Parametry (argumenty) se oddělují čárkami. Tělo funkce musí být uvedeno mezi složenými závorkami. Volání funkce se provádí zápisem jejího názvu vč. jednoduchých závorek s argumenty. Volání funkce může být uvedeno před její deklarací.

```
function nazev_funkce(parametry) {
    tělo funkce
}
```

Příklad:

```
<?php
// deklarace a definice funkce
function treti_mocnina($prom) {
            return $prom*$prom*$prom;
}
?>
```

Příklad volání funkce:

```
<?php
// vytisteni vysledku
echo treti_mocnina(3);
// ulozeni vysledku do promenne
$vysledek = treti_mocnina(3);
echo $vysledek;
?>
```

Přednáška 3: Pokročilé PHP

Funkce (2/6)

Volání funkce (např. funkce treti_mocnina() z předchozí funkce) se vyhodnotí jako výraz a vrátí hodnotu výsledku výrazu. Návratová hodnota je určena příkazem **return**, který musí být uveden uvnitř těla funkce. Po provedení příkazu return se vykonávání dalších příkazů uvnitř těla funkce **zastaví** a program pokračuje **v místě volání** funkce. Příkaz return není povinný, viz. následující příklad.

Příklad:

```
<?php
// funkce bez příkazu return
function treti_mocnina($prom) {
        echo $prom*$prom*$prom;
}
?>
```

Předávání parametrů funkci

Parametry umožňují předat funkci **vstupní hodnoty** ve formě **proměnných**. Tyto proměnné jsou pak **dostupné** pouze **uvnitř těla funkce**.

Příklad:

```
<?php
// funkce vypisuje radky tabulky
function vypis_radky($n) {
    for ($i=1;$i<=n;$i++) {
        echo ”<tr>\n”;
        echo ”<td></td>\n”;
        echo ”</tr>\n”;
    }
}
?>
```

Příklad volání funkce pro výpis řádků tabulky:

```
<?php
// vytisteni tabulky s 3 řádky
echo ”<table>\n”;
vypis_radky(3);
echo ”</table>\n”;
?>
```

Internet a zdravotnická informatika

Přednáška 3: Pokročilé PHP

Funkce (3/6)

Předávání parametrů hodnotou nebo odkazem
Výchozí předávání parametrů funkci se nazývá **předávání parametrů hodnotou**. To znamená, že proměnná uvnitř funkce definovaná parametrem pracuje **s kopií hodnoty**, která je funkci předávána.

Příklad:
```
<?php
// předání parametru hodnotou
function dvojnasobek($n) {
    $n = $n * 2;
    echo $n;
}
?>
```

Příklad volání funkce s předáním parametru hodnotou:
```
<?php
$a = 5;
dvojnasobek($a); // vypíše 10
echo $a; // vypíše 5
?>
```

Při **předávání parametrů odkazem** se změny, provedené uvnitř funkce, **projeví v proměnné**, která je funkci **předána při volání**. Znak **& před názvem** proměnné definuje parametr, předávaný odkazem.

Příklad:
```
<?php
// předání parametru odkazem
function zvyseni(&$plat, $procent) {
    $plat += $plat * $procent/100;
}
?>
```

Příklad volání funkce s předáním parametru odkazem:
```
<?php
$pl = 20000; // výchozí výše platu
echo $pl; // vypíše 20000
zvyseni($pl, 10); // zvýšení platu o 10%
echo $pl; // vypíše 22000
?>
```

Přednáška 3: Pokročilé PHP

Funkce (4/6)

Výchozí hodnota parametru
Parametru funkce je možné přiřadit **výchozí hodnotu**, to zapříčiní, že parametr bude **nepovinný**.

Příklad:

```
<?php
// výchozí hodnota parametru funkce
function prumer($soucet, $pocet, $zaok = ”m”) {
    $vysledek = $soucet / $pocet;
    if ($zaok ==”m”) echo round($vysledek);
    elseif ($zaok ==”d”) echo floor($vysledek);
    elseif ($zaok == ”n”) echo ceil($vysledek);
    else echo $vysledek; // bez zaokrouhlení
}
?>
```

Příklad výchozího parametru funkce:

```
<?php
prumer(20, 3); // vypíše 6.6666667
prumer(20, 3, ”m”); // vypíše 6
prumer(20, 3, ”d”); // vypíše 6
prumer(20, 3, ”n”); // vypíše 7
?>
```

V deklaraci funkce je důležité umístit parametry s výchozi hodnotou (nepovinné) vždy **na konec výčtu parametrů**.

Přednáška 3: Pokročilé PHP

Funkce (5/6)

Rozsah platnosti proměnných
Rozsah platnosti proměnné určuje, které části programu budou mít k proměnné přístup.

Platnost proměnných

```
<?php
$a = 1; // globální pohled
function test()
{
    echo $a; // lokální pohled
}
test(); // nevypíše nic
?>
```

Použití předdefinovaného pole **$GLOBALS** pro přístup k globálním proměnným

```
<?php
$a = 1;
$b = 2;
function sum()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}
sum();
echo $b; // vypíše 3
?>
```

Přednáška 3: Pokročilé PHP

Funkce (6/6)

Poznámky k funkcím

- Funkce může mít více vstupních parametrů. Ty se pak oddělují čárkou.
- Funkce v PHP mohou předávat parametry hodnotou (většinou) nebo odkazem. To se provede tak, že před název proměnné v hlavičce funkce se uvede ampersand (&).
- Funkce v PHP mohou být rekurzivní. To znamená, že funkce může volat sebe samu. Kromě učebnicového příkladu výpočtu faktoriálu se v PHP používají rekurze na úlohy typu procházení adresářem, apod.
- Funkce nemůže vracet více než jeden výstupní parametr. Lze to obejít tak, že funkce vrací pole.
- Jedna uživatelská funkce může volat jinou. Na pořadí, v jakém jsou uvedeny ve skriptu, přitom nezáleží. Volání funkce může být uvedeno dokonce již dříve než definice funkce samotné.

Předpřipravené funkce

- Funkce pro práci s **řetězci**
- Funkce pro práci s **poli**
- Funkce pro práci se **soubory**
- Funkce pro práci s **databází**

apod.

Internet a zdravotnická informatika

Přednáška 3: Pokročilé PHP

Předdefinované proměnné

$GLOBALS - Obsahuje odkaz na každou proměnnou, která je momentálně dostupná v globálním kontextu skriptu. Klíči tohoto pole jsou názvy globálních proměnných.

$_SERVER - Pole proměnných www serveru a spjatých s ním.

$_GET - Pole proměnných poskytovaných skriptu přes HTTP metodu GET.

$_POST - Pole proměnných poskytovaných skriptu přes HTTP metodu POST.

$_COOKIE - Pole proměnných cookies.

$_FILES - Pole proměnných poskytovaných skriptu přes HTTP post uploady souborů.

$_REQUEST - Pole proměnných poskytovaných skriptu přes libovolný vstupní mechanismus (např. POST nebo GET)

$_SESSION - Pole proměnných, které jsou momentálně registrovány v aktuální relaci skriptu.

$_ENV - Pole proměnných poskytovaných skriptu z prostředí.

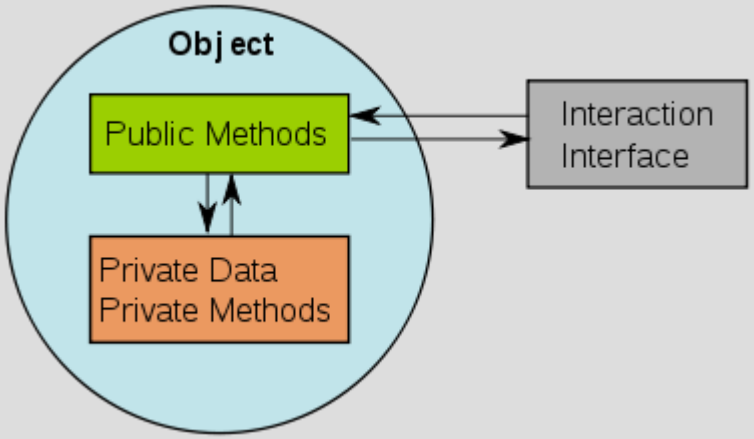
Přednáška 3: Pokročilé PHP

OOP (1/5)

Myšlenka **objektového programování** je prostá. Zatímco procedurální programování používá při vykonávání výpočetních úloh proměnné a funkce, objektové programování používá zvláštní struktury nazvané **objekty**. Objekt obsahuje jak **data** (například proměnné) tak i popis způsobů, jak s nimi manipulovat, tzv. **metody**.

Hlavní výhodou objektů je fakt, že umožňují **popisovat věci tak, jak fungují v běžném životě**.

Objekt je jednoznačně identifikovatelná entita, která má určité **vlastnosti (atributy)** a **chování**. OOP ztělesňuje nástroj, kterým lze kód chování „ukrýt" uvnitř objektu. Jinými slovy jsou objekty programátorskou konstrukcí sdružující dohromady **proměnné** (vlastnosti, atributy) a **funkce**, které s těmito proměnnými pracují (akce, chování). Uživateli objektů stačí **znát jenom určité funkce objektu** (rozhraní) a o vnitřní implementaci se nemusí starat. Tento koncept je známý pod pojmem ***zapouzdření*** - vnitřní implementace mechanismu je oddělena od jeho rozhraní.



Přednáška 3: Pokročilé PHP

OOP (2/5)

Objekty se v OOP musejí nejprve **nadefinovat**. K definici objektu slouží takzvané **třídy**. Třída je něco jako šablona nebo prototyp, na jehož základě se budou objekty tvořit; v PHP se třída definuje pomocí klíčového slova **class**.

```
<?php
class Student
{
    // definice atributů
    var $jmeno;
    var $prijmeni;
    var $vek = 21;
}
?>
```

```
<?php
class Student
{
    // definice atributů
    var $jmeno;
    var $prijmeni;
    var $vek = 21;

    // definice metody
    function vratVek() {
        return $this->vek;
    }
}
?>
```

Pro **přístup k atributům** objektu se používá operátor **->**.
Uvnitř těla třídy se metody definují úplně **stejně jako funkce**.

Přednáška 3: Pokročilé PHP

OOP (3/5)

Používání třídy

Pomocí klíčového slova **new** se vytvoří **instance** neboli objekt třídy. Konkrétnímu objektu (instanci) se přiřadí **atributy** (jmeno, prijmeni a vek) a následně se použije **metoda** vratVek(), která vypíše věk konkrétního studenta.

```php
<?php
$student_IZI = new Student;
$student_IZI->prijmeni = ”Novák”;
$student_IZI->jmeno = ”Josef”;
$student_IZI->vek = 20;
echo $student_IZI->vratVek(); // vypíše 20
echo $student_IZI->prijmeni; // vypíše Novák
?>
```

Samozřejmě, že bychom mohli vytvořit **více instancí** třídy Student a každá by mohla mít jiné atributy.
Třída nemusí, ale může definovat metodu, která se jmenuje stejně jako třída samotná a nazývá se **konstruktor**. Konstruktor se spustí automaticky při vytvoření instance a je možné jej např. použít pro nastavení výchozích hodnot.

```php
<?php
class Student
{
    // definice atributů
    var $jmeno;
    var $prijmeni;
    var $vek;

    // definice konstruktoru
    function student() {
        $this->vek = 20;
    }
}
?>
```

Přednáška 3: Pokročilé PHP

OOP (4/5)

Dědičnost
Dědičnost je jedna ze základních vlastností OOP a slouží k tvoření **nových datových struktur** na základě **starých**.

```php
<?php
class Student extends Clovek {
    // definice atributů
    var $studijniProgram;

    // definice metody
    function vratStudijniProgram() {
        return $this->studijniProgram;
    }
}
$ja = new Student;
$ja->vek = '21';
$ja->studijniProgram = 'BMT';
echo $ja->vratVek() . '<br>';
echo $ja->vratStudijniProgram() . '<br>';
?>
```

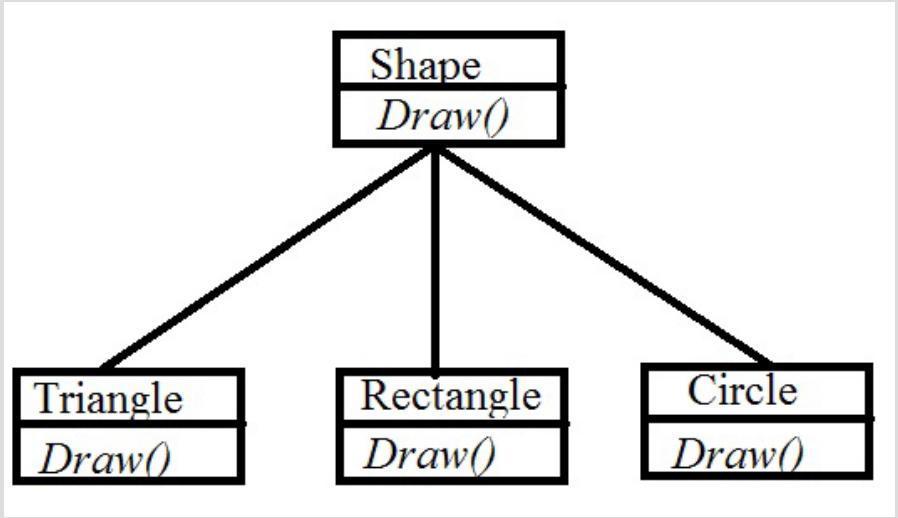
```php
<?php
class Clovek {
    // definice atributů
    var $jmeno;
    var $prijmeni;
    var $vek;

    // definice metody
    function vratVek() {
        return $this->vek;
    }
}
?>
```

Přednáška 3: Pokročilé PHP

OOP (5/5)

Polymorfismus
Polymorfismus umožňuje používat **jednotné rozhraní pro práci s různými typy objektů**.
Dle příkladu vpravo můžeme napsat metodu **speak()**, kterou budou dědit všechny konkrétní objekty zvířat z obecné třídy **Zvire**.
Tato metoda pak může být v každém objektu (**Pes**, **Kocka**, atd.) implementována jinak, např. mít jiný výstup.



Podstatou polymorfismu je tedy metoda nebo metody, které mají všichni potomci definované se stejnou hlavičkou, ale jiným tělem.