

Jazyk PHP

- Charakteristika, syntaxe
- Práce s formuláři – metoda GET, metoda POST
- Práce se soubory – uveďte na příkladech

PHP

PHP (*Personal Home Page – původně, Hypertext Preprocessor*)

Nejoblíbenější skriptovací jazyk na straně serveru v oblasti webového vývoje – běží na 79 % všech webových stránek (Facebook, Wikipedia,).

Vytvořil ho Rasmus Lerdorf v roce 1995.

Hlavním cílem jazyka PHP je zpracovávat data tak, aby je bylo možné dynamicky zobrazovat do webových stránek. Umožňuje vývojářům vylepšit statické stránky o reakce na uživatelské požadavky, ukládat načtená data do databáze a později je uživatelům vypisovat.

Využití jazyka: programování dynamických internetových stránek a webových aplikací, vytvoření šablony webu, kniha návštěv, ankety, počítadlo přístupů, spolupráce s MySQL atd.

PHP skripty se běžně píší v textovém editoru (např. PSPad) a ukládají se na webový server do souborů s příponou **.php**.

Při zpracování souboru zpracuje interpreter PHP jen to, co se nachází mezi speciálními značkami **<?php ?>**.

Zbytek jednoduše ignoruje (pošle to na výstup tak, jak to je). To umožňuje jednoduše kombinovat kód PHP a HTML.

<?php echo „Toto je text v PHP "; **?>**

Příkazy se oddělují středníkem a pro lepší přehlednost píšeme každý příkaz na samostatný řádek.

Pro lepší orientaci ve zdrojovém kódu slouží komentáře:

//jednořádkový komentář

/*komentář psaný

na více řádků*/

to co je „zakomentované“, překladač přeskočí,
pro vypsání na obrazovku používáme příkaz echo.

Proměnné

- Proměnné nemají pevně definován datový typ.
- Datový typ je určen obsahem proměnné.
- Automaticky se může měnit na jiný typ, existují funkce pro zjištění typu i přetypování proměnné
- **Základní datové typy:**

Integer – celé číslo

Float (double) – desetinné číslo s plovoucí čárkou

Boolean – pravdivostní hodnota (může být pouze true, nebo false)

String – řetězec znaků libovolné délky.

Array – pole, počáteční index je 0 pokud se neurčí jinak

Object – objekt

- proměnná začíná znakem \$ (pravý Alt+ů)
- **pozor** - u příkazů na velikosti písmen nezáleží (echo a ECHO je to samé), u proměnných naopak záleží (\$promenna a \$Promenna jsou dvě různé).
- nesmí začínat číslem, nesmí obsahovat mezeru a je lepší psát bez diakritiky
- více slov se spojí podtržítkem
- hodnotu do proměnné přiřazujeme pomocí =
- je-li hodnota proměnné text, píšeme ji do uvozovek pokud je hodnota číslo, píšeme bez uvozovek a s desetinnou tečkou

```
$pozdrav = "Ahoj";
```

```
$vek = 15;
```

```
$vyska = 1.84;
```

Příklady názvů proměnných:

\$pocet = 1; // správně

\$Pocet = 1; // toto je jiná proměnná než ta předchozí

\$počet = 1; // nezpůsobí chybu, ale použití diakritiky není vhodné

\$4.slovo = "slovo"; // chyba, název nemůže obsahovat tečku ani začínat číslicí

\$slovo4 = "slovo"; // správně

\$pocet slov = 4; // chyba, název nemůže obsahovat mezeru

\$pocetSlov = 4; // správně

Inicializace proměnné

```
$pozdrav = "Ahoj";
```

Použití hodnoty proměnné

```
echo $pozdrav;
```

Zničení proměnné

```
unset($pozdrav);
```

použití je spíše výjimečné

PHP je zruší samo nejpozději po skončení skriptu.

Vypsání hodnoty a typu proměnné

funkce **var_dump()**

Funkci lze předat proměnnou a na výstup se vypíše datový typ a hodnota.

Kontrola existence proměnné

Příkaz **isset()** vrátí true v případě, že proměnná byla inicializovaná a nebyla dosud zrušená, jinak řečeno není NULL. V opačném případě vrátí false.

Příkaz **empty()** má obrácenou logiku.

```
$pozdrav = "Ahoj!";
```

```
var_dump($pozdrav);
```

```
Vypíše: string(5) "Ahoj!"
```

string je datový typ, u řetězce je v závorce jeho velikost v **bajtech** a pak následuje výpis hodnoty

```
$pozdrav = "Čau";
```

```
var_dump($pozdrav);
```

```
string(4) "Čau"    („Č“ jsou v UTF-8 dva bajty)
```

Konstanty v PHP

- ✓ Konstanta slouží k pojmenování hodnoty, na kterou se budeme při provádění skriptu odkazovat. Hodnotu konstanty nelze za běhu měnit.

```
define ("jmeno_konstanty", "hodnota");
```

Číselné hodnoty označíme výstižnějším názvem, který lépe charakterizuje jejich význam.

```
define ("pi", "3.14");
```

```
define ("max", "100");
```

Operátory

- . spojování řetězců
- + sčítání
- odčítání
- * násobení
- / dělení čísel (racionálních i celých)
- % zbytek po dělení (modulo)

Aritmetické operátory se vykonávají zleva doprava a dle jejich priority

Operátory přiřazení

= přiřazení

.= připojení řetězce k téže proměnné

+= přičtení čísla k téže proměnné

-= odečtení čísla k téže proměnné

*=

/=

%=

Na rozdíl od aritmetických operátorů se operátory přiřazení vykonávají zprava doleva a v závislosti na jejich prioritě.

++	přičtení jedničky
--	odečtení jedničky
&&	a zároveň
	nebo
==	rovnost
!=	nerovnost
<>	nerovnost
<	je menší
>	je větší
<=	je menší nebo rovno
>=	větší nebo rovno

```
c = c + 1;  
c += 1;  
c++;
```

```
$cislo = 5;  
$cislo += 10;  
$cislo %= 2;
```

```
$a = 1;  
$b = ++$a  
Rozdíl!!!  
$a = 1  
$b = $a++;
```

Řetězce

- Řetězec je text, uzavřený v uvozovkách nebo apostrofech.
- Takto napsaný text můžeme buď přiřadit nějaké proměnné nebo jej nechat vypsát pomocí příkazu **echo**.
- Pokud napíšeme textový řetězec do apostrofů (jednoduchých uvozovek), vloží se do něj přesně ten text, který je uveden.
- Pomocí uvozovek však můžeme do řetězce zapisovat speciální znaky a vkládat proměnné.
- Speciální znaky se označují zpětným lomítkem.
vlození uvozovky \", dolaru \\$ a samotného zpětného lomítka \\
- Pro spojování řetězců se v PHP používá operátor . (tečka).

```
<?php
    $obsah = "25"."85"."8";
    echo $obsah;
?>
```

```
<?php
    $jmeno = "Honza";
    $prijmeni = "Novák";
    $celejmeno = $jmeno." ".$prijmeni;
    echo $celejmeno;
?>
```

Pole

Umožňuje uchovávat více hodnot (prvků pole), které spolu souvisí.

Každá hodnota je dostupná pomocí indexu, zapsaného v hranaté závorce.

Pokud není uvedeno jinak, pole se indexuje od nuly.

Jednotlivé hodnoty uložené v poli mohou být libovolného typu.

\$pole[0] = 3.14;	// reálné číslo
\$pole[1] = 77;	// celé číslo
\$pole[2] = "Toto je nějaký text...";	// textový řetězec

Pokud u pole použijeme prázdný index, číslování indexů je nastaveno automaticky od nuly.

Další zvláštnost je, že v jazyce PHP lze libovolně přidávat další prvky do pole.

Na rozdíl od jiných jazyků je v tomto jazyce možno vytvářet nejen číselné indexy, ale i textové (tzv. asociativní pole).

Pole může být i vícerozměrné přidáním dalších indexů.

Inicializaci pole lze provést také zjednodušeně pomocí funkce `array()`.

Počet prvků v poli lze zjistit pomocí funkce `count()`.

```
$pole = array(1, 2, 3, 4, 5);
```

- Přidání dalšího prvku - automaticky přidělený index

```
$pole[] = 6;
```

ručně zadaný index

```
$pole[10] = 7;
```

Automatické vložení pokračuje od nejvyšší hodnoty indexu.

```
$pole[] = 8;
```

- Výpis obsahu pole na obrazovku příkazem:

```
print_r($pole);
```

```
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 [10] => 7 [11]  
=> 8 )
```

Takovéto pole nelze procházet cyklem.

Textové indexy

```
$pole = array("po" => "pondělí", "ut" => "úterý",  
"st" => "středa", "ct" => "čtvrtek", "pa" =>  
"pátek", "so" => "sobota", "ne" => "neděle");
```

```
echo $pole["ut"];      Výstupem je: úterý
```

Pokud používáme textové indexy a poté začneme používat indexy číselné, automaticky se číslují od pozice 0.

Vícerozměrné pole

```
$pole = array(1, 2, 3, array(10, 20, 30), 4);
```

Na indexu číslo 3 leží celé pole, ve kterém je na indexu 0 uloženo 10, dále 20 a 30. Toto pole končí a v původním poli je na indexu 4 uloženo číslo 4.

Výpis hodnoty 20: `echo $pole[3][1];`

Podmínka

Podmínka se v programování značí slovem if, její syntaxe je:

if(logický výraz){ co se má stát }

nebo

if(logický výraz){ tělo podmínky }

else{ tělo větve else, neboli co se má stát, pokud není splněná podmínka }

Vyhodnocení podmínky musí vrátet logickou hodnotu:

TRUE – „pravda“, neboli podmínka je splněna

FALSE – „nepravda“, neboli podmínka není splněna

✓ Jedná se především o porovnávání dvou hodnot.

✓ Je-li podmínka splněna (**TRUE**), vykoná se blok příkazů části **IF**

Logické operátory

Rozhodovacích podmínek lze stanovit i více, spojují se pomocí logických operátorů, které určují pravdivost či nepravdivost celého výroku:

Operátor	Význam	Pravda když
and	logický součin	všechny podmínky jsou pravdivé
&&		
or	logický součet	alespoň jedna podmínka je pravdivá
 		
!	negace	pokud je podmínka nepravdivá
XOR	neekvivalence	pouze jedna podmínka je pravdivá

Logický součet

Platí, že true je vždy, pokud v součtu je alespoň jedna hodnota true.

Zápis logického součtu je dvojicí svislítek

Platí tedy:

`true || true = true;`

`true || false = true;`

`false || true = true;`

`false || false = false;`

Konstrukce podmínky příkazu IF

```
if ((podmínka 1) or (podmínka 2) or (podmínka N))
```

```
{
```

Tyto příkazy se provedou, pokud je splněna (**TRUE**) **minimálně jedna** ze stanovených podmínek.

```
}
```

```
else
```

```
{
```

Provede se, pokud není splněna žádná ze stanovených podmínek.

```
}
```

Logický součin

Platí, že true je pouze, pokud všechny hodnoty v součinu jsou true, neboli ani jedna hodnota není false.

Zápis logického součinu je dvojicí ampersandů

Platí tedy:

`true && true = true;`

`true && false = false;`

`false && true = false;`

`false && false = false;`

Konstrukce podmínky příkazu IF

```
if ((podmínka 1) and (podmínka 2) and (podmínka N))
```

```
{
```

Tyto příkazy se provedou, pokud jsou splněny (**TRUE**) **všechny** stanovené podmínky.

```
}
```

```
else
```

```
{
```

Provede se, pokud minimálně jedna z podmínek není splněna.

```
}
```

Porovnání čísel

Při porovnání čísel můžeme použít operace:

- Menší <
- Větší >
- Menší nebo rovno <=
- Větší nebo rovno >=
- Je rovno ==
- Není rovno !=

Příklad:

Mějme dvě proměnné x a y obsahující čísla 5 a 6.

Pokud je x a y větší než 0, necháme PHP vypsát slovo „ano“.

```
<?php
```

```
$x = 5;
```

```
$y = 6;
```

```
if($x > 0 && $y > 0)
```

```
{ echo "ano"; } ?>
```

Příkaz SWITCH – CASE

```
switch ($stat)
{
case "cz":
    echo "Česká republika";
break;
case "svk":
    echo "Slovenská republika";
break;
case "pl":
    echo "Polsko";
break;
default:
    echo "Nelze identifikovat stát!";
}
```

Umožňuje testovat podmínku na více hodnot.

Zjistí se hodnota proměnné **\$stat**

1. Prochází se jednotlivé větve (**case**) a jakmile se narazí na vyhovující podmínku, provede se blok kódu až k příkazu **break**;
2. Větev **default** se provede, pokud nevyhoví žádná podmínka (obdoba ELSE).

Příkazy pro tvorbu cyklů

Cykly jsou určeny k opakovanému provádění bloku kódu v zadaném počtu nebo do splnění podmínky. V PHP jsou dva základní druhy cyklů:

- Cyklus **WHILE**

Testuje podmínku před nebo po každé iteraci (kroku) a znovu se provede pouze v případě, že je splněna stanovená podmínka.

- Cyklus **FOR**

Počet iterací (počet opakování bloku kódu) je přesně stanoven před prvním provedením (v definici příkazu FOR).

Příklady použití cyklů

- procházení hodnot proměnné typu „pole“,
- inicializace proměnné typu „pole“,
- zpracování výsledku dotazu do databáze,
- vytváření položek rozbalovacího menu (formulářový prvek),
- výpis řady po sobě jdoucích číselných či textových hodnot,
- opakované provedení určitého bloku kódu,
- apod.

Cyklus WHILE

Nejjednodušší cyklus

Testuje na začátku podmínku

Nemusí se spustit ani jednou

Lehko se zacyklí

Jedná o cyklus s podmínkou na začátku.

while = dokud je splněna podmínka, prováděj blok příkazů cyklu.

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

Výpis čísel od 1 do 10

```
<?php
```

```
$x = 1;
```

```
while($x <= 10)
```

```
{ echo $x." "; $x++; }
```

```
?>
```

Výstup:

1 2 3 4 5 6 7 8 9 10

Spočítá sumu čísel od 1 do určeného čísla

```
$cislo = 10;
```

```
$soucet = 0;
```

```
// Kopie původního čísla pro výpis na konci
```

```
$puvodni = $cislo;
```

```
// Dokud číslo je větší než 0
```

```
while ($cislo > 0)
```

```
{ $soucet = $soucet + $cislo;
```

```
    $cislo = $cislo - 1;
```

```
}
```

```
echo "Suma čísel od 1 do $puvodni je: $soucet";
```

Cyklus FOR

- Zápis tohoto cyklu je složitější
- Nejčastěji se používá k přesnému počtu spuštění, nebo k danému kroku u proměnných (například výpočet některé matematické funkce pro x od 0 do 1 s krokem 0.01 apod.)
- Testuje na začátku podmínku
- Nemusí se spustit ani jednou
- Zacyklení lze provést jednoduše úmyslně, nebo chybnou ukončovací podmínkou

Cyklus FOR

Cyklus začíná názvem **for**.

Poté následuje v kulatých závorkách trojice hodnot, oddělených středníky.

První značí počáteční inicializaci proměnné (nejčastěji přes kterou procházíme).

Druhá obsahuje ukončovací podmínku, kterou testujeme na začátku.

Třetí hodnota značí změnu na konci každé iterace.

Po těchto hodnotách přichází samotné tělo cyklu, uvedené ve složených závorkách.

Cyklus FOR

Jedná o cyklus s přesně daným počtem kroků (opakování bloku příkazů).

```
for (výraz1; výraz2; výraz3)
{
    příkazy;
}
```

Výpis čísel od 1 do deseti

```
<?php
```

```
for($i = 1; $i <= 10; $i++)
```

```
{ echo $i." "; }
```

```
?>
```

Výstup:

1 2 3 4 5 6 7 8 9 10

výraz1

Proveden pouze jednou na začátku cyklu. Inicializace řídicí proměnné cyklu na počáteční hodnotu.

výraz2

Vyhodnocen na začátku každého opakování cyklu. Testování stanovené podmínky.

výraz3

Vyhodnocen na konci každého opakování cyklu. Změna hodnoty řídicí proměnné.

Násobky dvou od 2 do 100

```
for ($i = 2; $i <= 100; $i += 2)
{ echo $i . '<br>'; }
```

Procházení pole

```
$cisla = array(10,8,22,7,11,6);
for($i=0; $i<count($cisla); $i++)
{ echo $cisla[$i]; }
```

Vnořování cyklů

```
$iterace = 0;  
for ($i = 0; $i < 20; ++$i)  
{  
    for ($j = 0; $j < 5; ++$j)  
    {  
        echo "Právě se provádí $iterace.<br>";  
    }  
}
```

Provede se **20x** vnitřní cyklus s řídicí proměnnou **\$j**. Textový řetězec bude pod sebou vypsán **100x**. Vhodné např. pro práci s dvourozměrným polem.

Cyklus DO – WHILE

Jedná o cyklus s podmínkou na konci.

do

{

 příkazy;

}

while (**podmínka**);

Pravdivost výrazu se testuje na konci každé iterace namísto jejího začátku. První iterace cyklu **do...while** se provede vždy (pravdivost výrazu je testována až na konci iterace), což u cyklu **while** není zaručeno.

Cyklus FOREACH

Používá se pro procházení všech prvků proměnné typu „pole“.

Zjednodušuje příkaz **FOR**. Název příkazu je odvozen z anglických slov:

FOR – “pro”

EACH – “každý”

AS – “jako”

```
foreach ($nazev_pole as $hodnota)
{
    echo "$hodnota <br>";
}
```

```
$RajskySalat = array("rajče", "cibule", "sůl", "cukr", "pepř");
```

```
foreach ($RajskySalat as $ingredience)
```

```
    echo "$ingredience <br>";
```

Vypíše hodnoty všech prvků pole **\$nazev_pole**.

Break a continue

- Oba tyto příkazy jsou tzv. řídicí
- Lze s nimi řídit běh cyklů bez závislosti na ukončující podmínce
- Praxi se používají zejména při vzniku chyb, nebo jako reakce na neočekávaný běh

Příkaz break - okamžitě ukončí běh celého cyklu.

Cokoliv, co je od této pozice v těle cyklu se již nevykoná.

Příkaz continue - pouze přeskočí danou iteraci.

Cyklus poté probíhá podle svého klasického průběhu, jako kdyby bylo dokončeno tělo.

Znamená to tedy, že continue přeskočí všechny další instrukce v těle cyklu do místa, kde je umístěno. Poté cyklus standardně vyhodnotí, zda se smí spustit znovu a pokud ano, spustí se.

Formuláře

- je část stránky, která se běžnému uživateli jeví podobně jako dialogové okno
- slouží k získávání informací od uživatele
- získané informace jsou následně odeslány na server ke zpracování
- ke zpracování údajů z formuláře se používají CGI-skripty
- takže musíme vytvořit stránku s formulářem a ještě skript, který zaslaná data správně zpracuje
- formulář může obsahovat různá vstupní pole, tlačítka a přepínače
- formulářů může být na jedné stránce víc, ale nesmí být do sebe vnořeny

Formuláře

Formuláře slouží k zadávání dat uživatelem webových stránek.

Protože formulář slouží k posílání dat, má značka `<form>` několik atributů, které slouží právě k určení, kam se data z formuláře mají poslat, a jakým způsobem.

Atribut **action**, který určuje, kam se budou data posílat.

Atribut **method**, který určuje, jakým způsobem se budou data posílat.

Za atribut **action** se dosazuje adresa PHP skriptu, který přijme zasláná data.

Za atribut **method** se dosazuje nejčastěji **get**, nebo **post**.

Základem je HTML formulář

Metoda GET

- Výchozí metoda všech formulářů. Všechna data formuláře jsou zaslána obslužnému skriptu prostřednictvím URL adresy.
- Metodu lze využít při navigaci v rámci webové stránky a při předávání parametrů stránky.
- Data jsou vidět v adresním řádku (návštěvník je vidí). Proto by se měla metoda GET používat pro data krátká a pro data u kterých nevadí, že si je návštěvník přečte, případně je v adresním řádku změní.
- Metoda GET je vhodná pouze pro zasílání textových řetězců, **nelze použít pro odesílání souborů nebo velkého objemu hodnot !!!**

Metoda POST

- Metoda je určena pro odesílání většího objemu dat (velké množství textových hodnot, datové soubory různého typu, apod.) v přímo v těle HTTP požadavku webovému serveru.
- Přenášená data **nejsou součástí** URL adresy, vhodné pro přenos citlivých údajů (např. login, heslo, bezpečnostní kód, apod.).
- Metoda POST vždy odesílá nový HTTP požadavek webovému serveru. Nehledá se tedy stránka uložená v mezipaměti (cache) prohlížeče na lokálním disku, ale vždy se načte nová stránka z webového serveru.

Základem je HTML formulář.

Pro začátek předpokládejme jednoduchý formulář pro zadání jména a příjmení uživatele:

```
<form action="vypis.php" method="GET">
```

```
    Jméno: <input name="jmeno" type="text"><br>
```

```
    Příjmení: <input name="prijmeni" type="text"><br>
```

```
    <input type="submit" value="Odeslat">
```

```
</form>
```

Zobrazí se textové pole pro zadání jména, příjmení a tlačítko pro odeslání. Po kliknutí na tlačítko se obsah pole odešle na script vypis.php.

Metoda GET

PHP automaticky dekoduje data z URL adresy a uloží každou dvojici **promenna=hodnota** jako proměnnou s odpovídajícím názvem do pole **\$_GET ["jmeno_formularoveho_prvku"]**

Obslužný skript: **vypis.php**

```
<?php
```

```
    echo "Vaše jméno: " . $_GET["jmeno"] . "<br>";
```

```
    echo "Vaše příjmení: " . $_GET["prijmeni"];
```

```
?>
```

Metoda POST

PHP dekóduje data z HTTP požadavku webovému serveru a uloží je jako proměnnou s odpovídajícím názvem do pole:

`$_POST["jmeno_formularoveho_prvku"]`

Obslužný skript: **vypis.php**

```
<?php
```

```
    echo "Vaše jméno: " . $_POST["jmeno"] . "<br>";
```

```
    echo "Vaše příjmení: " . $_POST["prijmeni"];
```

```
?>
```

Vytvoření souboru

metoda `fopen("soubor", "w+")`

`$soubor=fopen("soubor.txt", "w+,,)` režim otvírání souborů

- a Otevře soubor pro přidání, pokud soubor neexistuje PHP jej vytvoří
- a+ Otevře soubor pro přidávání a čtení, pokud soubor neexistuje PHP jej vytvoří
- r Otevře soubor pouze pro čtení
- r+ Otevře soubor pro čtení a zápis
- w Otevře soubor pro zápis, původní data budou ztracena. Pokud soubor neexistuje PHP jej vytvoří
- w+ Otevře soubor pro zápis a čtení, původní data budou ztracena, pokud soubor neexistuje PHP jej vytvoří

Otevření souboru

Otevření souboru se provádí stejnou funkcí jako vytvoření tj.

`fopen("soubor","režim").`

`$soubor=fopen("soubor.txt", "r");`

Tímto jsme otevřeli soubor.txt pro čtení.

Zavření souboru

Po dokončení práce se souborem je rozumné jej zavřít pomocí metody

`fclose(id_souboru)`

`$soubor=fopen("soubor.txt", "r");`

`fclose($soubor);`

Vypsání obsahu

Otevřeme soubor a vypíšeme jeho obsah pomocí metody `fread(id_souboru, počet_znaků)`:

```
$soubor = fopen("soubor.txt", "r");  
$text = fread($soubor, 10);  
echo($text);  
fclose($soubor);
```

Metoda `fread()` přečte ze souboru prvních 10 znaků.

Obvykle je třeba načíst celý soubor, je tedy nutné zjistit počet jeho znaků.

To lze pomocí metody `filesize($soubor)`, která vrátí počet znaků vybraného souboru.

Čtení ze souboru

fgets()

čte jeden řádek otevřeného souboru.

Pokud se volá opakovaně, čte vždy další řádek souboru.

```
<?php
```

```
$soubor = fopen("./text.txt", "r");
```

```
$prvni=fgets($soubor); //prvni radek
```

```
$druhy=fgets($soubor); // druhý radek
```

```
fclose($soubor);
```

```
echo "První řádek: $prvni <br>";
```

```
echo "Druhý řádek: $druhy";
```

```
?>
```

Uložení souboru

Pomocí metody `fwrite(id_souboru, "text")` zapíšeme text na konec vybraného souboru:

```
$text = "cokoli, co chcete uložit do souboru";  
$soubor = fopen("soubor.txt", "a+");  
fwrite($soubor, $text);  
fclose($soubor);
```

Soubor.txt může být existující soubor, pokud neexistuje PHP ho vytvoří.

Kopírování

`copy(zdroj, cíl):`

Přejmenování

`rename("původní název", "nový název")`

`rename("soubor.txt", "soubor.html")`

Převod textového souboru na soubor html.

Odstranění

`unlink(název souboru)`

`unlink("soubor.txt");`

Příkazy pro adresáře

`mkdir()` – Vytvoří adresář

`is_dir()` – Zjistí, jestli je na zadané cestě adresář

`rmdir()` – Smaže prázdný adresář

`rename()` – Přejmenuje/přesune soubor/adresář

Struktura adresáře

`opendir()` – Otevře celý adresář (složku)

`readdir()` – Přečte celý adresář (složku)

Příkazy pro vkládání obsahu souborů

- ✓ PHP obsahuje příkazy **require** a **include**, které načítají a provádí kód ze zadaného externího souboru.
- ✓ Příkazy jsou totožné, liší se pouze v případě, kdy vkládaný soubor není nalezen:
 - require** - skončí s chybou (Fatal Error) a zastaví se zpracování stránky.
 - include** – vypíše varování (Warning) a skript se pokusí běžet dál.
- ✓ Oba příkazy očekávají, že vkládané soubory jsou v kódu HTML, nikoliv PHP. Je-li vkládaný soubor skript PHP, musí být ohraničen PHP tagy.

Syntaxe příkazů require a include

Syntaxe obou příkazů je shodná. Název souboru a cesta k němu se zapíše do uvozovek nebo apostrofů. Ukázka příkazu **include**:

1. Soubor je ve stejném adresáři jako stránka obsahující příkaz:

```
include "soubor.php";
```

2. Soubor je v podadresáři:

```
include "adresar/soubor.php";
```

3. Soubor je v nadřazeném adresáři:

```
include "../soubor.php";
```

Příklady použití require a include

Příkazy umožňují vytvářet přehlednější kód webových stránek, který je jednodušší na správu. Změna v externím souboru se projeví ve všech stránkách (souborech), kde byl obsah souboru vložen.

Použitelné zejména pro:

- ✓ vytváření / uzavírání spojení s databází,
- ✓ vkládání uživatelských funkcí, proměnných nebo konstant,
- ✓ vkládání uživatelských menu, hlaviček stránky apod.,
- ✓ vkládání formulářů nebo jeho částí,
- ✓ vkládání textů, apod.

Tvorba jednoduchých skriptů

Počítadlo návštěv

Tvorba hesla

Tvorba jednoduché kalkulačky

Načtení obsahu souboru s akt. datem

Generování obsahu, formuláře