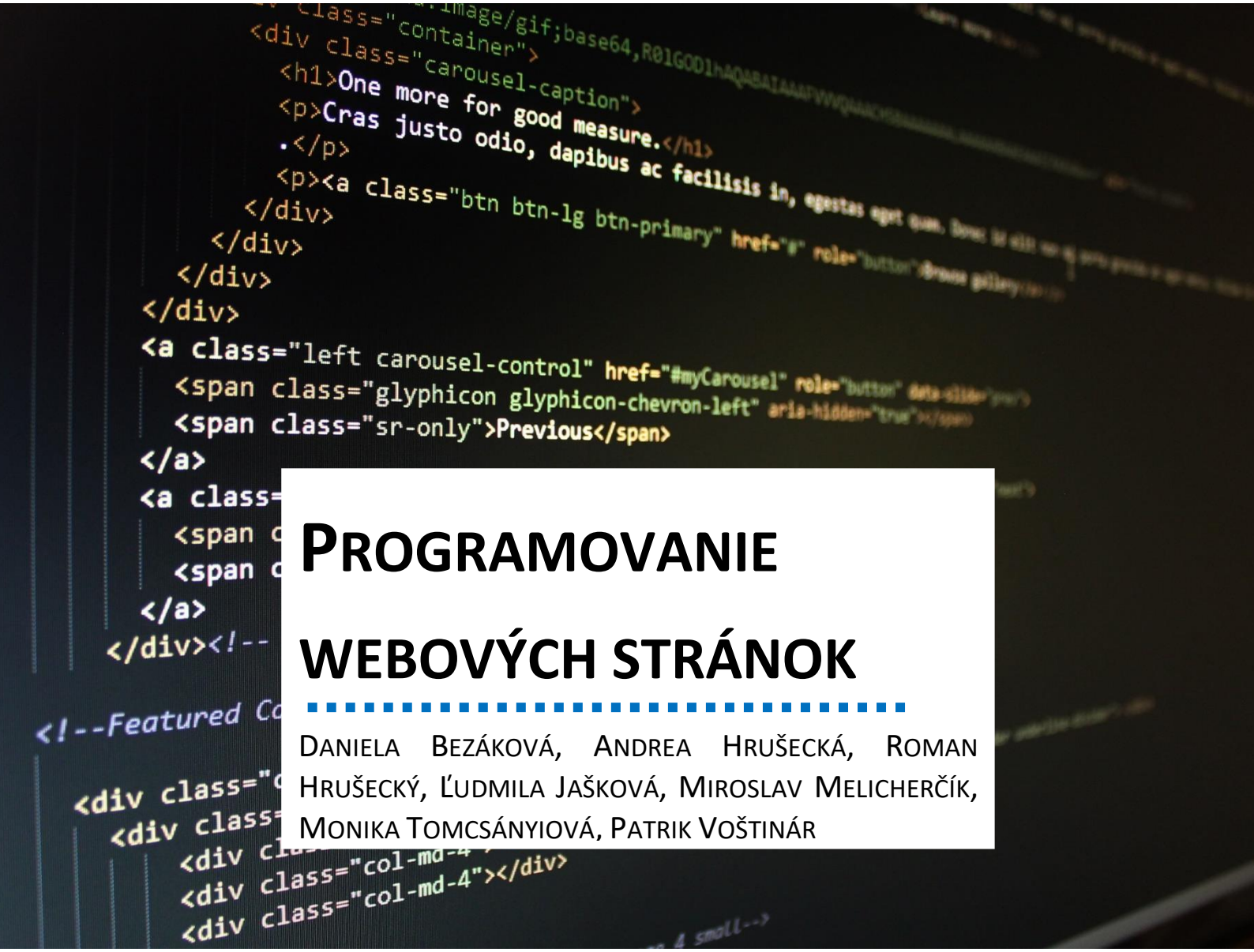


PROGRAMOVANIE WEBOVÝCH STRÁNOK

DANIELA BEZÁKOVÁ, ANDREA HRUŠECKÁ, ROMAN HRUŠECKÝ, ĽUDMILA JAŠKOVÁ, MIROSLAV MELICHERČÍK, MONIKA TOMCSÁNYIOVÁ, PATRIK VOŠTINÁR

EURÓPSKA ÚNIA
Európsky sociálny fond
Európsky fond regionálneho rozvoja

OPERAČNÝ PROGRAM ĽUDSKÉ ZDROJE

MINISTERSTVO ŠKOLSTVA, VEDY, VÝSKUMU A ŠPORTU SLOVENSKEJ REPUBLIKY

IT AKADÉMIA

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu v rámci Operačného programu Ľudské zdroje

www.minedu.sk www.employment.gov.sk/sk/esf/ www.itakademia.sk

Programovanie webových stránok

Spracované v rámci národného projektu IT Akadémia – vzdelávanie pre 21. storočie

Bratislava 2020

Programovanie webových stránok

Spracované s finančnou podporou národného projektu IT Akadémia – vzdelávanie pre 21. storočie

Autori: PaedDr. Daniela Bezáková, PhD., PaedDr. Andrea Hrušecká, PhD., PaedDr. Roman Hrušecký, PhD., doc. RNDr. Ľudmila Jašková, PhD., RNDr. Miroslav Melicherčík, PhD., doc. PaedDr. Monika Tomcsányiová, PhD., PaedDr. Patrik Voštinár, PhD.

Recenzenti: Mgr. Zuzana Lučaiová, PaedDr. Miroslav Ölvecký, PhD., doc. RNDr. Ľubomír Salanci, PhD.

Neprešlo jazykovou úpravou.

Vydavateľ: Centrum vedecko-technických informácií SR, Bratislava

Rok vydania: 2020

Vydanie : 1. vydanie

ISBN: 978-80-89965-68-7 EAN 9788089965687

Bratislava 2020

Tento projekt sa realizuje vďaka podpore z Európskeho sociálneho fondu v rámci Operačného programu Ľudské zdroje.

OBSAH

1 DYNAMICKÉ WEBOVÉ STRÁNKY

Vytvárať webové stránky len pomocou HTML, presnejšie písať celý obsah do zdrojového HTML súboru (tzv. statické stránky), nie je vždy to správne, resp. takým spôsobom niektoré stránky nevieme vytvoriť. Napríklad stránky pre internet banking, internetové obchody, rôzne diskusné fóra takýmto spôsobom vytvoriť nevieme, lebo nevieme aktuálny obsah. Ten sa totiž mení. Pri takýchto stránkach nám pomôže ich dynamická tvorba.

Rozdiel medzi statickými (HTML) a dynamickými stránkami je veľký. Pri statických stránkach sa na serveri nachádza finálna „verzia" stránky, teda to, čo je uložené na serveri (súbor) sa presne zobrazí klientovi vo webovom prehliadači. **Dynamická stránka** však vzniká (generuje sa) až pri požiadavke klienta o jej zobrazenie. Vtedy sa vykonajú všetky potrebné činnosti a vygeneruje sa HTML kód výslednej stránky, ktorá sa zobrazí vo webovom prehliadači.

Dynamické stránky rozdeľujeme na 2 typy podľa toho, kde sa dynamická časť stránky generuje - či na serveri alebo v prehliadači. To, čo sa vykonáva v prehliadači, si vieme čiastočne pozrieť v zdrojovom kóde. Čo sa so stránkou deje na serveri, to nevieme zistiť. Na serveri sa totiž spustí kód programu a až jeho výsledok sa nám zobrazí v prehliadači a my nedokážeme zistiť, čo sa udialo - či sa vykonali nejaké funkcie, cykly, či sa vložili nejaké súbory atď. Toto všetko je pred nami (ako používateľmi) skryté.

Prečo tvoriť dynamické stránky? Povedzme, že by sme chceli:

1) vypísať dni v mesiaci (čísla 1 až 31) v rámci kalendára na stránke,
2) zobraziť na stránke nejakú akciu len v presne stanovenom období (dátum, čas),
3) vypísať zoznam mien, ktoré máme uložené napr. v poli/zozname alebo inej dátovej štruktúre,
4) vypísať zoznam článkov, ktoré niekto pripravil/napísal a sú uložené napr. v súbore alebo v poli/zozname,
5) vypísať zoznam najbližších predstavení nejakého kina na viacerých stránkach,
6) spracovať formulár (napr. komentovať, objednať si, rezervovať si, poslať správu správcovi stránky,...),
7) načítať údaje zo súboru alebo databázy, filtrovať ich a potom zobraziť na stránke,
8) uložiť údaje do súboru prípadne databázy.

Riešiť predchádzajúce problémy pomocou statickej webovej stránky by bolo neefektívne a vo väčšine prípadov dokonca nemožné. Napr. v prípade (1) by to znamenalo ručne vypísať do zdrojového kódu stránky 31 čísel – (tabuľka 1.1 vľavo), v prípade (2) nahrať stránku na server v presne stanovenom čase.

Pri riešení týchto úloh nám môžu pomôcť dynamické webové stránky - môžeme v nich využiť výpis čísel pomocou cyklu, výpis výsledkov výpočtov, výpis obsahu poľa/zoznamu prípadne súboru, načítanie údajov zo súboru alebo databázy, prípadne zápis údajov do súboru alebo databázy. Na tvorbu dynamických webových stránok môžeme využiť rôzne programovacie jazyky (napr. PHP, C#, ASP.NET, Java, JavaScript, Python...). Každý má svoje výhody aj nevýhody (aj fanúšikov a odporcov :-)).

Tabuľka 1.1 Príklad použitia PHP skriptu na generovanie položiek výberovej ponuky.

`<option value='1'>1</option>`
`<option value='2'>2</option>`
`<option value='3'>3</option>`
`...`
`<option value='30'>30</option>`
`<option value='31'>31</option>`
	`for ($i = 1; $i <= 31; $i++) {`
	`  echo "<option value='$i'>$i`
	`</option>\n";`
	`}`

Výpis čísel len v HTML	*Výpis čísel pomocou PHP*

V prvej časti učebnice sa budeme venovať tvorbe dynamických webových stránok na strane servera a stránky budeme vytvárať pomocou programovacieho jazyka **PHP**, preto ich budeme nazývať PHP stránky. PHP vie pracovať s rôznymi súbormi, komunikovať s rôznymi databázami, pracovať s poštovými službami. Tento jazyk je veľmi populárny, voľne dostupný, používajú ho aj veľké systémy, resp. spoločnosti ako napr. Facebook, Wikipédia, WordPress.

Druhá časť učebnice bude venovaná tvorbe dynamických webových stránok na strane klienta pomocou programovacieho jazyka **JavaScript**.

1.1 Jazyk PHP, PHP stránky

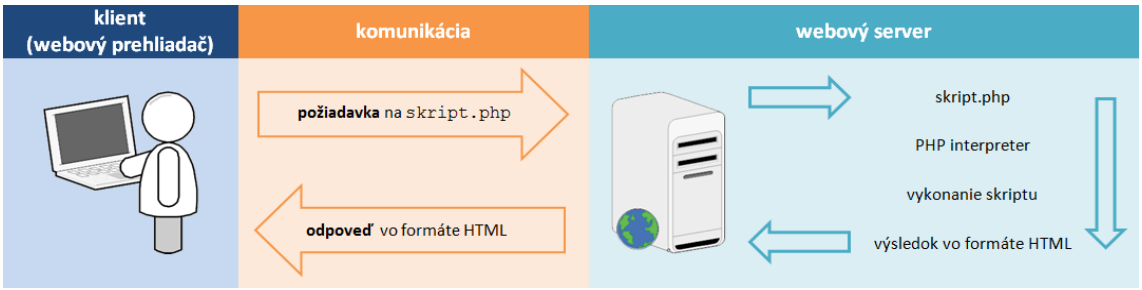
Webový server je program, ktorý spracováva tzv. HTTP požiadavky zvyčajne odoslaných z webového prehliadača. Po spracovaní požiadavky vráti webovému prehliadaču odpoveď, napr. webovú stránku, obrázok, čistý text...

K tvorbe a prezeraniu statických webových stránok nepotrebujeme okrem editora stránok a webového prehliadača nič iné. Naproti tomu, k zobrazeniu a testovaniu dynamických webových stránok na strane servera potrebujeme niekoľko vecí – webový server (**Apache**, IIS, ...), programovací jazyk na strane servera (**PHP**, ASP, Python, ...) a vo väčšine prípadov aj databázový server na ukladanie údajov (**MySQL**, MS SQL, SQLite, PostgreSQL, Oracle, ...). Dnešné webové aplikácie prakticky neexistujú bez databáz. Spolupráce PHP s databázami sa v našej učebnici dotkneme len okrajovo.

Čo sa deje s PHP stránkami?

Ako sa spracováva (zobrazuje) PHP stránka?

1) Používateľ zadá adresu PHP stránky do webového prehliadača.
2) Webový server (napr. Apache) rozpozná PHP a pošle PHP súbor tzv. PHP interpreteru.
3) PHP interpreter vykoná PHP kód v súbore a výsledok pošle späť webovému serveru.
4) Webový server pošle výsledok (výslednú stránku) webovému prehliadaču a ten ju zobrazí používateľovi.



Stručne o jazyku PHP

PHP[1] je **skriptovací jazyk** s otvoreným kódom na strane servera. Skriptovací jazyk znamená, že program v ňom vytvorený sa nevykonáva samostatne, ale vykonáva ho tzv. interpreter (vykonávač). V prípade PHP je to PHP interpreter, ktorý musí byť súčasťou webového servera. Program v skriptovacom jazyku sa nazýva skript.

ZAPAMÄTAJTE SI

PHP súbory nemôžeme len tak otvoriť vo webovom prehliadači, ale musíme ich spustiť (vykonať) pomocou PHP interpretera cez webový server.

Jazyk PHP je netypový jazyk. V skutočnosti za každou premennou nejaký typ je (automaticky generovaný jazykom PHP), ale pri vytváraní premennej ho nemusíme nijako definovať.

Súbory s PHP skriptami majú zväčša koncovku `.php`.

Jazyk PHP je tzv. **CASE sensitive**, teda v ňom záleží na veľkosti písmen v názvoch premenných, konštánt, funkcií atď. V jazyku PHP sú teda príkazy `vypis()` a `Vypis()` rôzne príkazy.

Ako spúšťať PHP stránky na svojom počítači?

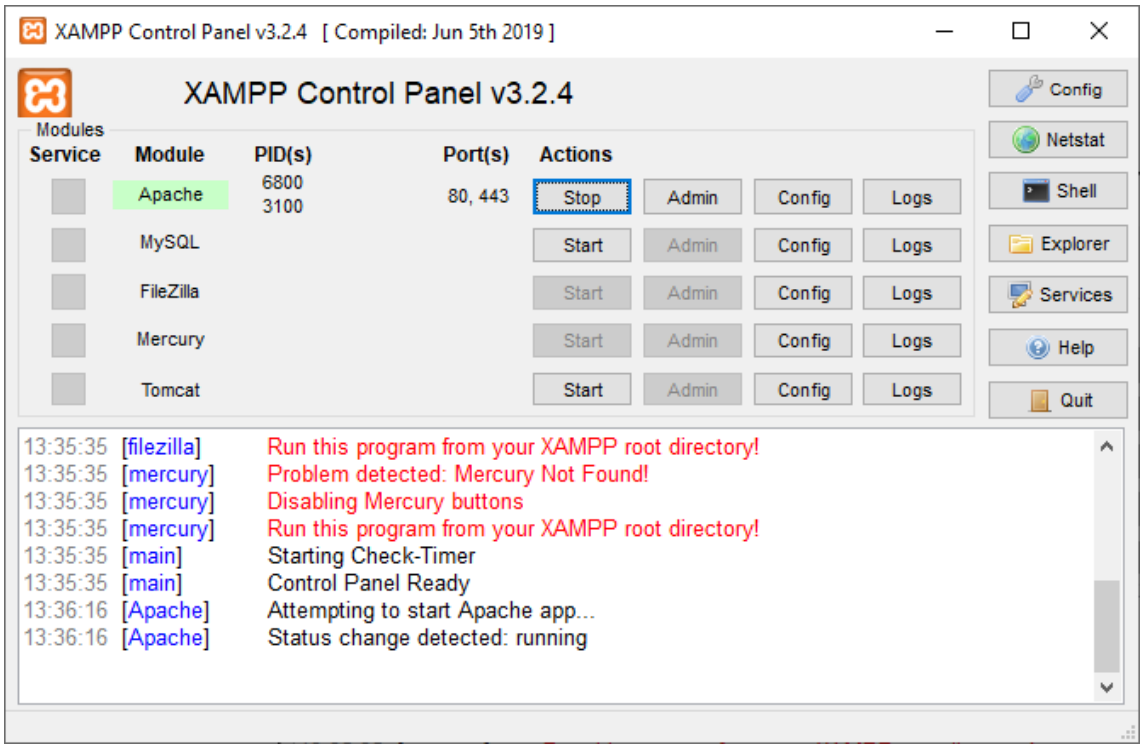
Ako sme už napísali vyššie, na spúšťanie (vykonávanie) PHP stránok potrebujeme viacero programov – webový server (napr. Apache), PHP interpreter a v prípade použitia databázy aj databázový server (napr. MySQL).

V súčasnosti už existujú aplikácie, ktoré v sebe integrujú všetky tri spomínané programy. Jednou z možností je program XAMPP[2]. XAMPP netreba inštalovať, stačí ho rozbaliť. Aby dobre fungovali cesty, treba spustiť súbor `setup_xampp.bat` a následne spustiť súbor `xampp-control.exe`. Ak sa nespustí webový server (Apache) automaticky, klikneme na príslušné tlačidlo `Start`. Korektné spustenie webového servera (Apache) vidíme na nasledujúcom obrázku.

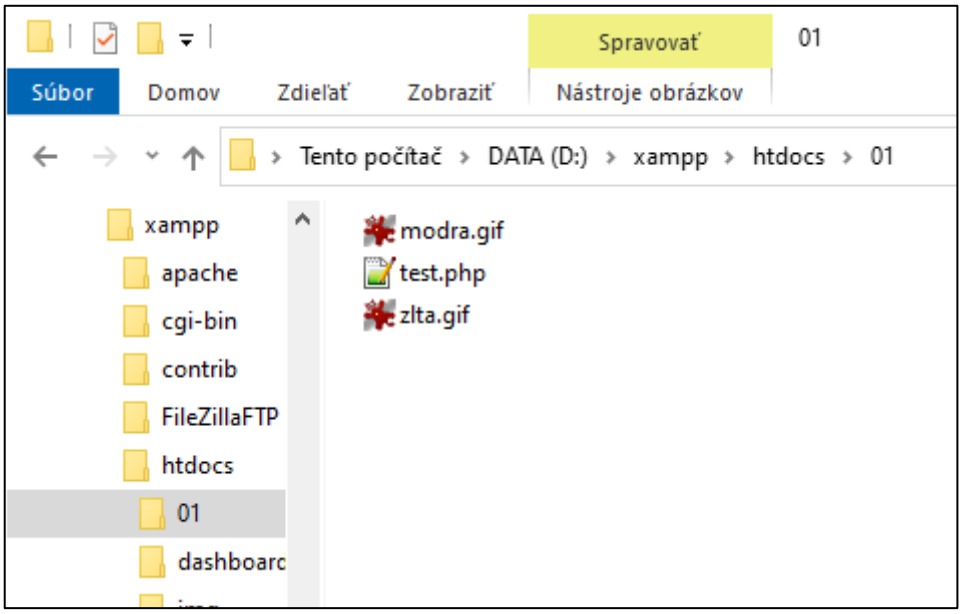
Odporúčame rozbaliť priamo na disk C: alebo D:.

[1] Ďalšie informácie a aj manuály nájdete na stránkach www.php.net, www.php5.sk.

[2] www.apachefriends.org



PHP stránky nemôžeme ukladať na ľubovoľné miesto na počítači, ale len tam, kam „vidí“ webový server. Pri štandardnom nastavení XAMPP je to priečinok `htdocs` v rámci priečinkov XAMPP na počítači.



localhost – takto označujeme webový server na lokálnom počítači

Ak si chceme PHP stránky spustiť na svojom počítači, spustíme si ľubovoľný webový prehliadač a v ňom zadáme adresu **`localhost`**. Zobrazí sa úvodná stránka nášho webového servera.

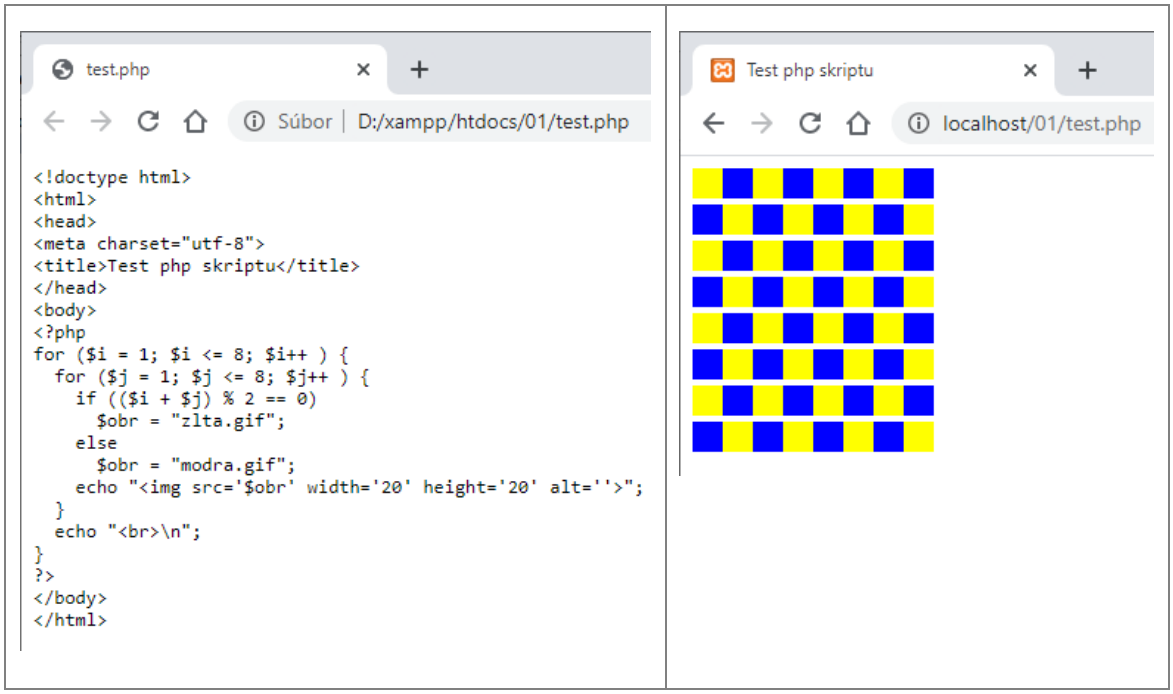
Teraz si ukážeme správny a nesprávny spôsob spúšťania (testovania) PHP stránok.

Nesprávny spôsob spúšťania PHP stránok

- Súbor -> Otvoriť... alebo
- Dvojkliknutie na PHP súbor

Správny spôsob spúšťania PHP stránok

- Zadanie adresy v prehliadači: `http://localhost/01/test.php`



Vidíme PHP kód!

NEvidíme PHP kód, ale vidíme výslednú dynamicky vygenerovanú webovú stránku!

Pri niektorých nastaveniach prehliadača sa môže stať, že nesprávne otvorená PHP stránka sa ani nezobrazí, ale sa môže uložiť (pokúšať sa uložiť) na disk počítača.

1.2 Tvorba PHP stránok

Stránky s PHP kódom môžeme písať v ľubovoľnom editore webových stránok. Pomôže nám, ak editor zvýrazňuje PHP syntax (príkazy, zátvorky, štruktúry atď.). Najlepšie je používať taký editor, ktorý má v sebe integrované PHP funkcie (napr. Notepad++, PhpStorm, Visual Studio Code, Atom, Brackets).

PHP stránka môže byť klasická HTML stránka s PHP vsuvkami, alebo môže byť celá zložená z PHP kódu.

ZAPAMÄTAJTE SI

PHP skript (kód) vkladáme medzi značky `<?php ... ?>` (túto časť nazývame PHP vsuvka). Do HTML kódu môžeme vložiť viacero PHP vsuviek.

Všetko, čo je v súbore mimo PHP vsuvky **`<?php ... ?>`**, sa automaticky prenesie do výsledného HTML súboru. Obsah PHP vsuvky sa vykoná a až jej výsledok sa vloží namiesto jej kódu. Výsledný zdrojový kód webovej stránky by mal byť korektný (validný) HTML kód. V princípe môžeme PHP vsuvky vkladať na ľubovoľné miesto, ale musí byť zachovaná validita výsledného HTML kódu. PHP vsuvkou môžeme dopĺňať/nahrádzať čokoľvek v zdrojovom kóde – obsah elementu (časť alebo celý), konkrétny element, kus kódu, celú stránku,...

PRÍKLAD 1.1

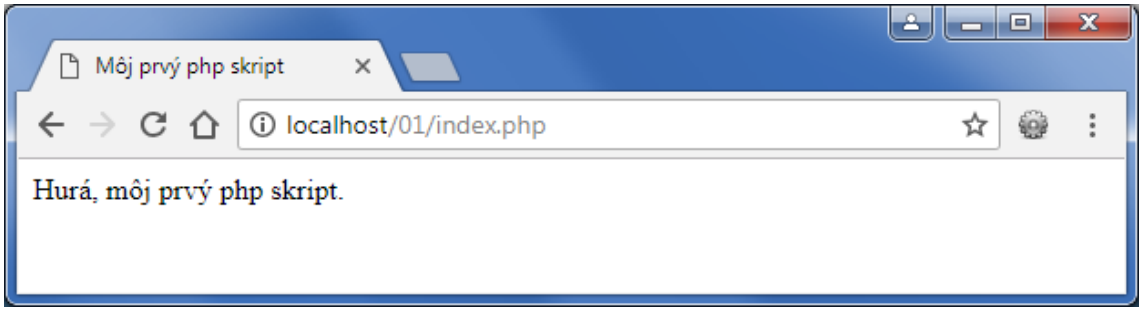
Vytváranie PHP kódu a generovanie výsledného zdrojového kódu webovej stránky si ukážeme na príklade. V nasledujúcom výpise je zdrojový kód stránky (súboru) `index.php` (`01/index.php`) spolu s PHP vsuvkou.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Môj prvý php skript</title>
</head>
<body>
<?php echo "Hurá, môj prvý php skript."; ?>
</body>
</html>
```

Pri načítaní stránky do webového prehliadača (`http://localhost/01/index.php`) webový server na našom počítači vykoná pomocou PHP interpretera PHP vsuvku a vygeneruje nový (a už výhradne HTML) zdrojový kód stránky:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Môj prvý php skript</title>
</head>
<body>
Hurá, môj prvý php skript.
</body>
</html>
```

Ten potom webový prehliadač interpretuje a zobrazí nám výslednú stránku ako na obrázku.



ZAPAMÄTAJTE SI

Pri korektnom zobrazení PHP stránky zo servera sa klient (používateľ cez webový prehliadač) nikdy nemôže dostať k zdrojovému PHP kódu, a teda nedokáže zistiť, ktorá časť stránky bola vygenerovaná PHP kódom a čo bol len statický HTML kód.

ÚLOHA 1.2

Otvorte súbor `prvy.html` (z priečinku `01`). Premenujte ho na `prvy.php`. Obsah súboru bude tvoriť zdrojový kód HTML stránky spĺňajúci štandard HTML5.

- Do tela stránky (element `<body>`) vložte PHP vsuvku s príkazom
  `echo "Toto je môj prvý PHP skript. Hurá, funguje.";`

Vyskúšajte si váš súbor zobraziť (spustiť) vo webovom prehliadači. Nezabudnite si pozrieť aj zdrojový kód zobrazenej stránky v prehliadači. Dbajte na to, aby výsledná stránka bola validným HTML dokumentom.

1.3 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto kapitoly je priblížiť tvorbu dynamických webových stránok na strane servera. Študenti sa dozvedia rozdiel medzi statickými a dynamickými webovými stránkami, oboznámia sa s nástrojmi na ich tvorbu, s tým, ako spúšťať dynamické stránky na strane servera a vytvoria prvý veľmi jednoduchý PHP súbor.

VÝKLAD

Prvá časť je hlavne teoretická. Študentov treba oboznámiť s tým:

- čo sú dynamické webové stránky,
- čo všetko je potrebné pri tvorbe dynamických webových stránok na strane servera,
- ako sa spracovávajú PHP stránky, keď príde požiadavka od klienta (prehliadača),
- ako sa do stránok vkladá PHP kód.

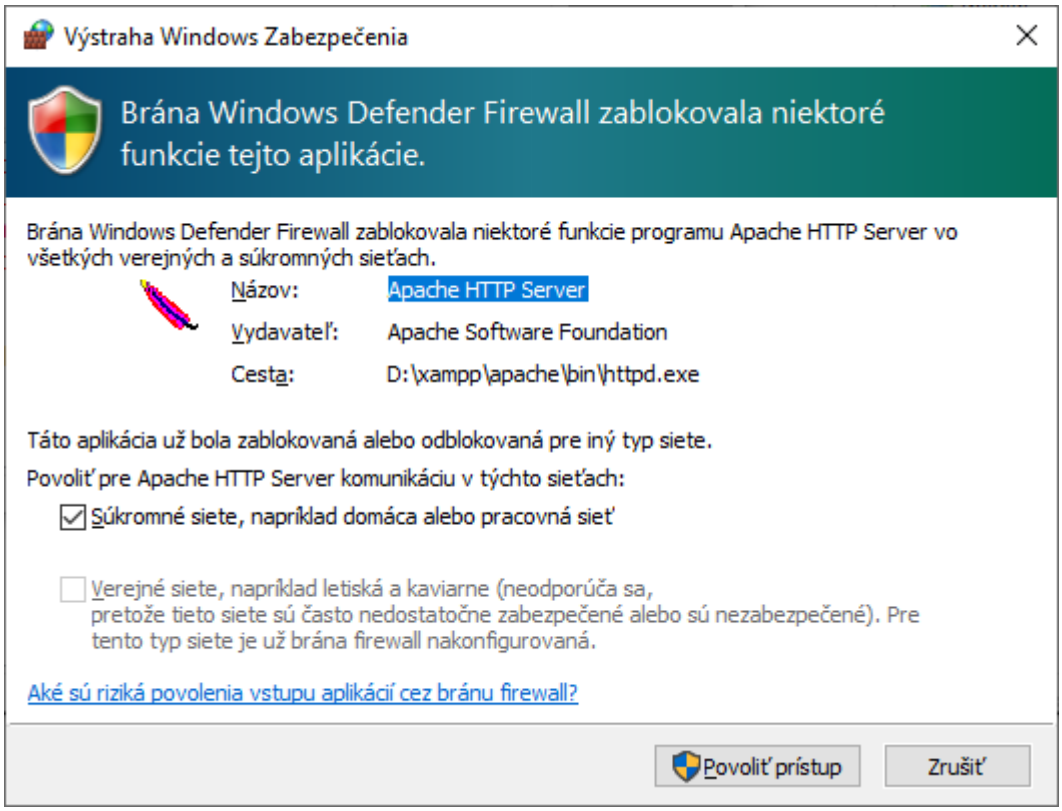
Pre prácu odporúčame používať XAMPP[3], čo je kombinácia populárnych webserverových aplikácií – Apache, PHP, MySQL a phpMyAdmin. XAMPP netreba inštalovať, stačí ho rozbaliť. Aby dobre fungovali cesty, treba pred samotným spustením XAMPP spustiť súbor `setup_xampp.bat` a následne spustiť súbor `xampp-control.exe`. Ak sa nespustí webový server (Apache) automaticky, klikneme na príslušné tlačidlo `Start`.

Automatické spúšťanie modulov XAMPP môžeme nastaviť v časti Config

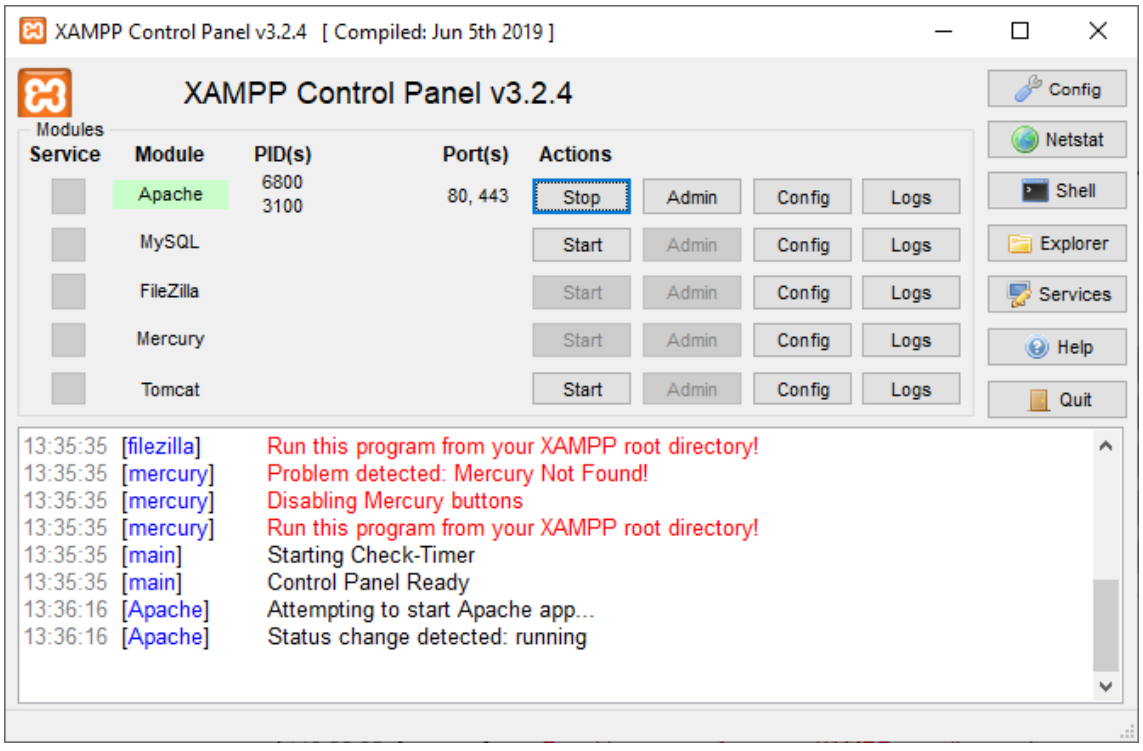
Pri spustení XAMPP (nielen prvom) môžu študentov miasť červené chybové správy v dolnej časti okna. Ak sa týkajú časti `[filezilla]` a `[mercury]`, môžeme ich upokojiť, že si tieto správy netreba všímať. Sú súčasťou balíka XAMPP, avšak my ich nebudeme využívať a chybové správy nemajú vplyv na iné časti XAMPP.

Pri prvom spustení XAMPP (resp. Apache) sa systém Windows môže pýtať na povolenia v rámci firewallu (Windows Defender Firewall). Aby webový server Apache fungoval, musíme ho povoliť.

[3] www.apachefriends.org



Pri korektnom spustení Apache sa jeho názov podfarbí a budú sa zobrazovať porty, na ktorých beží.



Na začiatku hodiny alebo pred samotným testovaním PHP stránok treba spustiť XAMPP. Stránky treba nakopírovať do priečinka `htdocs` v priečinku `XAMPP`. Keď sa zadá v prehliadači `localhost`, tak sa zobrazí práve obsah priečinka `htdocs`.

Pri štandardnom nastavení XAMPP sa v prehliadači zobrazí stránka z priečinka `dashboard`.

Ak na počítači pracujú (postupne) viacerí študenti, odporúčame vytvoriť pre nich v priečinku `htdocs` podpriečinky označené ich menami. Prípadne v nastaveniach XAMPP treba zmeniť domovský priečinok na nejaký zdieľaný priečinok na školskom serveri.

Pri module `Apache` klikneme na `Config`, potom `Apache (httpd.conf)` a v súbore nájdeme časť `DocumentRoot`. Tú zmeníme na našu požadovanú adresu a tú istú adresu nastavíme aj v časti `<Directory >`.

```
DocumentRoot "/xampp/htdocs"
<Directory "/xampp/htdocs">
```

Ukážkové súbory a aj súbory potrebné na vyriešenie úloh je potrebné nakopírovať do pracovných priečinkov každého študenta alebo do nejakého zdieľaného školského priečinka.

Okrem chybových správ v súbore `error.log` (pre XAMPP je v priečinku `apache/logs`) sa priebežne budeme zaoberať chybovými správami, ktoré sa zobrazujú aj priamo v prehliadači.

Ku časti: Ako skúšať PHP stránky na svojom počítači?

Pri nesprávnom zobrazení PHP stránky prehliadače Edge a Firefox zobrazia prázdnu stránku (ak sa vypisuje jednoduchý text) alebo nejaký kus kódu. Vtedy treba pozrieť zdrojový kód stránky v prehliadači. Toto treba študentom stále zdôrazňovať – „pozerajte si zdrojový kód stránky v prehliadači“.

K webovému serveru:

Namiesto programu XAMPP môžeme využiť iné programy/balíky napr. WAMP[4] alebo EasyPHP[5].

[4] **W**indows-**A**pache-**M**ySQL-**P**HP, www.wampserver.com
[5] EasyPHP, www.easyphp.org

2 PHP - ZÁKLADNÁ SYNTAX A ŠTRUKTÚRY (1)

V tejto časti sa oboznámime s väčšinou základných štruktúr jazyka PHP. Veríme, že väčšinu pojmov poznáte z iných programovacích jazykov, takže si treba hlavne všímať, ako jednotlivé prvky zapisujeme v PHP.

Príkazy v jazyku PHP oddeľujeme bodkočiarkou `;`.

Ak nebude v texte špeciálne uvedené, budeme príkazy zadávať do PHP vsuvky v hlavnom obsahu stránky (do elementu <body>).

2.1 Príkaz echo, reťazce (1)

Ak chceme niečo vypísať do výsledného HTML súboru, použijeme príkaz **`echo`**. Pomocou neho môžeme vypisovať čísla (celé, reálne) a texty.

PRÍKLAD 2.1

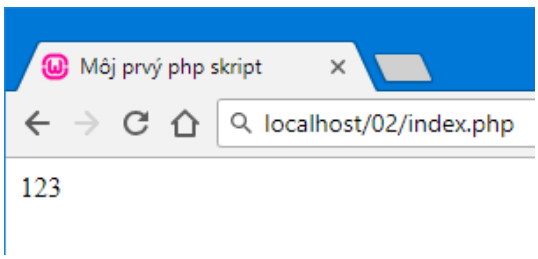
Vypíšme do tela stránky (elementu **`body`**) číslo 123.

02/index01.php

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Môj prvý php skript</title>
</head>
<body>
<?php echo 123; ?>
</body>
</html>
```

Zdrojový kód uložíme do súboru `index.php` a otvoríme vo webovom prehliadači. Nezabudnite, že v prehliadači zadáme adresu `localhost/02/index.php`.

Zobrazenie stránky (súboru) index.php v prehliadači



Vygenerovaný zdrojový kód stránky

```
1  <!doctype html>
2  <html>
3  <head>
4  <meta charset="utf-8">
5  <title>Môj prvý php skript</title>
6  </head>
7  <body>
8  123
9  </body>
10 </html>
11
```

ÚLOHA 2.2

Vyskúšajte zadávať nasledujúce príkazy a zistite, čo sa zobrazí v prehliadači (príkazy musíte vložiť do PHP vsuvky).

- `echo 3.14;` (desatinnú čiarku píšeme ako bodku `.`)
- `echo "Hurá, môj prvý php skript.";`
- `echo 'Hurá, môj prvý php skript.';`

Texty (reťazce) v PHP musíme vložiť do apostrofov (`'retazec1'`) alebo úvodzoviek (`"retazec2"`). Takto definované reťazce však nie sú rovnaké, resp. PHP k nim nepristupuje rovnako. Rozdiely si vysvetlíme neskôr. Pre bežné texty je jedno, ktorý typ reťazca použijeme.

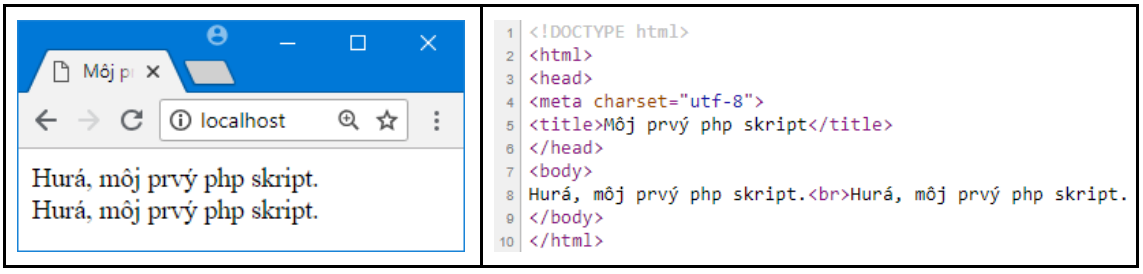
Ak vypisujeme texty, v skutočnosti vytvárame (generujeme) HTML zdrojový kód. Teda do reťazcov nemusíme dať len čistý text, ale môžeme v nich použiť aj HTML elementy.

02/index03.php

PRÍKLAD 2.3

Vypíšme reťazce oboma spôsobmi (pomocou úvodzoviek aj apostrofov), ale každý do samostatného riadka.

```
<body>
<?php
echo "Hurá, môj prvý php skript.<br>";
echo 'Hurá, môj prvý php skript.';
?>
</body>
```



ÚLOHA 2.4

Vypíšte texty: 3 + 4, 3 * 4, prípadne aj ďalšie príklady. Skúste ich vypísať pomocou jedného príkazu `echo`. A potom ich vypísať pomocou viacerých príkazov `echo`. Snažte sa, aby príklady boli v prehliadači pod sebou.

PRÍKLAD 2.5

02/index05.php

Vypíšme príklady z úlohy 2.4 tak, že každý príklad bude v samostatnom odseku.

```
<?php
echo '<p>3 + 4</p>';
echo '<p>3 * 4</p>';
?>
```

```
1  <!doctype html>
2  <html>
3  <head>
4    <meta charset="utf-8">
5    <title>Môj prvý php skript</title>
6  </head>
7  <body>
8  <p>3 + 4</p><p>3 * 4</p>
9  </body>
10 </html>
```

ÚLOHA 2.6

Vypíšte hlavný nadpis stránky (`h1`) s textom *Nadpis PHP stránky* a pod ním odsek s textom *Odsek textu s príkladom: 3 * 4* ako na obrázku.

2.2 Komentáre

Niekedy si pri písaní kódu potrebujeme niečo „poznačiť“, prípadne doplniť komentár ku bloku kódu, teda texty, ktoré sa nemajú vykonávať. V PHP môžeme vložiť dva typy komentárov.

1) `//` – takýto komentár platí po koniec riadka alebo po koniec PHP vsuvky (značka `?>`)
2) `/* viacriadkový komentár */` – toto je komentár, ktorý môžeme vložiť kdekoľvek v kóde a môže zasahovať do viacerých riadkov.

```
<body>
<?php
// Príklad 2.3 - výpis elementu
echo "Hurá, môj prvý php skript.<br>";
echo 'Hurá, môj prvý php skript.';

/* Úloha 2.4
echo "3 * 4";
echo "3 + 4";
echo "3 - 4";
echo "3 / 4";
*/
?>
</body>
```

ZAPAMÄTAJTE SI

Nesmieme zabúdať na to, že takéto komentáre platia iba v rámci PHP vsuvky `<?php ... ?>`

2.3 Premenné

Každá premenná musí začínať znakom **$**. Tým sa odlišuje od ostatných štruktúr a textov, napr. `$pocet`, `$meno`.

V názve premennej môžu byť len písmená, číslice a podčiarkovník (_). Hneď za znakom `$` nesmie nasledovať číslica.

Premennú v PHP vytvárame **priradením** hodnoty do premennej, teda napr. príkazom `$cislo = 123`. Nie je nutné premenné nejako špeciálne definovať ani určiť typ premenných. PHP si vnútorne definuje typ premennej podľa toho, akú jej priradíme hodnotu.

PRÍKLAD 2.7

02/index07.php

Zadefinujme si niekoľko premenných s rôznymi hodnotami a vypíšme ich.

```
$cislo = 123;    //(typ celé číslo)
$realne = 12.3;  //(typ reálne číslo)
$ret1 = "Ahoj";  //(typ reťazec)
$ret2 = 'Ahoj';  //(typ reťazec)
echo $cislo;
echo $realne;
echo $ret1;
echo $ret2;
```

2.4 Konštanty

Konštanta je ako keby premenná, ktorá má stále tú istú hodnotu a v kóde ju už nemôžeme zmeniť.

Ku konštantám sa musíme správať inak ako k premenným. Skôr ako chceme používať nejakú konštantu, musíme ju definovať. Konštantu definujeme príkazom `define("názov_konštanty", hodnota)`.

Názvy konštánt nesmú začínať znakom `$`, a tiež nesmú začínať číslom. Zvyčajne ich píšeme veľkými písmenami (aby sme ich odlíšili od textov a premenných), ale nie je to žiadne striktné pravidlo.

PRÍKLAD 2.8

Definujme konštantu `POCET` s hodnotou `12` a vypíšme ju. Potom definujme konštantu `AUTOR` s hodnotou `Janko Hraško`, a tiež ju vypíšme.

```
define("POCET", 12);
echo POCET;
echo '<br>';
define("AUTOR", 'Janko Hraško');
echo AUTOR;
```

2.5 Reťazce (2)

V predchádzajúcich príkladoch sme si vytvorili rôzne premenné a konštanty a vypisovali sme ich. Zvyčajne ich však vypisujeme spolu s nejakým textom. Napríklad:

```
$cislo = 123;
echo "Výsledok je ";
echo $cislo;
```

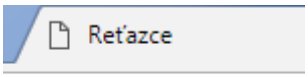
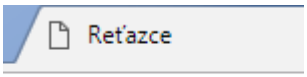
Takýto zápis je korektný, ale zbytočne predlžuje kód. V časti 2.1 (Reťazce (1), príkaz echo) sme písali, že reťazce môžeme zapisovať do úvodzoviek aj apostrofov. Poďme si ukázať rozdiely a takisto, ako nám môžu skrátiť a sprehľadniť kód.

Reťazec **v úvodzovkách** PHP interpreter analyzuje a v prípade, že v ňom identifikuje (nájde) premenné alebo špeciálne znaky, vyhodnotí ich. Naproti tomu v reťazci **v apostrofoch** PHP interpreter nič neanalyzuje a celý reťazec vypíše, ako je – premenné a špeciálne znaky sa vypíšu ako obyčajné znaky.

PRÍKLAD 2.9

02/index09.php

V nasledujúcich skriptoch sa pokúsime vypísať výsledok (premennú) spolu s textom. Jediný rozdiel je v zápise (definovaní) reťazca.

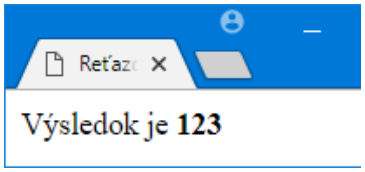
`$cislo = 123;` `echo "Výsledok je $cislo";`	`$cislo = 123;` `echo 'Výsledok je $cislo';`
 Výsledok je 123	 Výsledok je $cislo

Vidíme, že kým v príkaze, uvedenom v ľavej časti tabuľky, PHP interpreter analyzoval reťazec a našiel premennú (a aj ju vyhodnotil), tak v príkaze, uvedenom v pravej časti tabuľky, sa vypísal reťazec, ako sme ho zadali.

02/index10.php

PRÍKLAD 2.10

Zmeňme výpis z predchádzajúceho príkladu tak, aby hodnota čísla bola zvýraznená tučným písmom.

`$cislo = 123;` `echo "Výsledok je <strong>$cislo</strong>";`	 Výsledok je **123**

ÚLOHA 2.11

Porozmýšľajte a potom overte, čo vypíšu nasledujúce príkazy:

```
$cislo = 20;
echo "Môj brat má $cislo rokov.<br>";
echo '$cislo je moje šťastné číslo.<br>';
echo "<p>Vyrieš príklad $cislo - 4</p>";
echo 'Premenná <em>$cislo</em> má hodnotu $cislo<br>';
echo "Premenná <em>$cislo</em> má hodnotu $cislo<br>";
```

Zreťazovanie

V predchádzajúcich častiach sme vypisovali samostatné texty, premenné, konštanty alebo texty v kombinácii s premennými. Skúsme teraz jedným príkazom `echo` vypísať text a konštantu. Ako prvé by nás zrejme napadlo toto riešenie:

```
define("POCET", 12);
echo "Počet záznamov je: POCET";
```

Keď však toto riešenie vyskúšame, zistíme, že namiesto čísla 12 (hodnota konštanty `POCET`) sa vypíše text `POCET` a to napriek tomu, že sme na výpis použili úvodzovky. Keďže názvy konštánt sa nezačínajú žiadnym špeciálnym znakom, PHP ich v reťazci nemá ako identifikovať. Konštanty teda na rozdiel od premenných **nemôžeme** vypisovať priamo v reťazci. Môžeme však využiť

zreťazovanie (spájanie) reťazcov, ktoré robíme pomocou znaku bodka (`.`). Upravme predchádzajúci kód nasledovne:

```
define("POCET", 12);
echo "Počet záznamov je: " . POCET;
```

V tomto prípade už konštanta POCET nie je súčasťou reťazca uzavretého v úvodzovkách, ale samostatnou časťou, ktorú pomocou znaku `.` pripájame za text *"Počet záznamov je: "*. Druhý riadok kódu by sme mohli interpretovať takto: *Vypíš reťazec, ktorý dostaneš spojením textu "Počet záznamov je: " a hodnoty konštanty POCET*. Teraz teda dostaneme očakávaný výpis: *Počet záznamov je: 12*

ÚLOHA 2.12

02/index12.php

Pozrite a vyskúšajte si aj nasledujúce príklady zreťazenia. Potom zmeňte hodnoty premenných `$cislo` a `$retazec`, či konštanty `POCET` a skúste znova.

```
$cislo = 10;
define("POCET", 12);
$retazec = "PHP";
echo "Konštanta POCET má hodnotu " . POCET . "<br>";
echo "Na rukách máme " . $cislo . " prstov.<br>";
echo $retazec . " je super.<br>";
echo POCET . $retazec . $cislo . "<br>";
$veta = POCET . "-krát napíš: <em>Mám rád " . $retazec .
"!</em><br>";
echo $veta;
```

Vidíme, že PHP zo všetkých častí zreťazenia spraví jeden reťazec. Ten následne môžeme napr. vypísať, alebo ho priradiť do premennej. Nemusíme riešiť to, či jednotlivé časti sú text alebo číslo – PHP ich bez problémov spojí.

ÚLOHA 2.13

Pomocou PHP kódu vypíšte text: `Premenná $cislo má hodnotu 10.`, pričom posledné číslo vo vete (10) bude skutočná hodnota premennej `$cislo`.

Vypisovanie špeciálnych znakov

Pozrime si zdrojový kód stránky, ktorú sme vytvorili v príklade 2.12 (pozor, nie PHP kód, ale HTML kód už vygenerovanej stránky).

```
7 <body>
8 Konštanta POCET má hodnotu 12Na rukách máme 10 prstov.<br>PHPje super.<br>12PHP1012-krát napíš:
9 </body>
```

Vidíme, že všetky texty, ktoré sme vypisovali príkazom `echo`, sa vypísali vedľa seba. Vytvorili sme tak veľmi zle čitateľný HTML kód, v ktorom sa ťažko orientuje a zle sa v ňom hľadajú chyby. Ak by sme chceli vytvoriť krajší HTML kód, mali by sme zabezpečiť, aby každý príkaz `echo` "zapisoval na nový riadok". V tomto prípade nám nepomôže vložiť element `<br>`, lebo ten **nezalamuje** riadok v zdrojovom kóde, ale v samotnom prehliadači.

Zalomenie riadka v zdrojovom HTML kóde riešime pomocou špeciálneho znaku `\n`, ktorý **musíme** v reťazci zapisovať v úvodzovkách.

02/index14.php

PRÍKLAD 2.14

Doplňme zalomenie riadka `\n` do zdrojového kódu stránky.

```
echo "<h1>Nadpis stránky</h1>\n";
echo '<p>Odsek textu</p>';
echo "<h1>Nadpis stránky</h1>\n<p>Odsek textu</p>";
echo "<h2>Podnadpis</h2>\n<p>1. odsek</p>\n<p>2. odsek</p>";
```

HTML kód, ktorý vytvoríme týmto skriptom, vyzerá nasledovne:

```
<body>
<h1>Nadpis stránky</h1>
<p>Odsek textu</p><h1>Nadpis stránky</h1>
<p>Odsek textu</p><h2>Podnadpis</h2>
<p>1. odsek</p>
<p>2. odsek</p>
</body>
```

ÚLOHA 2.15

Upravte PHP skript z príkladu 2.12 tak, aby generoval čitateľnejší HTML kód.

Úvodzovky a apostrofy v reťazcoch

Pri vypisovaní znakov apostrof (`'`) a úvodzovky (`"`) si musíme dať veľký pozor. Nemôžeme ich len tak vložiť do reťazca, ktorý začína tým istým znakom, lebo by to PHP interpreter pochopil ako ukončenie reťazca a nevedel by, čo so zvyšným textom.

Nesprávny zápis

```
echo "text "xyz" text";
echo 'text 'xyz' text';
```

Správny zápis

```
echo "text 'xyz' text";
echo 'text "xyz" text';
```

Pre výpis úvodzoviek alebo apostrofov v texte môžeme tiež využiť znak `\`, ktorý zruší význam nasledujúceho znaku.

```
echo "text \"xyz\" text";
echo 'text \'xyz\' text';
```

ÚLOHA 2.16

Máme definované premenné `$obrazok = "obrazky/logo.jpg"` a `$adresa = "www.rtvs.sk"`. Pomocou PHP (príkazu `echo`) vytvorte HTML kód, v ktorom bude:

- element `<a href="http://www.facebook.com">Facebook</a>`,
- element `<img src='ikona.gif' alt='ikona' width='20' height='20'>`,
- element, definujúci odkaz na stránku, ktorej adresa je uložená v premennej `$adresa`

(textom odkazu môže byť tiež hodnota premennej `$adresa`),
- element definujúci obrázok, ktorého adresa je uložená v premennej `$obrazok` vrátane atribútu `alt`).Upravte PHP skript z príkladu 2.12 tak, aby generoval čitateľný HTML kód.

2.6 Operátory

V PHP rozlišujeme niekoľko typov operátorov. Najdôležitejšie z nich si vysvetlíme a ukážeme.

Číselné

Základné operátory sčitovanie, odčitovanie a násobenie (**+**, **-**, *****) netreba ani predstavovať. Fungujú tak, ako by sme očakávali.

Operátor delenia **/** funguje na celých a desatinných číslach, ale jeho výsledok je **vždy** desatinné číslo.

Operátor **%** (zvyšok po delení celých čísel) funguje len na celých číslach. V prípade, že ho použijeme pri desatinných číslach, tak PHP odtrhne desatinné časti a vykoná operáciu so zvyšnými celými číslami. Napr.

- príkaz **`echo 10 % 3;`** vypíše **1**,
- aj príkaz **`echo 10.5 % 3;`** vypíše **1**,
- aj príkaz **`echo 10.5 % 3.7;`** vypíše **1**,
- príkaz **`$zvysok = 12 % 4;`** priradí do premennej **`$zvysok`** hodnotu **0**.

Jazyk PHP poskytuje tiež operátory zvýšenia hodnoty o 1 (**++**) a zníženia hodnoty o 1 (**--**). Sú to vlastne skrátené zápisy operácií `+ 1` a `- 1`. Napríklad zápis `$vysl++` by sme mohli zapísať aj ako `$vysl = $vysl + 1`. Podobne je to aj pri operátore `--`.

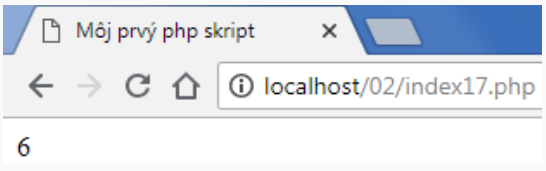
Operátory zvýšenia a zníženia hodnoty o 1 spájame len s premennými a medzi premennou a operátorom nesmie byť medzera.

PRÍKLAD 2.17

02/index17.php

Vyskúšajme si operátor zvýšenia hodnoty premennej $vysl o 1. Výsledok kódu bude číslo 6.

```
$vysl = 5;
$vysl++;
echo $vysl;
```



Priraďovacie

Okrem základného operátora priradenia (**=**) v PHP existujú aj tzv. zložené priraďovacie operátory, ktoré sú kombináciou dvoch operátorov, teda sú to skrátené zápisy.

`+=`, `-=`, `*=`, `/=`, `%=`, `.=`

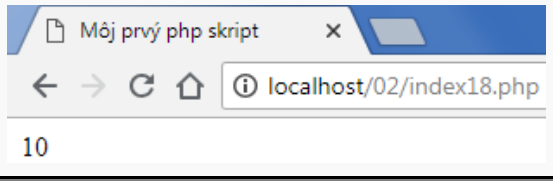
Napríklad zápis `$vysl += 4` je v skutočnosti príkaz `$vysl = $vysl + 4`. Oba sú však ekvivalentné a záleží len na používateľovi, ktorý bude používať. (Takéto priradenia sú napr. aj v jazyku Python.)

02/index18.php

PRÍKLAD 2.18

Vyskúšajme si operátor zloženého priradenia. Výsledkom bude číslo 10.

```
$vysl = 5;
$vysl *= 2;
echo $vysl;
```



ÚLOHA 2.19

Máme kód:

```
$vysl = 12;
$vysl  ..... ;
echo $vysl;
```

Aký **zložený** priraďovací príkaz treba doplniť namiesto bodiek a s akým číslom, aby sa v prehliadači vypísalo číslo: a) 24, b) 29, c) 3, d) 5? Má každá z týchto úloh len jediné riešenie?

Existuje aj operátor `and` – a zároveň (môžeme ho písať aj veľkými písmenami – `AND`) a operátor `or` – alebo (môžeme ho písať aj veľkými písmenami – `OR`). Avšak tieto operátory sa nemusia vyhodnocovať v rovnakom poradí ako operátory `&&` a `||`, preto ich nebudeme používať.

Logické

Medzi základné logické operátory patria:

- `&&` – a zároveň,
- `||` – alebo,
- `!` – negácia (ak použijeme operátor spolu s premennou, tak medzi operátorom a názvom premennej nesmie byť medzera, napr. `!$vysledok`).

Pri vytváraní logických výrazov môžeme používať zátvorky.

PRÍKLAD 2.20

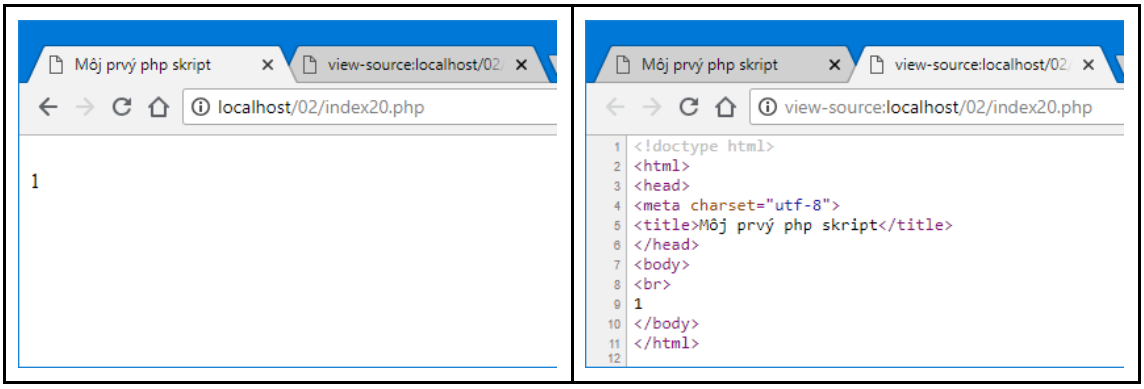
Čo bude výsledkom nasledujúcich príkazov?

```
$a = true; $b = false;
$vysledok1 = $a && $b;
$vysledok2 = $a && !$b;
```

Dajme si jednotlivé výsledky vypísať (každý v samostatnom riadku):

```
echo $vysledok1;
echo "<br>\n";
echo $vysledok2;
```

Zistíme, že v prehliadači máme prvý riadok prázdny, teda sa nám nevypísala premenná `$vysledok1` (pozri obrázok vľavo, resp. zdrojový kód na obrázku vpravo). V skutočnosti sa premenná `$vysledok1` vypísala, ale jej hodnota bol prázdny reťazec, lebo výsledok operácie `&&` bola hodnota `FALSE`, ktorú PHP pri vypisovaní prerobil na prázdny reťazec. Naproti tomu, v premennej `$vysledok2` mala byť hodnota `TRUE`, ale PHP vypísalo `1`, teda zase PHP pri vypisovaní prerobil hodnotu `TRUE` na číslo `1`.



Z uvedeného príkladu môžeme vidieť, že PHP automaticky konvertuje typy hodnôt podľa potreby a situácie. Ak by sme hodnoty premenných `$a` a `$b` zmenili na číselné, výsledok by bol rovnaký. Pri premennej `$a` je jedno aké veľké číslo by sme dali.

```
$a = 1; $b = 0;  // $a = 123;
$vysledok1 = $a && $b;
```

ZAPAMÄTAJTE SI

Logickej hodnote `FALSE` sú ekvivalentné hodnoty `0`, `0.0`, prázdny reťazec, `"0"`, prázdne pole a špeciálna hodnota `NULL`. Všetky ostatné hodnoty chápe PHP ako `TRUE`.

ÚLOHA 2.21

Zistite, aké hodnoty budú v premenných `$v1` a `$v2` po jednotlivých priradeniach.

```
$c = 1; $d = 0; $e = 0;
$v1 = $c || $d && $e;
$v2 = ($c || $d) && $e;
```

Porovnávacie

Medzi základné operátory porovnania patria: `==`, `!=` (nerovnosť), `<`, `>`, `<=`, `>=`.

```
$vysledok = 3 < 5;
echo $vysledok;
echo 3 > 5;
```

2.7 Príkaz if

Podmienený príkaz `if` použijeme vtedy, ak chceme časť kódu vykonať len pri splnení nejakej podmienky (prípadne viacerých podmienok). Základná verzia príkazu je:

```
if (podmienka) {
  // príkazy
}
```

V časti `podmienka` zapíšeme jednu alebo viacero podmienok, ktoré chceme otestovať. Ak používame viacero podmienok, spájame ich logickými operátormi. Podmienka musí byť vždy v zátvorkách `( )`.

02/podmienky22.php

PRÍKLAD 2.22

Naprogramujme dva podmienené výpočty:

- Do premennej `$body` uložme počet bodov získaných za test. Ak študent získa viac ako 60 bodov, vypíšeme text *Test si spravil*.
- Do premennej `$den` uložme nejaký deň v týždni. Ak to bude sobota alebo nedeľa vypíšeme text *Krásny víkend*.

Skúšajme meniť hodnoty premenných `$body` a `$den`.

```
$body = 85;
if ($body > 60) {
  echo 'Test si spravil';
}
echo '<br>';
$den = 'sobota';
if ($den == 'sobota' || $den == 'nedela') {
  echo 'Krásny víkend!';
}
```

Ak chceme vykonať nejaký príkaz (príkazy) vtedy, keď podmienka neplatí, môžeme použiť zápis príkazu `if` s časťou **`else`**. Časť `else` môžeme použiť len raz.

```
if (podmienka) {
  // príkazy 1
} else {
  // príkazy 2
}
```

02/podmienky23.php

PRÍKLAD 2.23

Upravme príklad 2.22 tak, že využijeme aj časť `else` v podmienke:

- Ak počet bodov v premennej `$body` nebude viac ako 60, vypíšeme text *Test si nespravil*.
- Ak v premennej `$den` nebude sobota alebo nedeľa, vypíšeme text *Šup do školy*.

```php
$body = 85;
if ($body > 60) {
  echo 'Test si spravil';
} else {
  echo 'Test si nespravil';
}
```

```php
$den = 'streda';
if ($den == 'sobota' || $den ==
'nedela') {
  echo 'Krásny víkend!';
} else {
  echo 'Šup do školy';
}
```

PRÍKLAD 2.24

02/podmienky24.php

Takúto podmienku môžeme využiť napr. pri vypisovaní známky na základe bodov.

```php
$body = 85;
if ($body > 90) {
  echo 1;
} else if ($body > 80) {
  echo 2;
} else if ($body > 70) {
  echo 3;
} else if ($body > 60) {
  echo 4;
} else {
  echo 5;
}
```

ÚLOHA 2.25

Napíšte skript, ktorý podľa hodnoty premennej `$cas` vypíše jednu zo správ:

- *Dobré ráno,* ak je menej ako 8 hodín,
- *Dobrý deň*, ak je viac ako 8 hodín, ale menej ako 18,
- *Dobrý večer*, ak je medzi 18 a 22 a
- *Dobrú noc* v ostatných prípadoch.

Svoj skript vyskúšajte pre rôzne hodnoty premennej `$cas` (len medzi 0 a 24).

PRÍKLAD 2.26

Zoberme kód z príkladu 2.20:

```php
$a = 1; $b = 0;
$vysledok1 = $a && $b;
```

Ak by sme nechceli premennú `$vysledok1` rovno vypisovať, ale podľa nej vypísať príslušné texty, mohli by sme zapísať

```php
$a = 1; $b = 0;
$vysledok1 = $a && $b;
if ($vysledok1) {
  echo 'platí';
} else {
  echo 'neplatí';
}
```

Vidíme, že do podmienky nemusíme dať vždy skutočnú podmienku (výraz), ale môžeme dať len premennú. Ak v premennej nie je logická hodnota (`TRUE`/`FALSE`), PHP aplikuje automatické konvertovanie, ktoré sme uviedli v časti *Logické operátory*. Napr. ak by sme do podmienky vložili premennú s nenulovým číslom, PHP ju vyhodnotí ako `TRUE`. Môžeme to využiť napr. vtedy, ak chceme ďalej pracovať len s nenulovou alebo neprázdnou premennou.

ZAPAMÄTAJTE SI

Podmienka musí byť vždy v zátvorkách `( )`.

2.8 Cykly

Ak potrebujeme viackrát vykonať rovnaké príkazy, môžeme použiť niektorý cyklus (napr. `for` alebo `while`).

Cyklus for

Cyklus `for` zapisujeme nasledovne:

```
for (inicializácia; podmienka; príkaz) {
  // príkazy
}
```

- **`inicializácia`** – príkaz v tejto časti sa vykonáva iba na začiatku cyklu (zvyčajne inicializácia hodnoty počítadla cyklu)
- **`podmienka`** – táto časť je zvyčajne výraz – ak je tento výraz `true`, zoznam príkazov vnútri cyklu sa vykoná, inak sa cyklus ukončí
- **`príkaz`** – príkaz v tejto časti sa vykoná na konci každého cyklu (zvyčajne zvyšovanie počítadla cyklu)

02/cykly27.php

PRÍKLAD 2.27

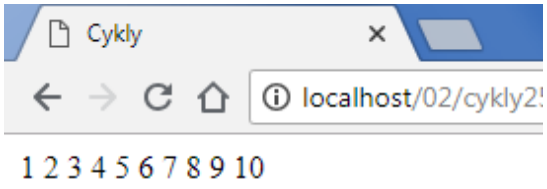
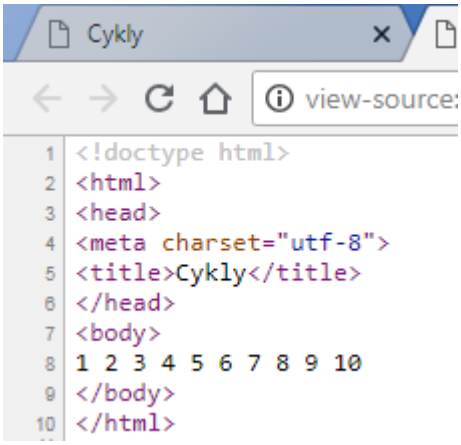
Vypíšme pomocou cyklu `for`

a. čísla od 1 do 10
b. násobky čísla 5 v opačnom poradí od 100 do 0, teda 100 95 90 5 0

Mohli by sme to vyriešiť aj inak?

```
for ($i = 1; $i <= 10; $i++) {
  echo "$i ";
}

for ($i = 100; $i >= 0; $i = $i - 5) {
  echo "$i ";
}
```

Zobrazenie prvého cyklu (a) v prehliadači a v zdrojovom kóde:

zobrazenie v prehliadači	zdrojový kód
	

ÚLOHA 2.28

Pomocou cyklu `for` vypíšte:

- **a.** násobky čísla 5 do 100, teda 0 5 10 15 ... 95 100,
- **b.** prvých 10 čísel Fibonanciho postupnosti, teda 1 1 2 3 5 8 13 21 34 55.

Ďalšie číslo Fibonanciho postupnosti vypočítame ako súčet posledných dvoch čísel postupnosti. Prvé a druhé číslo postupnosti sú 1.

Cyklus while

Druhým typom cyklu je cyklus **`while`**. Ten zapisujeme nasledovne:

```
while (podmienka) {
  // príkazy
}
```

Ak podmienka platí, vykonávajú sa príkazy v tele cyklu. Podmienka musí byť vždy v zátvorkách `( )`.

PRÍKLAD 2.29

Vypíšme pomocou cyklu `while`

- **a.** čísla od 1 do 10
- **b.** násobky čísla 5 od 0 do 100

02/cykly29.php

```
$i = 1;
while ($i <= 10) {
  echo "$i ";
  $i++;
}
$i = 0;
while ($i <= 100) {
  echo "$i ";
  $i = $i + 5;   // $i += 5;
}
```

ÚLOHA 2.30

Napíšte skript, ktorý vypíše:

- do jedného riadku mocniny dvojky menšie ako hodnota premennej `$koniec`:, napr. ak `$koniec` je 100, bude výpis nasledovný: 1 2 4 8 16 32 64
- do jedného riadku čísla od 0 do 30, pričom násobky trojky zvýrazní: **0** 1 2 **3** 4 5 **6** 7 8 **9** 10 11 **12** 13 14 **15** 16 17 **18** 19 20 **21** 22 23 **24** 25 26 **27** 28 29 **30**

V skripte využite cyklus `for` a/alebo `while`.

2.9 Polia

Tak ako v iných programovacích jazykoch aj v PHP je každý prvok poľa identifikovaný svojím indexom, resp. kľúčom (budeme používať oba termíny). Kľúč uvádzame v hranatých zátvorkách, napr. `$pole[7]`. Tento kľúč môže byť celé číslo alebo reťazec.

Číselné indexy

Číselné indexy štandardne začínajú od 0. Takéto pole môžeme naplniť postupným pridávaním prvkov do poľa v tvare `$pole[] = hodnota` alebo zadaním celého poľa v tvare `$pole = [hodnota1, hodnota2, ... , hodnotaN]`.

02/polia31.php

PRÍKLAD 2.31

Naplňme pole `$mena` postupne prvkami *Adam*, *Barbora*, *Cyril* a *Dano*. Pole `$tyzden` naplňme zadaním celého poľa. V poli budú dni od *pondelka* do *piatka*.

```
$mena[] = 'Adam';
$mena[] = 'Barbora';
$mena[] = 'Cyril';
$mena[] = 'Dano';
$tyzden = ['pondelok', 'utorok', 'streda', 'štvrtok', 'piatok'];
```

ÚLOHA 2.32

Doplňte pole `$tyzden` o dni *sobota* a *nedeľa*.

Obsah poľa môžeme vypísať pomocou príkazu **`print_r(pole)`**, kde `pole` môže byť premenná typu pole, alebo aj priamo zadané pole napr. **`['a', 'b', 'c', 'd']`**.

PRÍKLAD 2.33

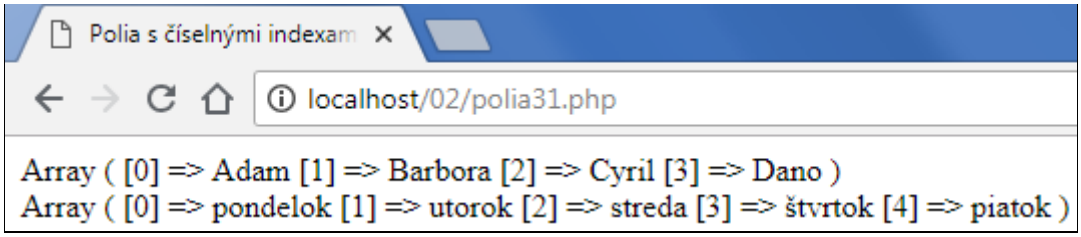
02/polia33.php

Vypíšme obsah polí `$mena` a `$tyzden` (z príkladu 2.29) pomocou funkcie `print_r()`.

Príkaz `print_r()` budeme využívať len na pomocné výpisy.

```
print_r($mena);
echo "<br>\n";
print_r($tyzden);
```

V prehliadači sa nám zobrazí:



A v zdrojovom kóde:

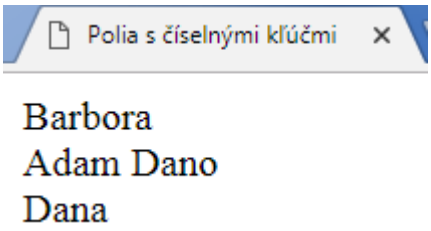
```
<body>
Array
(
    [0] => Adam
    [1] => Barbora
    [2] => Cyril
    [3] => Dano
)
<br>
Array
(
    [0] => pondelok
    [1] => utorok
    [2] => streda
    [3] => štvrtok
    [4] => piatok
)
</body>
```

Pomocou príkazu `echo` nevieme vypísať celé pole, teda nefunguje `echo $mena;`, ale vieme vypísať jednotlivé prvky poľa.

PRÍKLAD 2.34

02/polia34.php

Vypíšme 2. prvok poľa `$mena` (s kľúčom 1) a potom do ďalšieho riadku prvky s kľúčmi 0 a 3 oddelené medzerou. Ďalej zmeňme meno *Dano* na *Dana* (kľúč 3) a tento prvok znovu vypíšme.

zdrojový kód	výstup v prehliadači
`echo $mena[1];` `echo " \n";` `echo "$mena[0] $mena[3] \n";` `$mena[3] = 'Dana';` `echo $mena[3];`	

Prechádzanie poľa

Najuniverzálnejší spôsob prechádzania poľa je pomocou funkcie **`foreach()`**.

Cyklus `foreach()` môžeme použiť dvomi spôsobmi. V prípade, že nás z poľa zaujímajú len hodnoty, použijeme

`foreach ( pole as $hodnota ) { príkazy }`.

Do premennej `$hodnota` sa postupne (v každom prechode cyklu) uložia jednotlivé **hodnoty poľa**. V prípade, že potrebujeme nejakým spôsobom pracovať aj s kľúčmi poľa, musíme použiť

`foreach ( pole as $kluc => $hodnota ) { príkazy }`.

Do premenných `$kluc` a `$hodnota` sa postupne (v každom prechode cyklu) uložia **kľúče a príslušné hodnoty poľa**.

Pri výpise poľa pomocou cyklu `foreach()` si už môžeme sami definovať, ako bude výpis vyzerať.

02/polia35.php

PRÍKLAD 2.35

Výpíšme obsah poľa `$mena` tak, aby bolo každé meno na samostatnom riadku v prehliadači aj v zdrojovom kóde. Ďalej vypíšme obsah poľa `$tyzden` tak, aby sa v samostatných riadkoch vypísali jednotlivé dni v tvare kľúč = deň.

```
foreach($mena as $hodnota) {
  echo "$hodnota <br>\n";
}
foreach($tyzden as $kluc => $hodnota) {
  echo "$kluc = $hodnota <br>\n";
}
```

ÚLOHA 2.36

Vytvorte pole `$trieda` a naplňte ho krstnými menami študentov vašej triedy. Potom toto pole vypíšte tak, aby boli všetky mená v jednom riadku a za každým bola čiarka a medzera (aj za posledným menom).

Reťazcové kľúče - Asociatívne polia

Okrem číselných kľúčov môžu mať polia aj reťazcové kľúče. Takéto polia nazývame **asociatívne**. Takéto pole naplníme postupným pridávaním prvkov do poľa v tvare `$pole['kľúč'] = hodnota` alebo zadaním celého poľa v tvare `$pole = ['kľúč1' => hodnota1, 'kľúč2' => hodnota2, 'kľúč3' => hodnota3, ...]`.

PRÍKLAD 2.37

02/polia37.php

Vytvorme pole `$osoba` s kľúčmi `meno`, `priezvisko`, `rok narodenia`, `mobil`, `mail` a hodnotami *Jožko*, *Mrkvička*, *2000*, *0999123456*, *jozko@mrkvicka.sk*. Potom vypíšme prvok s kľúčom `priezvisko`.

```
$osoba['meno'] = 'Jožko';
$osoba['priezvisko'] = 'Mrkvička';
$osoba['rok narodenia'] = 2000;
$osoba['mobil'] = '0999123456';
$osoba['mail'] = 'jozko@mrkvicka.sk';

echo $osoba['priezvisko'];
```

PRÍKLAD 2.38

02/polia38.php

Vypíšme vetu "Vitaj meno priezvisko.", kde meno je prvok poľa `$osoba` s kľúčom `meno` a priezvisko je prvok s kľúčom `priezvisko.`

Keď príklad vyriešime podobne ako v príklade 2.31, teda zadáme

```
echo "Vitaj $osoba['meno'] $osoba['priezvisko'].<br>\n";
```

tak nám prehliadač vypíše chybu, lebo prvok poľa nemôžeme priamo vypísať v reťazci**,** ak je kľúčom reťazec. Teda funguje

```
echo "Prvé meno je $mena[0].";
```

ale nefunguje

```
echo "Volám sa $osoba['meno'].";
```

Môžeme to vyriešiť rozdelením reťazca

```
echo "Volám sa " . $osoba['meno'] . ".";
```

alebo využijeme zložené zátvorky `{ }`, ktoré nám vyhodnotia v nich uzatvorený výraz

```
echo "Volám sa {$osoba['meno']}.";
```

My budeme využívať druhý spôsob (zložené zátvorky), lebo pri zreťazovaní sa často zabúda na písanie bodiek a tým vznikajú chyby.

Správne riešenie nášho príkladu je teda:

```
echo "Vitaj {$osoba['meno']} {$osoba['priezvisko']}.<br>\n";
```

ÚLOHA 2.39

Vytvorte pole `$tovar` zadaním celého poľa s kľúčmi `Made Zigy`, `Sphero BB-8 Star Wars`, `LEGO Boost 17101`, `OZOBOT 2.0 BIT` a hodnotami *38.90*, *100.90*, *141.90*, *65.90*. Potom vypíšte všetky prvky poľa, každé v novom riadku, v tvare "*kľúč - hodnota €*".

2.10 Ladenie a analyzovanie chýb

Každý programátor robí chyby. Je však dôležité naučiť sa čítať chybové správy a potom chyby opravovať. Niekedy aj veľmi jednoduchá chyba spôsobí problém (chybu), no chybová správa PHP interpretera nemusí byť jasná, resp. sa z nej nedá dobre určiť, o akú chybu ide.

Jednou z pomôcok pri hľadaní chýb je aj vizuálne zobrazovanie jednotlivých častí kódu v editore (kľúčové slová, reťazce, čísla, premenné atď.). Ďalšou vizuálnou pomôckou je zobrazovanie párov zátvoriek. Keď sa k niektorej zátvorke nastavíme, tak sa spolu s koncovou (začiatočnou) zátvorkou zvýraznia. Takto môžeme skontrolovať, či máme správne uzátvorkovaný kód. Ak sa nič nezvýrazní, tak zrejme k danej zátvorke chýba dvojica.

Aké sú časté chyby?

Chýbajúca ;

02/chyby01.php

PRÍKLAD 2.40

Spustite v prehliadači súbor `02/chyby01.php`. Čo sa zobrazí?

Zrejme sa vypíše nasledujúca správa:

Parse error**: syntax error, unexpected '$cislo' (T_VARIABLE), expecting ',' or ';' in **D:\...\02\chyby01.php** on line **12

Táto správa hovorí, že na 12. riadku súboru je chyba - že sa neočakávalo `'$cislo'` (premenná), ale očakáva sa čiarka `','` alebo bodkočiarka `';'`. A naozaj nám na 10. riadku chýba bodkočiarka, a preto na 12. riadku neočakával premennú.

```
 9  <?php
10  echo "<h2>Výpis reťazca s premennou</h2>"
11
12  $cislo = 123;
13  echo "Výsledok je $cislo";
14  ?>
```

ÚLOHA 2.41

Opravte chybu v súbore `02/chyby01.php` a opravený kód si pozrite v prehliadači. Potom zmažte `;` na 12 riadku a pozrite sa, aká chyba sa vypíše v prehliadači.

Opäť v kóde chýba `;`. Vo výpise správy

Parse error**: syntax error, unexpected 'echo' (T_ECHO) in **D:\...\02\chyby01.php** on line **13

sa dozvieme, že sa neočakával príkaz `echo`, avšak už chýba časť expecting, čiže nevieme, čo sa namiesto neho očakávalo. To znamená, že pri rovnakej chybe nemusíme dostať rovnakú chybovú správu. V prípadoch, keď sa v chybovej správe uvádza, čo sa neočakávalo, treba hľadať zdroj chyby niekde **pred uvedeným neočakávaným textom**.

Chýbajúce ukončenie alebo začatie reťazca

PRÍKLAD 2.42

Spustime v prehliadači súbor `02/chyby02.php`. Čo sa zobrazí?

02/chyby02.php

Na 10. riadku nemáme ukončený reťazec, čo nám vizuálne zobrazí aj editor:

```
 9  <?php
10  echo "<h2>Výpis reťazcov</h2>;
11
12  echo "Vypisujem reťazec v úvodzovkách.<br>";
13  echo 'Vypisujem reťazec v apostrofoch.<br>';
14
15  ?>
```

Vidíme, že príkaz echo na 12. riadku je sivý, hoci príkazy editor píše modrou farbou a reťazec, ktorý je na 12. riadku nie je sivý, ako zvyknú byť reťazce, ale čierny. Chybová správa, ktorá sa zobrazí, však nič nehovorí o chýbajúcej úvodzovke (`"`):

Parse error: syntax error, unexpected 'Vypisujem' (T_STRING), expecting ',' or ';' in **D:\...\02\chyby02.php** on line **12**

Už vieme, že chybu máme hľadať pred riadkom 12.

ÚLOHA 2.43

Opravte chybu v súbore `chyby02.php` a opravený kód si pozrite v prehliadači. Zmažte iné úvodzovky (`"`) alebo apostrofy (`'`) (skúšajte ich zmazať na začiatku reťazca aj na konci reťazca) a pozorujte, aké chyby sa zobrazujú.

Ak zmažeme koncovú úvodzovku (") reťazca na 12. riadku, zobrazí sa chyba:

Parse error**: syntax error, unexpected end of file, expecting variable (T_VARIABLE) or ${ (T_DOLLAR_OPEN_CURLY_BRACES) or {$ (T_CURLY_OPEN) in **D:\...\02\chyby02.php** on line **18

Táto chyba hovorí, že sa neočakával koniec súboru. Ak sa pozrieme na kód v editore, tak vidíme, že až po koniec je sivý, dokonca aj HTML značky. V takom prípade musíme hľadať, čo sme neukončili. Nemusí to byť len neukončený reťazec, ale aj neukončený cyklus, teda chýba koncová zátvorka, neukončená `if` alebo `else` vetva,...

```
 9  <?php
10  echo "<h2>Výpis reťazcov</h2>";
11
12  echo "Vypisujem reťazec v úvodzovkách.<br>;
13  echo 'Vypisujem reťazec v apostrofoch.<br>';
14
15  ?>
16  </body>
17  </html>
18
```

Zámena porovnania s priradením

Zoberme si kód z príkladu 2.22. Ak by sme v podmienke zamenili porovnanie (`==`) za priradenie (`=`), PHP nevyhlási chybu, ale daný príkaz vykoná a jeho úspešné vykonanie nahradí logickou hodnotou `TRUE`.

```
$den = 'streda';
if ($den = 'sobota' || $den == 'nedela') {
  echo 'Krásny víkend!';
} else {
  echo 'Šup do školy';
}
```

Ak by sme porovnania v podmienke vymenili (`if ('sobota' = $den || 'nedela == $den') { }`), PHP by vyhlásilo chybu.

Ďalšie časté chyby:

- chýbajúca zátvorka (väčšinou koncová) `) }`
- chýbajúci znak `$` pri premennej
- chýbajúci operátor zreťazenia `.`

02/chyby.php

ÚLOHA 2.44

Otvorte súbor `chyby.php` (z priečinku `02`). Nájdite v ňom chyby a opravte ich. Sledujte, čo sa vám vypisuje na stránke.

2.11 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto kapitoly je oboznámenie sa s väčšinou základných štruktúr jazyka PHP. Predpokladáme, že s väčšinou pojmov sa už študenti stretli pri inom programovacom jazyku. Učiteľ to môže využiť a poukázať na podobnosti alebo odlišnosti.

VÝKLAD

Druhá časť je zameraná na:

- vypisovanie údajov pomocou PHP,
- premenné a konštanty,
- prácu s reťazcami,
- operátory,
- vetvenie,
- cykly,
- polia,
- a ladenie chýb.

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

Príkaz echo, reťazce (1)

K úlohe 2.4: Očakávame, že študenti použijú element `<br>` alebo `<p>`.

Komentáre

Poznámka: Treba si dať pozor na to, aby sme nepoužívali HTML komentáre v PHP kóde. Ak chceme do výslednej (vygenerovanej) stránky vložiť HTML komentár, musíme použiť reťazce (ako pri iných HTML elementoch).

Reťazce (2)

`Echo` môžeme zapísať aj ako funkciu s jedným parametrom, teda **`echo(čosi)`**, my však tento spôsob nebudeme používať.

Logické operátory

Prehľadnú tabuľku o prioritách nájdete na
http://php.net/manual/en/language.operators.precedence.php.

Porovnávacie operátory

`echo 3 > 5;` - nevypíše sa nič, lebo `FALSE` sa prekonvertuje na prázdny reťazec

Príkaz if

```
if (podmienka) {
  // príkazy
}
```

Ak sa vykonáva len jeden príkaz, môžeme zložené zátvorky vynechať. My ich však budeme písať vždy, aby to študentov nemýlilo a nezabúdali písať zátvorky. Vyhneme sa tak prípadným chybám. Napríklad ak by sme chceli pridať ďalší príkaz do niektorej časti (`IF`, `ELSE`) a zabudli pridať zložené zátvorky.

Zámer pridať ďalší príkaz do časti `ELSE` - príkaz sa vypíše vždy.	Zámer pridať ďalší príkaz do časti `IF` - príkaz spôsobí chybu "Syntax error".

`$body = 85;`
`if ($body > 60)`
`  echo 'Test si spravil';`
`else`
`  echo 'Test si nespravil';`
`  echo 'Môžeš to skúsiť znovu';`
	`$body = 85;`
	`if ($body > 60)`
	`  echo 'Test si spravil';`
	`  echo "Získal si $body bodov";`
	`else`
	`  echo 'Test si nespravil';`

Cykly

```
for (inicializácia; podmienka; príkaz) {
  // príkazy
}
```

Poznámka: Pri cykle for je najväčšia zmena oproti jazyku Pascal. Treba si dať pozor hlavne na časť **`príkaz`** – tú v Pascale nemusíme riešiť, pretože Pascal automaticky zvyšuje premennú cyklu o jedna. Ak sa študenti stretli s `for` cyklom v jazyku Python (a v iných jazykoch zatiaľ neprogramovali), je pre nich zápis cyklu `for` v PHP neznámy. Je vhodné venovať mu viac pozornosti a vysvetliť študentom význam všetkých troch zložiek v zátvorke (**`inicializácia, podmienka, príkaz`**). Taktiež je potrebné upozorniť ich na existenciu a význam zložených zátvoriek `{ }`, ktoré z Pythonu nepoznajú a nie sú zvyknutí označovať začiatok a koniec bloku **`príkazy`**.

Ak chceme v cykle vykonať viac ako jeden príkaz, musíme ich dať do blokov pomocou zložených zátvoriek `{ }`, podobne ako pri príkaze `if`. Ak však potrebujeme vykonať len jeden príkaz, zložené zátvorky môžeme vynechať. Podobne ako pri `if`.

Polia

Rôzne kľúče (číselné a reťazcové) môžeme vzájomne kombinovať v jednom poli. Zámerne to však neuvádzame. Číselné indexy zvyčajne začínajú nulou (ak pridávame prvky do poľa bez udania indexu). V prípade, že neuvedieme index, hodnote je automaticky priradený index maximum+1. Teda PHP nevyhľadáva prípadné medzery medzi indexami, ale novému prvku

priradí maximum+1. V prípade, že by sme mali definované indexy poľa 0, 1, 3 a vkladáme nový prvok bez udania indexu, potom mu PHP pridelí index 4.

Pole vieme indexovať od ľubovoľného čísla, ale zámerne to v materiáli neuvádzame, lebo s číselnými indexmi poľa nebudeme veľmi pracovať.

Okrem funkcie `foreach()` môžeme pole prechádzať už známymi cyklami `for` prípadne `while`. Tie však môžeme použiť len vtedy, ak sú indexy poľa číselné a spojité. Pri týchto cykloch totiž zvyčajne použijeme funkciu `count()`, ktorá vráti počet prvkov poľa. Týmto zápisom sa však nebudeme venovať, nakoľko budeme väčšinou pracovať s reťazcovými kľúčmi.

Využívanie zložených zátvoriek v echo

Do zložených zátvoriek môžeme uzavrieť jednoduchú premennú, prvok poľa alebo vlastnosť objektu (ak využívame objektové programovanie). Znak `$` musí nasledovať hneď po `{`, inak výpis nebude korektný. Prvok poľa s reťazcovým kľúčom sa korektne vypíše len keď je uzavretý v zložených zátvorkách, teda: `echo "Volám sa {$osoba['meno']}.";`

Zložené zátvorky v príkaze `echo` môžeme napr. využiť, keď chceme odlíšiť názov premennej od okolitého textu, napríklad, ak by sme chceli vygenerovať vetu z nasledujúcich premenných

```
$opakuj = 5;
$aky = 'dvoj';
echo "Urob $opakuj-krát $akyskok.<br>\n";
```

tak premennú `$opakuj` rozpozná, ale premennú `$aky` nie (dostaneme upozornenie, že premenná `akyskok` nie je definovaná). Preto ju musíme uzavrieť do zložených zátvoriek:

```
$opakuj = 5;
$aky = 'dvoj';
echo "Urob $opakuj-krát {$aky}skok.<br>\n";
```

Ladenie a analyzovanie chýb

K príkladu 2.42: Tým, že sme reťazec na 10. riadku neuzavreli, pokračoval vlastne ďalej až po najbližšiu `"`. A za ňou PHP interpreter našiel slovo *Vypisujem*, ktorému nerozumie, preto v chybovej správe píše, že ho neočakáva.

3 PHP - SPRACOVANIE FORMULÁROV (1)

V tejto časti si ukážeme, ako pomocou formulárov môžeme získať údaje od používateľov stránok a ako s týmito údajmi ďalej pracovať.

3.1 Odoslanie formulára, tlačidlo

Pripomeňme si, že aby sme mohli odoslať formulár, musí v ňom byť tlačidlo typu submit

03/formular01.php

Vytvorme nasledujúci jednoduchý formulár s tlačidlom na odoslanie formulára. Formulár sa bude odosielať metódou `POST`.

```
<form method="post">
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

ZAPAMÄTAJTE SI

Metóda POST:

- údaje z formulára sú posielané cez HTTP odpoveď na požiadavku v asociatívnom poli $_POST (narozdiel od metódy GET, kde sa údaje získavajú z URL adresy)
- môžu sa posielať aj citlivé údaje, lebo nie sú viditeľné
- dĺžka (veľkosť) údajov je obmedzená len nastaveniami webového servera

Čo sa stane po odoslaní formulára? Údaje z formulára sa pošlú súboru, ktorý je špecifikovaný v atribúte `action` pre `<form>`. Ak tento atribút nie je uvedený, tak sa pošlú tomu istému súboru, teda súboru, v ktorom je formulár. My budeme využívať práve takéto spracovanie formulárov.

ÚLOHA 3.1

Zobrazme výslednú stránku v prehliadači a otestujme. Čo sa zmení na stránke po odoslaní formulára (kliknutí na tlačidlo)?

Po odoslaní formulára metódou `POST` sa naplní asociatívne pole **`$_POST`** všetkými zadanými hodnotami z formulára (vrátane hodnoty odosielacieho tlačidla). Pole `$_POST` má tvar [kľúč1 => hodnota1, kľúč2 => hodnota2, kľúč3 => hodnota3, ...], kde kľúčmi sú názvy prvkov formulára (atribúty `name`) a hodnotami sú odoslané údaje z príslušných prvkov formulára.

```
<input type="submit" name="tlacidlo"
value="Odošli údaje">
```

=>

```
$_POST['tlacidlo']
```

PRÍKLAD 3.2

03/formular02a.php

03/formular02b.php

Vypíšme hodnotu odoslaného tlačidla. Všímajme si, čo sa zobrazí pred a po odoslaní formulára (kliknutím na tlačidlo). Výpis umiestnime podľa toho, či chceme mať výpis pred formulárom alebo za.

Ak stránku (súbor) s formulárom viackrát upravujeme a znovu zobrazujeme v prehliadači, odporúčame nepoužívať tlačidlo Obnoviť ale kliknúť do panela s adresou a potvrdiť klávesom Enter.

```
<?php
echo $_POST['tlacidlo'];
?>
<form method="post">
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

```
<form method="post">
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
<?php
echo $_POST['tlacidlo'];
?>
```

Pri prvom zobrazení stránky sa nám zobrazí upozornenie, že neexistuje kľúč `tlacidlo`. Tým sa myslí kľúč v poli, v našom prípade je to kľúč `tlacidlo` z poľa `$_POST`. Po odoslaní formulára sa upozornenie nezobrazí.

Spôsobené je to tým, že pri prvom zobrazení stránky je ešte pole `$_POST` prázdne, teda sa snažíme vypísať hodnotu neexistujúceho prvku poľa. Aby sme predišli zobrazeniu takejto "škaredej" správy, pred samotným výpisom premennej (prvku poľa) najprv otestujeme, či taká premenná existuje.

Na overenie existencie premennej, resp. prvku poľa, nám slúži funkcia **`isset()`**. Funkcia vráti hodnotu `TRUE` alebo `FALSE` podľa toho, či daná premenná (index poľa) existuje.

03/formular03.php

PRÍKLAD 3.3

Doplňme do predchádzajúceho príkladu kontrolu existencie premennej, a tým odstráňme zobrazenie upozornenia.

```
<?php
if (isset($_POST['tlacidlo'])) {
  echo $_POST['tlacidlo'];
  // echo 'Stlačil si tlačidlo';
}
?>
<form method="post">
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

3.2 Textové pole - INPUT a TEXTAREA

03/formular04.php

PRÍKLAD 3.4

Doplňme do formulára jednoduché textové pole, do ktorého bude používateľ zadávať údaje. Tie následne (po odoslaní formulára) vypíšeme.

```php
<?php
if (isset($_POST['meno'])) {
  echo $_POST['meno'];
  // echo 'meno: ' . $_POST['meno'];
}
?>
<form method="post">
  <label for="meno">Meno:</label> <input type="text" name="meno"
id="meno">
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

Rovnako by sme pracovali aj s položkou s heslom (pole typu password).

V prípade, že používateľ nezadá žiadne meno a formulár odošle, nevypíše sa nič, ani chyba, resp. upozornenie. Dôvodom je to, že PHP interpreter nastaví ako hodnotu textového poľa prázdny reťazec a ten priradí do `$_POST['meno']`. Teda funkcia `isset()` vráti hodnotu `TRUE`. Ak do príkazu `echo` pridáme nejaký reťazec, zistíme, že PHP danú časť kódu naozaj vykoná (napr. `echo 'meno: ' . $_POST['meno'];`).

PRÍKLAD 3.5

Upravme formulár tak, že ak používateľ meno nezadá, vypíše sa správa *Nezadal si meno!*

03/formular05.php

```php
if (isset($_POST['meno']) && ($_POST['meno'] != '')) {
  echo 'meno: ' . $_POST['meno'];
} else {
  echo 'Nezadal si meno!';
}
```

Pri odoslaní formulára toto riešenie funguje, ale bohužiaľ, správa sa zobrazí aj pri prvom zobrazení stránky (pred odoslaním formulára). A to nie je dobre.

Problémom je, že naša kontrola mena sa vykoná vždy pri zobrazení stránky, teda aj keď nie je odoslaný formulár. Preto musíme náš kód doplniť o ďalšiu podmienku, ktorá by sa mala stať základnou pri spracovaní formulára - kontrola stlačenia odosielacieho tlačidla (teda existencie premennej `$_POST['tlacidlo']`) - `if (isset($_POST['tlacidlo'])) { }`. A všetky ďalšie kontroly týkajúce sa zadaných hodnôt do formulára by sa mali vykonať len keď je podmienka splnená.

PRÍKLAD 3.6

Vylepšime kontrolu formulára z príkladu 3.5.

03/formular06.php

```php
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && ($_POST['meno'] != '')) {
    echo 'meno: ' . $_POST['meno'];
  } else {
    echo 'Nezadal si meno!';
  }
}
```

Ak sme do textového poľa nezadali nič, program nás na to síce upozorní, ale len vtedy, ak odošleme formulár. Pri prvom zobrazení stránky sa zobrazí len formulár bez upozornenia.

V prípade, že vo formulári nepoužijeme jednoriadkové textové pole, ale viacriadkové textové pole pomocou elementu `<textarea>`, spracovanie vstupu bude rovnaké.

03/formular07.php

PRÍKLAD 3.7

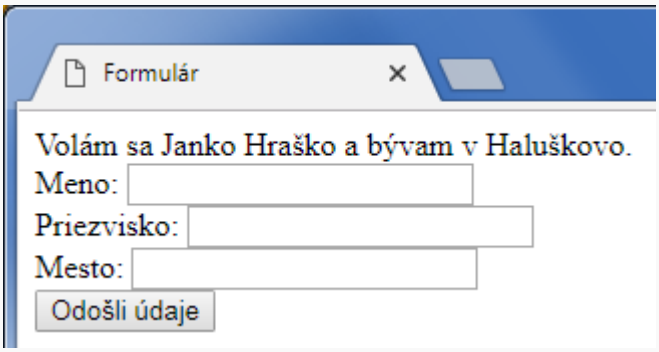
V príklade 3.6 zmeňme položku meno na adresa a prvok input na textarea.

```
<?php
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['adresa']) && ($_POST['adresa'] != '')) {
    echo 'adresa: ' . $_POST['adresa'];
  } else {
    echo 'Nezadal si adresu!';
  }
}
?>

<form method="post">
  <label for="adresa">Adresa:</label>
  <textarea name="adresa" id="adresa"></textarea><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

Vidíme, že vo funkčnosti spracovania formulára sme nič nezmenili, len sme vymenili texty, ktoré sa majú vypísať a zmenili sme názvy položiek.

ÚLOHA 3.8

Vytvorte formulár s tromi textovými poľami pre meno, priezvisko a mesto. Ak nie sú zadané všetky údaje, vypíše sa správa *Nezadal si všetky údaje!*, inak sa vypíše správa *Volám sa … a bývam v … .* (ako na nasledujúcom obrázku)



ÚLOHA 3.9

Vytvorte formulár s dvoma textovými poľami pre dve strany obdĺžnika. Ak nie sú zadané všetky údaje, vypíše sa správa *Nezadal si obe strany obdĺžnika!*, inak sa vypíše správa *Obdĺžnik so stranami ... , ... má obvod … a obsah ... .*

3.3 Výberová ponuka - SELECT

Pri výberovej ponuke (element `<select>`) zvyčajne vyberáme jednu hodnotu z viacerých. Hodnota celého elementu po odoslaní formulára sa bude rovnať hodnote vybranej položky - hodnote atribútu `value`.

PRÍKLAD 3.10

03/formular10.php

Vložme do stránky výberovú ponuku a po odoslaní zobrazme jej hodnotu. Všímajme si hodnoty (texty) vo výberovej ponuke a hodnoty, ktoré sa naozaj vypíšu po spracovaní formulára.

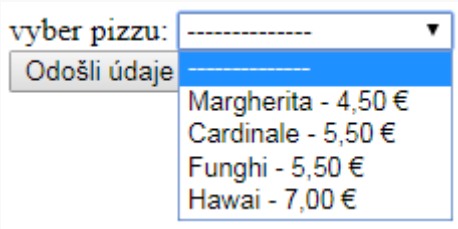
```
<?php
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['vyber']) && ($_POST['vyber'] != '')) {
    echo 'vyber: ' . $_POST['vyber'];
  } else {
    echo 'Nič si nevybral!';
  }
}
?>

<form method="post">
  <label for="vyber">vyber:</label>
     <select name="vyber" id="vyber">
       <option value="">0</option>
       <option value="10">1</option>
       <option value="20">2</option>
       <option value="30">3</option>
       <option value="40">4</option>
       <option value="50">5</option>
     </select><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

Zapamätajte si: Pri spracovávaní hodnoty (value) si musíme uvedomiť, že všetky hodnoty sú reťazce, aj keď sú v nich uložené čísla. Ak sú to však korektné čísla, PHP si ich vie prerobiť (pretypovať), teda môžeme s nimi robiť matematické operácie.

Vo formulári sme chceli, aby používateľ mohol zvolil jednu z hodnôt 1 až 5. Ak by sme mali len tieto hodnoty v elementoch `<option>`, bola by automaticky zobrazená hodnota 1 (prvá v poradí). Ak by bola po odoslaní formulára vo výbere 1 (hodnota 10), tak by sme nevedeli rozlíšiť, či používateľ naozaj vybral položku 1 alebo odoslal formulár bez toho, aby niečo zvolil. Preto je vhodné, aby vo výberovej ponuke bola "prázdna" hodnota. My sme do ponuky zaradili aj položku `<option value="">0</option>`.

ÚLOHA 3.11

Vytvorte formulár, v ktorom bude výberová ponuka rôznych druhov pizze. V ponuke bude názov pizze spolu s jej cenou a po spracovaní formulára sa vypíše len zvolený názov pizze, prípadne správa, že sa nič nezvolilo. Môžete využiť tieto informácie: *Margherita - 4,50 €, Cardinale - 5,50 €, Funghi - 5,50 €, Hawai - 7,00 €.* Výberová ponuka môže vyzerať nasledovne:



3.4 Prepínač - RADIOBUTTON

Inou možnosťou, ako na stránke vyberať jednu hodnotu z viacerých, sú prepínače - radiobuttony. Z HTML vieme, že ak chceme vyberať z viacerých prepínačov, musia mať rovnaké hodnoty atribútu `name`.

Samotné spracovanie prepínačov je rovnaké ako pri výberovej ponuke - hodnota zvoleného prepínača bude hodnotou celej skupiny prepínačov.

Pri odoslaní formulára sa prepínače správajú trochu inak ako pri výberovej ponuke. Ak totiž neoznačíme žiaden radiobutton, v poli `$_POST` nevznikne žiadny kľúč prislúchajúci danej skupine prepínačov (ani s prázdnym reťazcom). Ten vznikne jedine vtedy, ak označíme niektorý radiobutton. Vďaka tomu môžeme podmienku zjednodušiť na: `if (isset($_POST['vyber'])) { }`.

PRÍKLAD 3.12

03/formular12.php

Nahraďme výberovú ponuku z príkladu 3.10 prepínačmi.

```
<?php
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['vyber'])) {
    echo 'vyber: ' . $_POST['vyber'];
  } else {
    echo 'Nič si nevybral!';
  }
}
?>


<form method="post">
  <label for="vyber1">1</label> <input type="radio" name="vyber"
value="10" id="vyber1"><br>
```

```
  <label for="vyber2">2</label> <input type="radio" name="vyber"
value="20" id="vyber2"><br>
  <label for="vyber3">3</label> <input type="radio" name="vyber"
value="30" id="vyber3"><br>
  <label for="vyber4">4</label> <input type="radio" name="vyber"
value="40" id="vyber4"><br>
  <label for="vyber5">5</label> <input type="radio" name="vyber"
value="50" id="vyber5"><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

Vidíme, že spracovanie formulára je takmer rovnaké ako pri výberovej ponuke, len sme zjednodušili podmienku.

ÚLOHA 3.13

Upravte riešenie úlohy 3.11 tak, aby ste namiesto výberovej ponuky využili prepínače. Formulár môže vyzerať nasledovne:

vybral si pizzu: Cardinale
- Margherita - 4,50 €
- Cardinale - 5,50 €
- Funghi - 5,50 €
- Hawai - 7,00 €

Odošli údaje

3.5 Začiarkávacie políčko - CHECKBOX

Aby mohol používateľ označiť viacero hodnôt súčasne, používame začiarkávacie políčka (checkboxy). Ich spracovanie je v podstate rovnaké ako spracovanie prepínačov, ale každé políčko musíme kontrolovať samostatne. Každé začiarkávacie políčko má inú hodnotu atribútu `name`. Hodnoty atribútov `value` môžu byť rovnaké.

Začiarkávacie políčka sa správajú rovnako ako prepínače - ak ich neoznačíme, po odoslaní formulára nevznikne v poli `$_POST` príslušný kľúč.

Začiarkávacie políčka zvyčajne používame vtedy, ak používateľ nemusí zvoliť žiadnu hodnotu, alebo ich môže označiť viacero. Ak by sme chceli kontrolovať, či označil aspoň jednu (prípadne aspoň niekoľko), museli by sme si to spočítať, a teda kontrolovať všetky políčka, či boli/neboli označené.

PRÍKLAD 3.14

Nahraďme prepínače z príkladu 3.12 začiarkávacími políčkami. Využijeme aj predchádzajúci PHP kód, len ho viackrát skopírujeme, teda pre každé políčko vypíšeme, či bolo/nebolo vybraté.

03/formular14.php

```
<?php
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['vyber1'])) {
     echo "vyber 1: {$_POST['vyber1']} <br>";
  } else {
     echo 'Nezadal si vyber 1!<br>';
  }
  if (isset($_POST['vyber2'])) {
     echo "vyber 2: {$_POST['vyber2']} <br>";
  } else {
     echo 'Nezadal si vyber 2!<br>';
  }
  if (isset($_POST['vyber3'])) {
     echo "vyber 3: {$_POST['vyber3']} <br>";
  } else {
     echo 'Nezadal si vyber 3!<br>';
  }
  if (isset($_POST['vyber4'])) {
     echo "vyber 4: {$_POST['vyber4']} <br>";
  } else {
     echo 'Nezadal si vyber 4!<br>';
  }
  if (isset($_POST['vyber5'])) {
     echo "vyber 5: {$_POST['vyber5']} <br>";
  } else {
     echo 'Nezadal si vyber 5!<br>';
  }
}
?>

<form method="post">
  <label for="vyber1">1</label> <input type="checkbox" name="vyber1"
value="10" id="vyber1"><br>
  <label for="vyber2">2</label> <input type="checkbox" name="vyber2"
value="20" id="vyber2"><br>
  <label for="vyber3">3</label> <input type="checkbox" name="vyber3"
value="30" id="vyber3"><br>
  <label for="vyber4">4</label> <input type="checkbox" name="vyber4"
value="40" id="vyber4"><br>
  <label for="vyber5">5</label> <input type="checkbox" name="vyber5"
value="50" id="vyber5"><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

03/formular15.php

PRÍKLAD 3.15

Zjednodušme príklad 3.14 tak, že budeme vypisovať len tie začiarkávacie políčka, ktoré používateľ označil. Teda vynecháme `else` vetvy o nezadaní príslušného políčka.

```
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['vyber1'])) {
    echo "vyber 1: {$_POST['vyber1']} <br>";
  }
  if (isset($_POST['vyber2'])) {
    echo "vyber 2: {$_POST['vyber2']} <br>";
  }
```

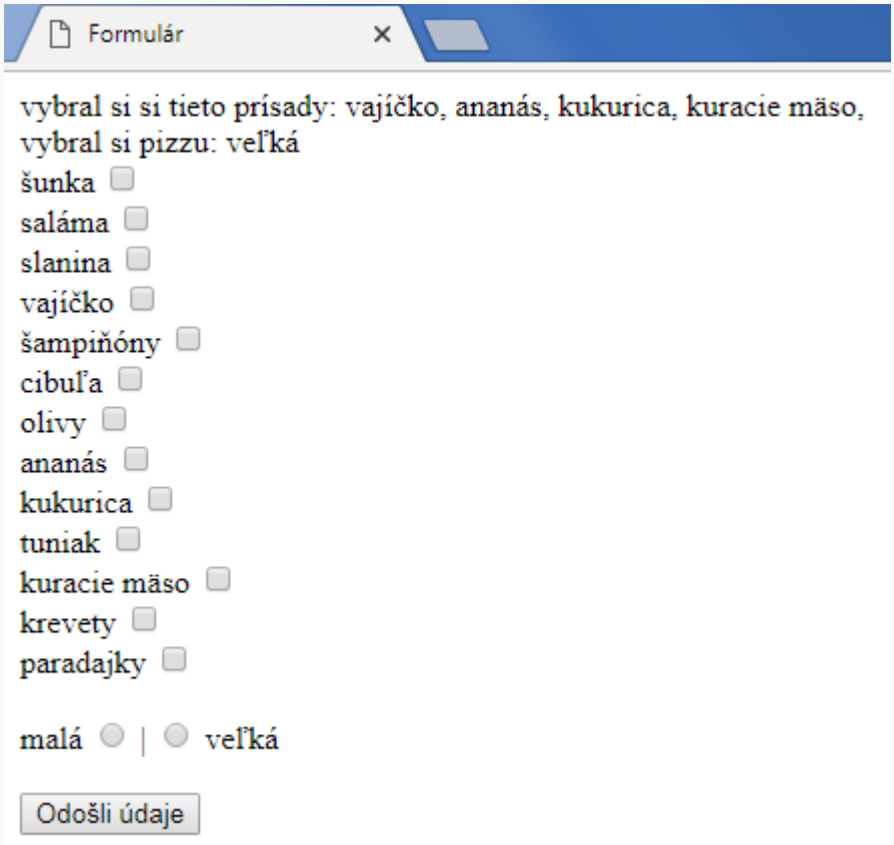
```
  if (isset($_POST['vyber3'])) {
    echo "vyber 3: {$_POST['vyber3']} <br>";
  }
  if (isset($_POST['vyber4'])) {
    echo "vyber 4: {$_POST['vyber4']} <br>";
  }
  if (isset($_POST['vyber5'])) {
    echo "vyber 5: {$_POST['vyber5']} <br>";
  }
}
```

ÚLOHA 3.16

Vytvorte formulár, pomocou ktorého sa bude dať zisťovať, v akých programovacích jazykoch sa používatelia učili programovať. Napr. Pascal, Python, C++, C#, Java, PHP, Imagine, Scratch,… Po spracovaní formulára sa vypíšu len tie jazyky, ktoré používateľ zvolil.

ÚLOHA 3.17

Vytvorte formulár, pomocou ktorého bude možné nakombinovať si vlastnú pizzu. Budú sa môcť zvoliť prísady, a tiež sa bude dať zvoliť, či to má byť malá alebo veľká pizza. Po odoslaní formulára sa vypíšu zvolené údaje. Formulár po odoslaní môže vyzerať nasledovne:



3.6 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto kapitoly je oboznámenie so spracovaním najpoužívanejších formulárových elementov.

VÝKLAD

Táto časť je zameraná na:

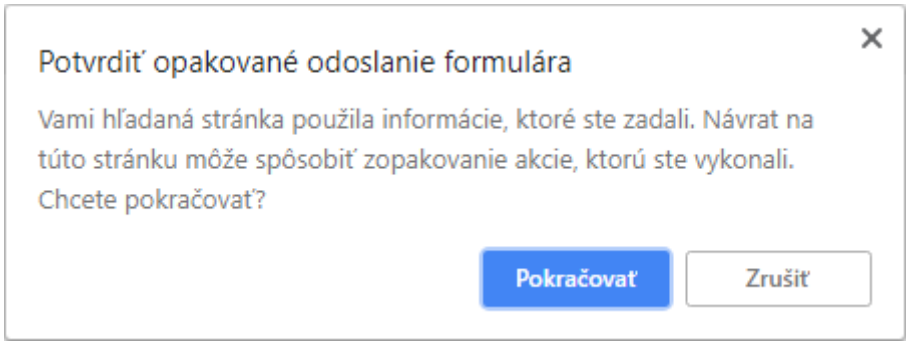
- odosielanie formulára,
- a na spracovanie väčšiny formulárových elementov.

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

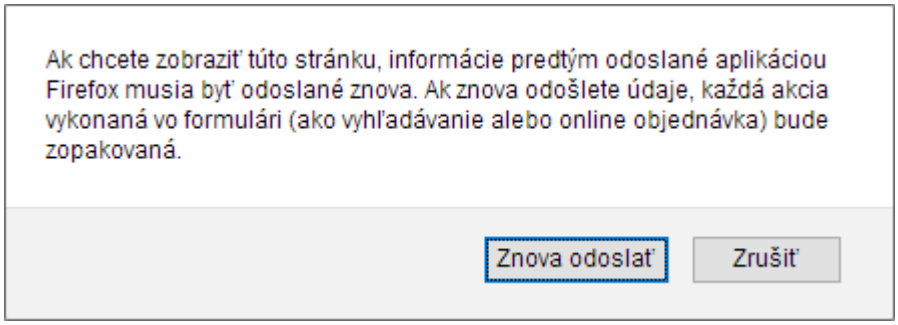
Študentov treba upozorniť, že formulár musí vždy obsahovať tlačidlo typu submit, aby sa dal odoslať.

K načítaniu stránky, resp. obnoveniu stránky s formulárom:

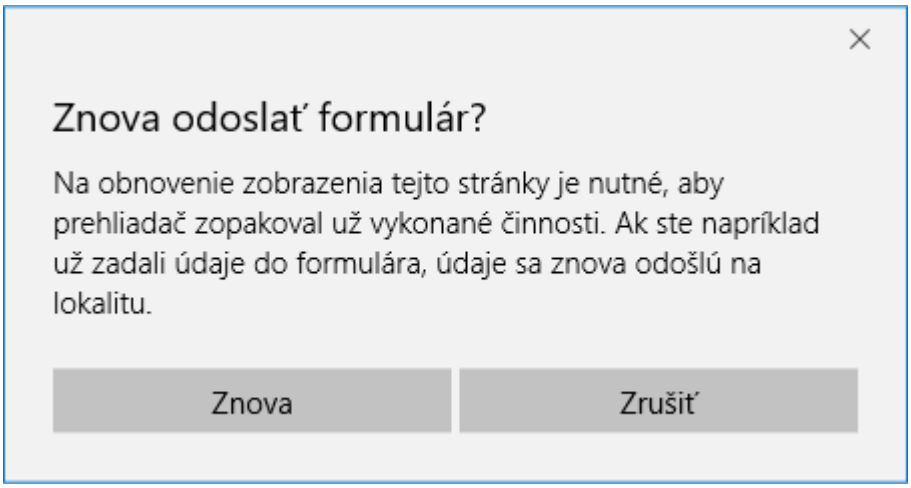
Treba upozorniť študentov, že načítať tú istú stránku s formulárom, keď klikneme na tlačidlo *Obnoviť (F5),* je iné ako keď klikneme do panela s adresou a potvrdíme klávesom *Enter*. A to hlavne vtedy, keď sme formulár odoslali. Prehliadač si totiž pamätá, či stránka dostala nejaké vstupy (z odoslaného formulára). Ak áno, tak pri obnovení stránky cez tlačidlo *Obnoviť (F5)* sa prehliadač spýta, či sa majú znovu odoslať údaje z formulára.



Reakcia prehliadača Google Chrome po stlačení tlačidla *Obnoviť (F5)*.



Reakcia prehliadača Mozilla Firefox po stlačení tlačidla *Obnoviť (F5)*.



Reakcia prehliadača Microsoft Edge po stlačení tlačidla *Obnoviť (F5)*.

Preto, ak chceme znovu načítať stránku v prehliadači, odporúčame **nepoužívať** tlačidlo *Obnoviť (F5)*, ale kliknúť do panela s adresou a potvrdiť klávesom *Enter*.

K spracovaniu formulára:

PHP kód, ktorý spracováva formulár môže byť umiestnený v inom súbore ako HTML formulár, alebo môže byť v rovnakom súbore ako HTML formulár. V učebnici budeme využívať len druhý prístup (formulár a PHP kód v jednom súbore). Tento prístup nám umožňuje lepšie reagovať na zadané údaje a podľa potreby zobrazovať/skrývať formulár, prípadne zobrazovať pri formulári chybové správy. Pri prvom prístupe (HTML formulár a PHP kód v samostatných súboroch) by sa napr. chybové správy pri formulári zobrazovali komplikovanejším spôsobom. Museli by sme si ich nejako posielať alebo ukladať - a to je niečo, čo sa ešte v učebnici neukazovalo.

`$_POST` je superglobálna premenná, ktorá je vždy prístupná (aj keď je prázdna). Môžeme k nej hocikedy pristupovať, a to z ľubovoľnej vlastnej funkcie, triedy či súboru.

K časti 3.1:

Pre tieto ukážky je jedno, či použijeme tlačidlo cez element `<input>` alebo cez element `<button>`.

K úlohe 3.1:

Treba upozorniť študentov, že sa možno zdá, že sa nič nedeje. V skutočnosti sa odošle formulár tomu istému súboru a ten sa znovu načíta. Treba sledovať ikonu obnovenia stránky.

K elementu SELECT:

Ak atribút value neexistuje, hodnotou bude text, ktorý sa pri danej položke zobrazuje, teda ak sa hodnoty a texty zhodujú, nemuseli by sa atribúty `value` uvádzať.

4 PHP - ZÁKLADNÁ SYNTAX A ŠTRUKTÚRY (2)

V tejto časti doplníme základné informácie o jazyku PHP. Ukážeme si prácu s náhodnými číslami, dátumom a časom, naučíme sa vytvárať vlastné funkcie a využívať viaceré PHP vsuvky.

4.1 Náhodné čísla

Ak budeme potrebovať vygenerovať náhodné číslo, môžeme použiť funkciu `rand()`. Funkcia generuje náhodné celé číslo z intervalu <0, `getrandmax()`>. Ak potrebujeme upraviť rozsah, použijeme `rand(min, max)`. Vtedy funkcia vráti náhodné **celé číslo** z intervalu <min, max>.

PRÍKLAD 4.1

Vypíšme 5 náhodných čísel pod seba. Ďalej vypíšme náhodné číslo z intervalu <5, 20>. Nakoniec vypíšme náhodné číslo z intervalu <10, 100>, pričom to môže byť len násobok čísla 5.

04/nahodne.php

```
<?php
echo "<h1>Náhodné čísla</h1>\n";
for ($i = 1; $i <= 5; $i++) {
  echo rand() . "<br>\n";
}
echo "<h1>Náhodné číslo z intervalu &lt;5, 20&gt;</h1>\n";
echo rand(5, 20) . "<br>\n";
echo "<h1>Náhodné číslo z intervalu &lt;10, 100&gt;, ale len násobky
5</h1>\n";
$a = rand(1, 19) * 5 + 5;
echo "$a<br>\n";
?>
```

POZNÁMKA

Okrem funkcie `rand()` môžeme na generovanie náhodných čísel použiť aj funkciu **`mt_rand()`**. Táto funkcia generuje čísla pomocou iného algoritmu, vracia lepšie výsledky a je 4x rýchlejšia ako `rand()`. V prípade, že chceme použiť `mt_rand(min, max)`, musíme sa uistiť, že `min` <= `max`, inak sa vypíše upozornenie.

4.2 Dátum a čas

Na webových stránkach sa často vyskytujú dátumové informácie. Napr. informácia o aktuálnom dátume, informácia o poslednom prihlásení, o tom, kedy bude dostupný tovar, v ktorých dňoch a časoch sa premieta daný film atď. Tieto informácie môžu byť zapísané rôznymi spôsobmi.

ÚLOHA 4.2

Prezrite si niekoľko webových stránok a nájdite na nich dátumové, prípadne časové údaje. Akým spôsobom (v akom formáte) boli zapísané?

Aby sme zistili, či je daný dátum platný, použijeme funkciu `checkdate(mesiac, den, rok)`. Funkcia vracia `TRUE` alebo `FALSE`.

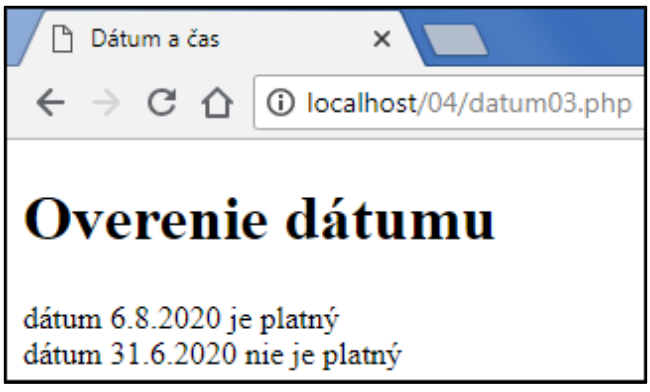
04/datum03.php

PRÍKLAD 4.3

Overme platnosť dátumov 6.8.2020 a 31.6.2020.

```
echo "dátum 6.8.2020 ";
if (checkdate(8,6,2020)) {
  echo "je platný<br>\n";
} else {
  echo "nie je platný<br>\n";
}
echo "dátum 31.6.2020 ";
if (checkdate(6,31,2020)) {
  echo "je platný<br>\n";
} else {
  echo "nie je platný<br>\n";
}
```

Tento kód vypíše:



ÚLOHA 4.4

Vygenerujte 10 náhodných dátumov (pre deň náhodné číslo od 1 do 31, pre mesiac náhodné číslo od 1 do 12 a pre rok náhodné číslo od 1800 po 2100) a overte a vypíšte ich platnosť.

Na formátovanie dátumu a/alebo času použijeme funkciu `date(format)`. Parameter `format` je povinný a definuje, v akom formáte funkcia vráti (ako reťazec) **aktuálny dátum a čas**.

Tabuľka 2 Použitie niektorých formátovacích parametrov pre funkciu date()

príkaz	vypíše sa
`date("j.n.Y")`	6.8.2020
`date("j. F Y, h:i A")`	6. August 2020, 08:24 AM
`date("G:i:s")`	8:26:19
`if (date('G') <= 6) { echo 'Dobré ráno '; }`	
`if (date('n') == 2) { echo 'Február '; }`	

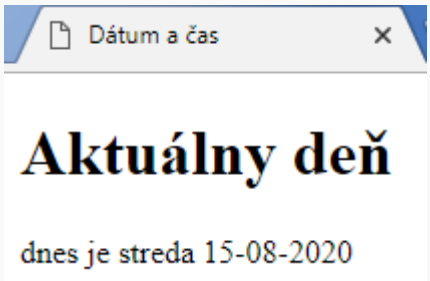
Kompletný zoznam znakov, ktoré sú rozpoznávané v reťazcovom parametri `format`, sú v súbore `date-format.html` (prevzaté z manuálu PHP[6]). V prípade, že v reťazci `format` uvedieme znak, ktorý funkcia `date()` nerozpozná, vypíše ho ako obyčajný znak (napr. bodka, dvojbodka, medzera, čiarka, ...).

POZNÁMKA

Pri práci s dátumom nemusí mať webserver nastavenú tzv. časovú zónu. V prípade, že časové funkcie nedávajú správne hodnoty, resp. dávajú upozornenie (Warning), nastavíme časovú zónu príkazom `date_default_timezone_set('Europe/Bratislava');` pred použitím príkazu `date()`.

ÚLOHA 4.5

Vypíšte aktuálny dátum v tvare “deň-mesiac-rok”, kde deň a mesiac majú vždy dve čísla, napr.: 15-08-2020. K dátumu pripíšte, aký deň v týždni to je, vypíšte to po slovensky. Napr.:



Návod: Môžete využiť formát w a pole s názvami dní v správnom poradí.

[6] www.php.net

ÚLOHA 4.6

Vypíšte aktuálny dátum tak, aby sa mesiac vypísal v slovenčine.

4.3 Vlastné funkcie

Definovanie vlastných funkcií existuje vo väčšine programovacích jazykov a PHP nie je výnimkou. Funkcie nám umožňujú znovu využiť časť kódu pre rovnakú akciu a aj sprehľadňujú a zjednodušujú kód.

Vlastné funkcie definujeme pomocou nasledujúcej štruktúry:

```
function menoFunkcie(){
  // telo funkcie
}
```

Názov funkcie by mal byť krátky a výstižný. Nesmie mať rovnaký názov ako už existujúce PHP funkcie. V jej názve môžeme použiť len písmená, číslice a podčiarkovník. Názov funkcie nemôže začínať číslicou.

Funkciu definujeme v rámci PHP kódu, ale skôr, ako ju potrebujeme zavolať.

PRÍKLAD 4.7

04/vypis01.php

Zadefinujme funkciu `vypis()`, ktorá bude vypisovať čísla od 1 do 10 oddelené medzerou. Potom zavolajme funkciu `vypis()`.

```
<?php
function vypis() {
  for ($i = 1; $i <= 10; $i++) {
    echo "$i ";
  }
}

vypis();
?>
```

Všimnite si, že funkciu sme zavolali aj s prázdnymi zátvorkami. Keby sme ju zapísali bez zátvoriek, tak by ju interpreter považoval za konštantu a ak taká konštanta neexistuje, vypísala by sa chyba.

Parametre funkcií

Väčšina funkcií, ktoré budeme programovať, bude potrebovať parametre. Parametre uvádzame v okrúhlych zátvorkách za názvom funkcie.

```
function menoFunkcie(parametre){
  // telo funkcie
}
```

PRÍKLAD 4.8

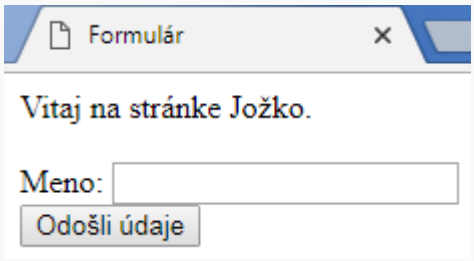
04/vypis02.php

Upravme funkciu `vypis()` z predchádzajúceho príkladu tak, aby vypisovala všetky čísla od 1 po zadané číslo. Potom vypíšme všetky čísla od 1 do 15 a do ďalšieho riadku od 1 do 5.

```
function vypis($do) {
  for ($i = 1; $i <= $do; $i++) {
    echo "$i ";
  }
}

vypis(15);
echo "<br>\n";
vypis(5);
```

ÚLOHA 4.9

Napíšte funkciu `privitaj()`, ktorá ako parameter dostane *meno* a vypíše vetu "Vitaj na stránke *meno*.". Vytvorte formulár s textovým poľom podobne ako v príklade 3.6. Po odoslaní formulára zavolajte funkciu `privitaj()` so zadaným menom. Výsledná stránka môže vyzerať nasledovne:



Ak má funkcia viac parametrov, tak ich oddeľujeme čiarkou. Teda zapisujeme:

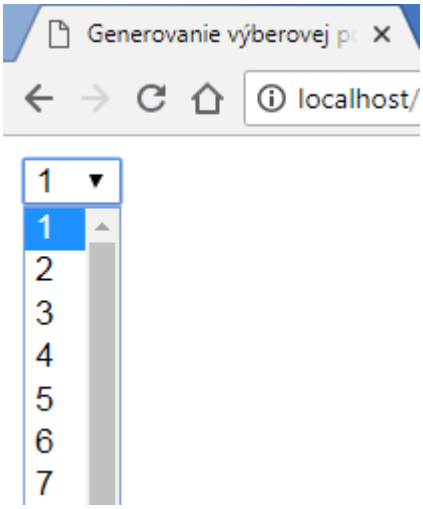
- `function moja_funkcia($x) { }`
- `function moja_funkcia($x, $y) { }`
- `function moja_funkcia($x, $y, $z) { }`

PRÍKLAD 4.10

04/vypis03.php

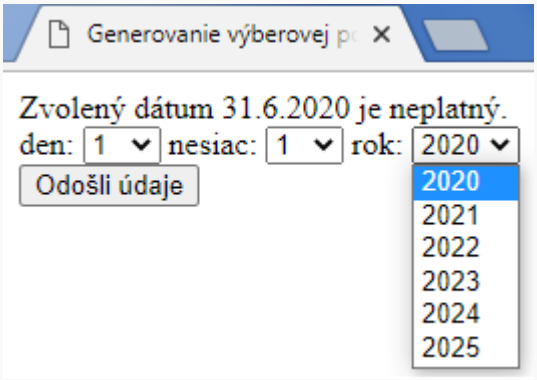
Napíšme funkciu `vypis_poloziek($od, $do)`, ktorá pre formulárový prvok `<select>` vytvorí jednotlivé číselné položky (`option`), ktoré budú začínať číslom `$od` a končiť číslom `$do`. Potom vytvoríme formulár s prvkom `<select>`, ktorého meno je *den* a položky bude mať od 1 do 31.

```php
<?php
function vypis_poloziek($od, $do) {
  for ($i = $od; $i <= $do; $i++) {
    echo "<option value='$i'>$i</option>\n";
  }
}

echo "<form>\n";
echo "<select name='den'>\n";
vypis_poloziek(1,31);
echo "</select>\n";
echo "</form>\n";
?>
```

zdrojový HTML kód	výstup v prehliadači
<form> <select name='den'> <option value='1'>1</option> <option value='2'>2</option> <option value='3'>3</option> <option value='4'>4</option> <option value='5'>5</option> ... <option value='29'>29</option> <option value='30'>30</option> <option value='31'>31</option> </select> </form>	

ÚLOHA 4.11

Napíšte skript, v ktorom vytvoríte formulár s tromi položkami `<select>` pre dátum, mesiac a rok (rok od aktuálneho roka + 5 ďalších rokov) a odosielacím tlačidlom. Po odoslaní formulára vypíšte vybraný dátum a aj to, či je tento dátum platný. Výsledná stránka môže vyzerať nasledovne:



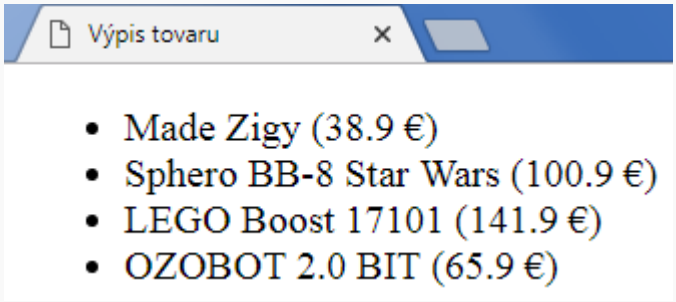
ÚLOHA 4.12

Napíšte funkciu `vypis_tovaru($pole)`, ktorá očakáva nasledujúcu štruktúru vstupného poľa: `['nazov1' => cena1, 'nazov2' => cena2, 'nazov3' => cena3, ...]`.

Funkcia vypíše jednotlivé prvky poľa ako zoznam s odrážkami v tvare "názov tovaru (cena)". Pomocou tejto funkcie vypíšte pole

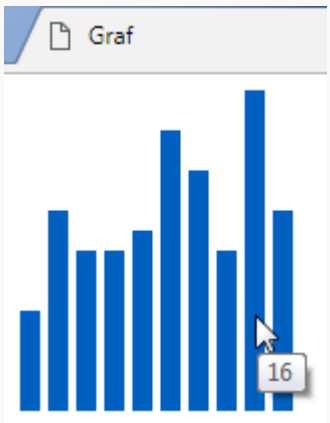
```
$tovar = ['Made Zigy' => 38.90, 'Sphero BB-8 Star Wars' => 100.90,
'LEGO Boost 17101' => 141.90, 'OZOBOT 2.0 BIT' => 65.90];.
```

Výsledná stránka by mohla vyzerať nasledovne:



ÚLOHA 4.13

Napíšte funkciu `stlpcovy_graf()`, ktorá ako parameter dostane číselné pole a hodnoty poľa zobrazí ako stĺpcový graf. Vedľa seba budú obrázky (element `<img>`), ktorých výška bude podľa príslušnej hodnoty z poľa (ak treba môže sa vynásobiť 10). Elementom `<img>`) treba nastaviť vlastnosť `title`, ktorá bude reprezentovať danú hodnotu stĺpca. Napr.: pre pole [5, 10, 8, 8, 9, 14, 12, 8, 16, 10] sa vytvorí nasledujúci graf:



Funkcie s návratovou hodnotou

Občas by sme potrebovali, aby funkcia nič nevypisovala, ale napríklad niečo vypočítala a vrátila výsledok. PHP na vrátenie hodnoty využíva `return`, podobne ako aj niektoré iné programovacie jazyky.

ZAPAMÄTAJTE SI

Príkazom `return` sa sa vykonávanie funkcie končí, aj keby bol za ním ešte iný príkaz.

04/mocnina3.php

PRÍKLAD 4.14

Napíšme funkciu `mocnina3()`, ktorá vráti tretiu mocninu zadaného čísla. Potom vypíšme tretie mocniny čísel od 1 do 10.

```
<?php
function mocnina3($cislo) {
  return $cislo * $cislo * $cislo;
}

for ($i = 1; $i <= 10; $i++) {
  echo "$i - " . mocnina3($i) . "<br>\n";
}
?>
```

04/sucet.php

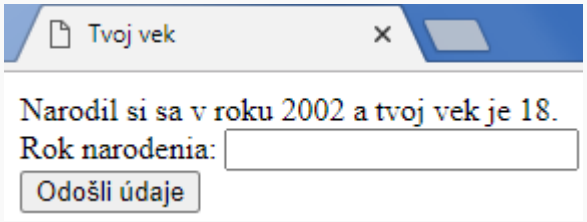
PRÍKLAD 4.15

Napíšme funkciu `sucet_cisel($pole)`, ktorá vypočíta súčet čísel v číselnom poli $pole. Potom túto funkciu otestujme na poliach `[2, 1, 4, 3, 6, 5, 8, 7, 10, 9]` a `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]`.

```
<?php
function sucet_cisel($pole) {
  $suc = 0;
  foreach($pole as $hodnota) {
    $suc += $hodnota;
  }
  return $suc;
}

$a = [2, 1, 4, 3, 6, 5, 8, 7, 10, 9];
$b = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29];

echo "súčet čísel v poli je " . sucet_cisel($a) . "<br>\n";
echo "súčet čísel v poli je " . sucet_cisel($b) . "<br>\n";
?>
```

ÚLOHA 4.16

Napíšte funkciu `vek()`, ktorá ako parameter dostane rok narodenia a jej výsledkom bude aktuálny vek (vypočíta sa pomocou aktuálneho roku). Vytvorte formulár, v ktorom sa bude dať zadať rok narodenia a po odoslaní formulára sa vypíše príslušný vek. Napr.:

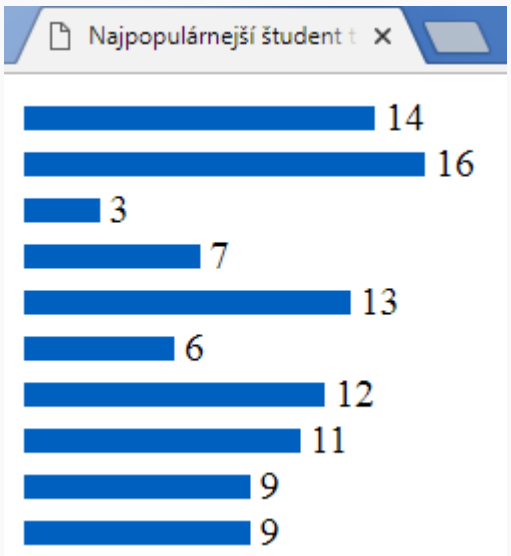


ÚLOHA 4.17

Napíšte funkciu `vytvor($n, $m)`, ktorá bude simulovať hlasovanie za najpopulárnejšieho študenta triedy. Parameter `$n` znamená počet študentov v triede a parameter `$m` znamená počet ľudí, ktorí budú hlasovať. Funkcia najskôr vytvorí pole dĺžky `$n` s hodnotami 0 a potom `$m`-krát vygeneruje náhodné číslo študenta a tomu pripočíta hlas. Výsledkom funkcie bude takto vygenerované pole.

Ďalej vytvorte funkciu `zobraz_vysledky($pole)`, ktorá ako parameter dostane pole vytvorené predchádzajúcou funkciou `vytvor()`. Táto funkcia Vytvorí riadkový graf pre jednotlivé prvky poľa. V každom riadku bude jeden obrázok (element `<img>`), ktorého šírka bude podľa príslušnej hodnoty z poľa (odporúčame vynásobiť 10, aby boli rozdiely viditeľnejšie).

Výsledná stránka pre volanie `vytvor(10,100)` a `zobraz_vysledky(pole)` by mohla vyzerať nasledovne:



4.4 Viaceré PHP vsuvky

Doteraz sme naše stránky riešili tak, že sme všetko, čo sme chceli vygenerovať pomocou PHP, dávali do jednej PHP vsuvky. V rámci stránky však môžeme mať viac PHP vsuviek kombinovaných s HTML kódom. Tak nemusíme všetky HTML elemtenty vypisovať pomocou príkazu `echo`. Ak niečo definujeme v jednej PHP vsuvke, môžeme to v niektorej nasledujúcej PHP vsuvke použiť. Napr.:

04/viacere_vsuvky.php

```php
<?php
$nadpis = 'Moja webstránka';
$autor = 'Janko Hraško';
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title><?php echo $nadpis; ?></title>
</head>

<body>
<h1><?php echo $nadpis; ?></h1>
<footer>vytvoril: <?php echo $autor; ?></footer>
</body>
</html>
```

V úlohe 3.11 sa mali vo výberovej ponuke vypísať rôzne druhy pizze. Keby sme však mali položiek veľa, tak by sme to určite neriešili písaním všetkých položiek samostatne, ale využili by sme PHP skript na jej vygenerovanie. Potrebujeme však mať údaje uložené v nejakej štruktúre. Vyriešme podobný príklad.

04/vyber_tovaru.php

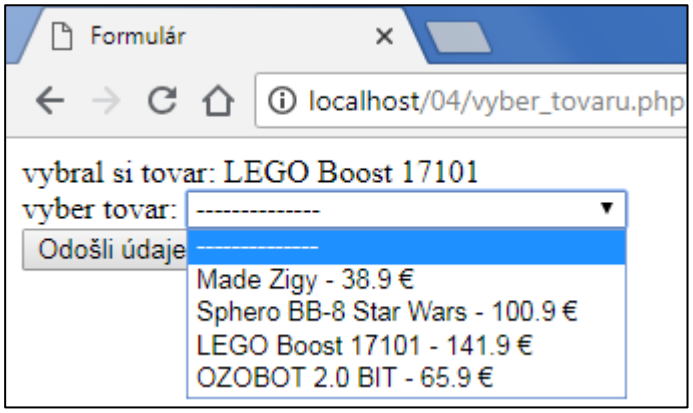
PRÍKLAD 4.18

Vytvorme výberovú ponuku s robotickými hračkami. Údaje o hračkách máme uložené v poli `$tovar = ['Made Zigy' => 38.90, 'Sphero BB-8 Star Wars' => 100.90, 'LEGO Boost 17101' => 141.90, 'OZOBOT 2.0 BIT' => 65.90];`, ktoré si môžeme zadefinovať na začiatku stránky. Vo výberovej ponuke sa budú zobrazovať názvy aj s cenou, ale po odoslaní formulára sa vypíše iba názov.

```php
<?php
$tovar = ['Made Zigy' => 38.90, 'Sphero BB-8 Star Wars' => 100.90
, 'LEGO Boost 17101' => 141.90, 'OZOBOT 2.0 BIT' => 65.90];
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Formulár</title>
</head>
<body>

<?php
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['vyber']) && ($_POST['vyber'] != '')) {
    echo 'vybral si tovar: ' . $_POST['vyber'];
  } else {
    echo 'Nevybral si žiaden tovar!';
  }
}
?>

<form method="post">
  <label for="vyber">vyber tovar:</label>
  <select name="vyber" id="vyber">
    <option value=''>--------------</option>
<?php
foreach($tovar as $kluc => $hodnota) {
  echo "<option value='$kluc'>$kluc - $hodnota €</option>";
}
?>
  </select><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>

</body>
</html>
```

PHP vsuvky

Výsledný formulár môže vyzerať nasledovne:



PRÍKLAD 4.19

Aby sme mohli vyberať viac tovarov, upravme náš predchádzajúci príklad tak, že namiesto výberovej ponuky použijeme začiarkávacie políčka.

Najskôr upravme len formulár:

```
<form method="post">
<?php
foreach($tovar as $kluc => $hodnota) {
  echo "<label for='$kluc'>$kluc - $hodnota €</label> <input
type='checkbox' name='$kluc' value='$kluc' id='$kluc'><br>";
}
?>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

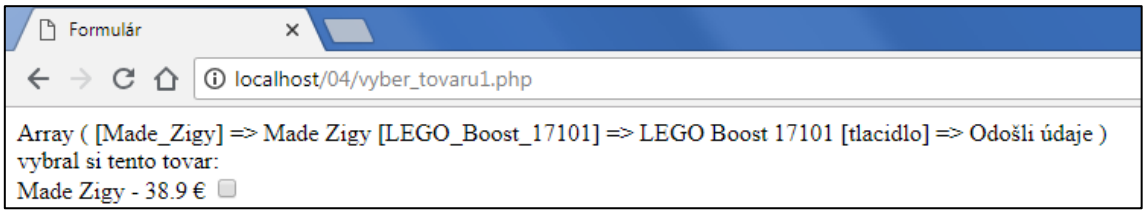
A teraz overovanie. Keďže potrebujeme overiť všetky začiarkávacie políčka a ich mená (`name`) sú kľúče z nášho poľa, možeme na to využiť cyklus.

```
if (isset($_POST['tlacidlo'])) {
  echo "vybral si tento tovar: ";
  foreach($tovar as $kluc => $hodnota) {
    if (isset($_POST[$kluc])) {
      echo $_POST[$kluc] . ", ";
    }
  }
  echo "<br>\n";
}
```

Aj napriek tomu, že začiarkneme nejaké produkty, nevypíšu sa nám. Dajme si vypísať `$_POST`.

```
if (isset($_POST['tlacidlo'])) {
  print_r($_POST);
  echo "<br>";
  echo "vybral si tento tovar: ";
  ...
}
```

A zobrazí sa:



Môžeme si všimnúť, že kľúče v tomto poli sa nezhodujú s kľúčmi v poli `$tovar` - medzery sú nahradené podčiarkovníkmi. To znamená, že naše kľúče v poli `$_POST` naozaj nemohol nájsť. Problémom je atribút `name` (`id`) elementu `input` (ale aj iných elementov), ktoré nesmú obsahovať medzeru.

To znamená, že naša štruktúra údajov uložená v poli `$tovar` nie je vyhovujúca. Aby sme mohli aj naďalej generovať formulár a aj jeho overovanie pomocou cyklu, potrebujeme vymyslieť inú štuktúru údajov. Ukážeme rôzne riešenia.

Viac jednoduchých polí

PRÍKLAD 4.20

Vyriešime príklad 4.19 pomocou dvoch jednoduchých polí.

Nahraďme pole `$tovar = ['Made Zigy' => 38.90, 'Sphero BB-8 Star Wars' => 100.90, 'LEGO Boost 17101' => 141.90, 'OZOBOT 2.0 BIT' => 65.90];` dvoma poľami `$tovar_nazvy = ['Made Zigy', 'Sphero BB-8 Star Wars', 'LEGO Boost 17101', 'OZOBOT 2.0 BIT'];` a `$tovar_ceny = [38.90, 100.90, 141.90, 65.90];`. Obe polia sú rovnako dlhé a cena zodpovedá názvu na rovnakej pozícii.

	0	1	2	3
$tovar_nazvy	'Made Zigy'	'Sphero BB-8 Star Wars'	'LEGO Boost 17101'	'OZOBOT 2.0 BIT'
$tovar_ceny	38.90	100.90	141.90	65.90

Teraz upravme formulár:

```
<form method="post">
<?php
foreach($tovar_nazvy as $kluc => $hodnota) {
  echo "<label for='t_$kluc'>$hodnota - $tovar_ceny[$kluc] €</label>
<input type='checkbox' name='t_$kluc' value='$hodnota'
id='t_$kluc'><br>\n";
}
?>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

Stačí nám prechádzať cez jedno pole (lebo indexy v oboch poliach sú rovnaké). Atribút **`for`** pre element **`label`** a atribúty **`name`** a **`value`** pre element **`input`** majú rovnakú hodnotu - **`t_$kluc`**. Prečo sme nemohli dať len **`$kluc`**? Lebo **`$kluc`** nadobúda číselné hodnoty a vieme, že **`name`** a **`id`** nesmú začínať číslom. Tak sme preto pridali pred **`$kluc`** prefix **`t_`**. Text v **`label`** sme skombinovali z oboch polí ako hodnoty z poľa, ktoré prechádzame (`$tovar_nazvy`) a hodnoty z poľa `$tovar_ceny` na rovnakom indexe ako je kľúč v prechádzanom poli.

04/vyber_tovaru20a.php

Dokončime overovanie. Najskôr jednoduchšia verzia:

```
if (isset($_POST['tlacidlo'])) {
  echo "vybral si tento tovar: ";
  foreach($tovar_nazvy as $kluc => $hodnota) {
    if (isset($_POST["t_$kluc"])) {
      echo $_POST["t_$kluc"] . ", ";
    }
  }
  echo "<br>\n";
}
```

Toto riešenie už funguje, ale môže byť neefektívne (keď je pole tovarov väčšie ako pole $_POST). Prechádzame cez celé pole `$tovar_nazvy`, aj keď v poli `$_POST` nebude žiaden tovar.

Môžeme prechádzať cez pole `$_POST`. Z tohto poľa nás budú zaujímať len tie prvky, ktorých kľúče začínajú `t_`. Na to využijeme reťazcovú funkciu `strpos()`.

04/vyber_tovaru20b.php

indexy/pozície v reťazcoch začínajú od 0

Funkcia `strpos(reťazec1, reťazec2)` vráti pozíciu prvého výskytu `reťazec2` v `reťazec1` alebo `FALSE`, ak sa `reťazec2` v `reťazec1` nenašiel. Záleží na veľkých a malých písmenách. Napr.:

príkaz	**výsledok**
`strpos('drevo','rev')`	1
`strpos('drevo','lev')`	FALSE
`strpos('drevo','d')`	0
`strpos('drevo','D')`	FALSE

Potrebujeme teda zistiť: `if (strpos($kluc, 't_') == 0)`

Vyskúšajme túto podmienku na inom príklade:

```
if (strpos('Ahoj', 'A') == 0) {
  echo "Nachádza<br>\n";
} else {
  echo "Nenachádza<br>\n";
}
if (strpos('Ahoj', 'a') == 0) {
  echo "Nachádza<br>\n";
} else {
  echo "Nenachádza<br>\n";
}
```

Predpokladali by sme, že v prvom prípade to vypíše *Nachádza* a v druhom *Nenachádza*, ale nám to v oboch prípadoch vypíše *Nachádza*. Prečo sa druhý prípad zle vyhodnotil?

Funkcia `strpos('Ahoj', 'a')` vráti FALSE a pri porovnaní pomocou `==` ho PHP pretypuje na `0` (písali sme o tom aj v 2. kapitole). Vyriešime to pomocou operátora `===`, ktorý kontroluje rovnosť hodnôt aj typu, teda či sú hodnoty identické a hodnoty `0` a `FALSE` nie sú identické.

ÚLOHA 4.21

Otestujte predchádzajúci kód s opraveným operátorom.

Aby sme mohli zisťovať údaje z poľa `$tovar_nazvy`, potrebujeme z kľúča (ktorý získame z poľa `$_POST`) odstrániť `t_`. Na to využijeme funkciu `substr()`.

Funkcia `substr(reťazec, štart, dĺžka)` vráti časť reťazca. `Reťazec` a `štart` sú povinné parametre. Parameter `štart` určuje, na akej pozícii začína výsledný reťazec. `0` znamená prvý znak v reťazci. Parameter `dĺžka` je voliteľný a znamená koľko maximálne znakov bude mať výsledný reťazec. Ak ho neuvedieme, tak výsledkom budú všetky znaky od `štart` po koniec reťazca. Napr.: `$vstup = "PHP v príkladoch"`.

príkaz	vypíše sa
`substr($vstup, 4)`	v príkladoch
`substr($vstup, 2, 5)`	P v p

Overovanie, v ktorom prechádzame cez pole `$_POST`, môže vyzerať nasledovne:

```
if (isset($_POST['tlacidlo'])) {
  echo "vybral si tento tovar: ";
  foreach($_POST as $kluc => $hodnota) {
    if (strpos($kluc, 't_') === 0) {
      $kluc = substr($kluc, 2);
      echo "{$tovar_nazvy[$kluc]}, ";
    }
  }
  echo "<br>\n";
}
```

Viac asociativnych polí

Aby sme sa vyhli upravovaniu kľúčov, môžeme využiť viac asociatívnych polí. Len musíme dobre definovať kľúče, čo znamená, že nesmú začínať číslom.

PRÍKLAD 4.22

Vyriešme príklad 4.19 pomocou dvoch asociatívnych polí.

Nahraďme pole `$tovar = ['Made Zigy' => 38.90, 'Sphero BB-8 Star Wars' => 100.90, 'LEGO Boost 17101' => 141.90, 'OZOBOT 2.0 BIT' => 65.90];` dvoma poliami `$tovar_nazvy = ['MZ' => 'Made Zigy', 'SSW' => 'Sphero BB-8 Star Wars', 'LB' => 'LEGO Boost 17101', 'OB' => 'OZOBOT 2.0 BIT'];` a `$tovar_ceny = ['MZ' => 38.90, 'SSW' => 100.90, 'LB' => 141.90, 'OB' => 65.90];`.

	'MZ'	'SSW'	'LB'	'OB'
$tovar_nazvy	'Made Zigy'	'Sphero BB-8 Star Wars'	'LEGO Boost 17101'	'OZOBOT 2.0 BIT'
$tovar_ceny	38.90	100.90	141.90	65.90

Upravme formulár:

```
<form method="post">
<?php
foreach($tovar_nazvy as $kluc => $hodnota) {
  echo "<label for='$kluc'>$hodnota - $tovar_ceny[$kluc] €</label>
<input type='checkbox' name='$kluc' value='$hodnota'
id='$kluc'><br>\n";
}
?>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

V tomto prípade máme kľúče vhodné, preto ich nepotrebujeme nijako upravovať.

04/vyber_tovaru22a.php

Dorobme overovanie, najskôr cez prechádzanie poľa `$tovar_nazvy`:

```
if (isset($_POST['tlacidlo'])) {
  echo "vybral si tento tovar: ";
  foreach($tovar_nazvy as $kluc => $hodnota) {
    if (isset($_POST[$kluc])) {
      echo $_POST[$kluc] . ", ";
    }
  }
  echo "<br>\n";
}
```

04/vyber_tovaru22b.php

A teraz cez prechádzanie poľa `$_POST`:

```
if (isset($_POST['tlacidlo'])) {
  echo "vybral si tento tovar: ";
  foreach($_POST as $kluc => $hodnota) {
    if (isset($tovar_nazvy[$kluc])) {
      echo $tovar_nazvy[$kluc] . ", ";
    }
  }
  echo "<br>\n";
}
```

ÚLOHA 4.23

Vytvorte stránku ako na obrázku:

Na začiatku stránky v PHP vsuvke zadefinujte premenné `$nadpis`, `$autor` a `$tovar` (alebo `$tovar_nazvy` a `$tovar_ceny`) a funkciu `vypis_poloziek($od, $do)`. V ďalších PHP vsuvkách vypíšte správne titulok stránky, nadpis stránky a pätu stránky. V tele stránky budú v zozname s odrážkami vypísané jednotlivé položky a pri každej bude výberová ponuka (od 0 do 10). Mená jednotlivých výberových ponúk sú kódy tovarov. Po odoslaní formulára sa vypíše rekapitulácia objednávky ako na obrázku:


Ponuka tovaru
Rekapitulácia objednávky:
Made Zigy: 2 ks
Sphero BB-8 Star Wars: 3 ks
OZOBOT 2.0 BIT: 1 ks
Ponuka tovaru
Made Zigy (38.9 €) 0

4.5 Metodické pokyny pre učiteľa

CIEĽ

Doplniť základné informácie o jazyku PHP - o prácu s náhodnými číslami, dátumom a časom, o vytváraní vlastných funkcií a o využívaní viacerých PHP vsuviek.

VÝKLAD

Táto časť je zameraná na:

- náhodné čísla,
- dátum a čas,
- vytváranie vlastných funkcií,
- využívanie viacerých PHP vsuviek.

Popri tom treba študentom vysvetliť niektoré reťazcové funkcie a operátor rovnosti hodnôt aj typov.

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

Náhodné čísla

`getrandmax()` je funkcia, ktorá vracia najväčšiu (celočíselnú) možnú hodnotu, ktorá môže byť vygenerovaná funkciou `rand()`

Dátum a čas

`date()` - funkcia má aj druhý nepovinný parameter timestamp, ktorý reprezentuje zadaný dátum a čas v špeciálnom tvare (vieme ho získať pomocou funkcie `mktime(hodiny, minúty, sekundy, mesiac, deň, rok)`)

Vlastné funkcie

Jazyk PHP nepozná pojem procedúra a teda všetky vlastné príkazy sú funkcie. V prípade, že chceme, aby funkcia skutočne vrátila nejakú hodnotu, použijeme príkaz `return`.Vlastná funkcia nesmie mať rovnaký názov ako už existujúce PHP funkcie a ani iné vlastné funkcie, lebo PHP nepodporuje prekrývanie funkcií.

K úlohe 4.13: Úloha by sa dala rozšíriť tak, že by nebol graf jednofarebný, ale použilo by sa viacero farieb (farebných obrázkov) a precvičili by sa náhodné čísla. Farby (názvy súborov) by mohli byť uložené v poli, napr. `$farby = ['modra.gif', 'cervena.gif', 'zlta.gif, 'zelena.gif', 'cierna.gif']`. Podľa dĺžky poľa by sa vygenerovalo číslo od 1 do dĺžky poľa a príslušná farba by sa použila v grafe.

Ešte rozšírený variant úlohy by mohol byť, že farby by boli ako položky výberovej ponuky a po odoslaní formulára by sa graf vykreslil príslušnou farbou.

Parametre – inicializačná hodnota

Toto je rozširujúca časť. Je na zvážení učiteľa, či sa jej bude so študentmi venovať.

PRÍKLAD

Upravme funkciu `vypis_poloziek()` z príkladu 4.10 tak, aby vypísala všetky čísla od čísla `$od` po číslo `$do` s krokom `$krok`.

02/vypis05a.php, 02/vypis05b.php

Funkcia by mohla vyzerať nasledovne:

```
<?php
function vypis_poloziek($od, $do, $krok) {
  for ($i = $od; $i <= $do; $i += $krok) {
    echo "<option value='$i'>$i</option>\n";
  }
}
```

A volať by sme ju mohli napríklad takto:

```
vypis_poloziek(1,12,1);
vypis_poloziek(2,20,2);
vypis_poloziek(4,40,4)
```

Ak by sme väčšinou potrebovali vypisovať čísla s krokom 1 a len vo výnimočných prípadoch chceme výpis s iným krokom, hodilo by sa nám, keby sme funkciu `vypis_poloziek()` volali len s dvomi parametrami a vo výnimočných prípadoch by sme funkciu zavolali so všetkými tromi. Presne takúto funkčnosť nám umožňuje tzv. **inicializačná hodnota parametra funkcie** – parametru funkcie priradíme hodnotu. V prípade, že taká hodnota je na konci zoznamu parametrov, nemusíme parameter vôbec uvádzať, a teda počet parametrov bude menší.

Uveďme si to však na našom príklade funkcie `vypis_poloziek()`, ktorú by sme mohli zmeniť nasledovne:

```
<?php
function vypis_poloziek($od, $do, $krok = 1 ) {
  for ($i = $od; $i <= $do; $i += $krok) {
    echo "<option value='$i'>$i</option>\n";
  }
}
```

Potom môžeme takto definovanú funkciu zavolať aj uvedenými spôsobmi:

```
vypis_poloziek(2020,2030);
vypis_poloziek(1,28);
vypis_poloziek(2,10,2)
```

Našu funkciu by sme mohli upraviť aj takto:

```
function vypis_poloziek($od = 1, $do = 10, $krok = 1) {}
```

Všetkým trom parametrom sme dali inicializačné hodnoty.

ÚLOHA

Zamyslite sa a odpovedzte, aké čísla sa zobrazia vo výberovej ponuke pri rôznych volaniach funkcie `vypis_poloziek($od=1, $do=10, $krok=1)`:

- `vypis_poloziek(0,30,3);`
- `vypis_poloziek(1,50,5);`
- `vypis_poloziek(1,31);`
- `vypis_poloziek(11,20);`
- `vypis_poloziek(5);`
- `vypis_poloziek(10);`
- `vypis_poloziek();`

Viaceré PHP vsuvky

K úlohe 4.19: V učebnici sú uvedené 2 spôsoby pomocou rôznych polí. Učiteľ môže zvážiť, či ukáže obe, alebo len jedno z nich. Táto úloha by sa dala vyriešiť aj pomocou jednej zložitejšej štruktúry.

Jedna zložitejšia štruktúra

Posledná možnosť, ktorú si ukážeme, využíva iba jednu zložitejšiu štruktúru. Každý tovar bude jedno asociatívne pole s kľúčmi `'kod'`, `'nazov'` a `'cena'`. Potom tieto asociatívne polia spojíme do jedného veľkého poľa. Tento spôsob sa najviac približuje spracovaniu údajov z databáz (tomu sa však už nebudeme venovať).

PRÍKLAD

04/vyber_tovaru24.php

Vyriešime príklad 4.19 pomocou údajov z jednej štruktúry.

Nahraďme pôvodné pole `$tovar` nasledujúcou štruktúrou:

```
$tovar = [['kod' => 'MZ', 'nazov' => 'Made Zigy', 'cena' => 38.90],
          ['kod' => 'SSW', 'nazov' => 'Sphero BB-8 Star Wars',
           'cena' => 100.90],
          ['kod' => 'LB', 'nazov' => 'LEGO Boost 17101',
           'cena' => 141.90],
          ['kod' => 'OB', 'nazov' => 'OZOBOT 2.0 BIT',
           'cena' => 65.90]];
```

Zodpovedá jej nasledujúci obrázok:

	'kod'	'nazov'	'cena'
0	'MZ'	'Made Zigy'	38.90
1	'SSW'	'Sphero BB-8 Star Wars'	100.90
2	'LB'	'LEGO Boost 17101'	141.90
3	'OB'	'OZOBOT 2.0 BIT'	65.90

Každý prvok nášho poľa `$tovar` je vlastne asociatívne pole.

Skúsme vypísať niektoré prvky poľa `$tovar`:

`print_r($tovar[0]);`	Array ([kod] => MZ [nazov] => Made Zigy [cena] => 38.9)
`print_r($tovar[2]);`	Array ([kod] => LB [nazov] => LEGO Boost 17101 [cena] => 141.9)

Ak chceme vypísať názov tovaru s indexom 1, tak zadáme echo `$tovar[1]['nazov']` a pod.

Upravme najskôr formulár:

```
<form method="post">
<?php
foreach($tovar as $hodnota) {
  echo "<label for='{$hodnota['kod']}'>{$hodnota['nazov']} -
{$hodnota['cena']} €</label> <input type='checkbox'
name='{$hodnota['kod']}' value='{$hodnota['nazov']}'
id='{$hodnota['kod']}'><br>\n";
}
?>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

Uvedomme si, že v `$hodnota` sú postupne jednotlivé asociatívne polia, ako prvé - `['kod' => 'MZ', 'nazov' => 'Made Zigy', 'cena' => 38.90]`.

A nakoniec dokončíme overovanie. Spravíme to len cez prechádzanie poľom `$tovar`.

```
if (isset($_POST['tlacidlo'])) {
  echo "vybral si tento tovar: ";
  foreach($tovar as $hodnota) {
    if (isset($_POST[$hodnota['kod']])) {
      echo $_POST[$hodnota['kod']] . ", ";
    }
  }
  echo "<br>\n";
}
```

Užitočné reťazcové funkcie

názov funkcie	**Popis**
`strlen(ret)`	Vráti dĺžku reťazca.
`substr(ret, zač [, počet])`	Vráti podreťazec dĺžky počet od znaku na pozícii zač.
`strtolower(ret)`	Vráti reťazec prevedený na malé písmená.

`strtoupper(ret)`	Vráti reťazec prevedený na veľké písmená.
`strrev(ret)`	Vráti prevrátený reťazec.
`trim(ret), ltrim(ret), rtrim(ret)`	Odstráni tzv. whitespace znaky z príslušného konca.
`chr(ascii-kod)`	Vráti znak zodpovedajúci ascii-kódu.
`ord(ret)`	Vráti ASCII kód prvého znaku reťazca.
`nl2br()`	Prevedie konce riadkov (\n) na .
`strpos(kde, čo [, zač])`	Vráti prvú pozíciu čo v reťazci kde zľava alebo FALSE.
`strrpos(kde, čo [, zač])`	Vráti prvú pozíciu čo (len 1 znak) v reťazci kde zprava, alebo FALSE.
`strrchr(kde, čo)`	Vráti podreťazec od pozície čo v reťazci kde zprava do konca reťazca, alebo FALSE. Vyhľadáva len znaky.
`strstr(kde, čo)`	Vráti podreťazec od pozície čo v reťazci kde zľava do konca reťazca, alebo FALSE.
`str_replace(čo, čím, kde)`	Nahradí všetky výskyty čo reťazcom čím v kde.

Funkcia `substr(reťazec, štart, dĺžka)` môže mať aj záporné parametre. Záporná hodnota `štart` definuje pozíciu od konca reťazca. Záporný parameter `dĺžka` určuje, koľko znakov od konca nebude vo výslednom reťazci. Napr.: `$vstup = "PHP v príkladoch"`

Príkaz	**vypíše sa**
`substr($vstup, -3)`	Och
`substr($vstup, 4, -3)`	v príklad
`substr($vstup, -7, 4)`	Klad

5 PHP - SPRACOVANIE FORMULÁROV (2)

V kapitole č. 3 sme si ukázali základné spracovanie formulárov. To má však viacero nedostatkov, ktoré vyriešime v tejto kapitole.

5.1 Zobrazenie odoslaných údajov znovu vo formulári

DISKUTUJTE

Máme formulár, v ktorom sú niektoré položky povinné. Čo sa má stať, keď nevyplníme všetky povinné položky a odošleme formulár?

Ako by sa mal znovu zobraziť takto odoslaný formulár?

Ak po odoslaní formulára ho znovu zobrazíme a chceme v ňom zobraziť už odoslané údaje, musíme nastaviť tzv. default hodnoty pre jednotlivé HTML elementy. Toto nastavenie sa líši v závislosti od typu prvku formulára - rôzne prvky majú iný spôsob. Poďme si ich postupne ukázať a vysvetliť.

Jednoriadkové textové pole

Ak chceme v textovom poli zobraziť nejakú hodnotu pri načítaní formulára (prednastavená hodnota), musíme nastaviť atribút `value`.

```
<input type="text" name="meno" id="meno" value="hodnota">
```

rovnako by sme pracovali aj s prvkom pre heslo (typ password)

PRÍKLAD 5.1

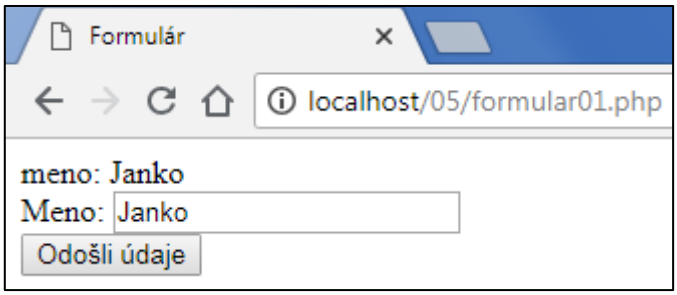
Doplňme formulár z príkladu 3.6 (`03/formular06.php`) tak, aby sa odoslané meno zobrazilo vo formulárovom poli, teda do atribútu `value` dáme hodnotu z poľa `$_POST`.

05/formular01.php

```
<label for="meno">Meno:</label> <input type="text" name="meno"
id="meno"
value="<?php if (isset($_POST['meno'])) {echo $_POST['meno'];} ?>">
```

Skôr, ako vypíšeme hodnotu z poľa `$_POST`, musíme overiť, či existuje.

Výsledný formulár po odoslaní môže vyzerať nasledovne:



ÚLOHA 5.2

Upravte úlohu 3.8 tak, aby sa po odoslaní formulára zobrazili odoslané hodnoty znovu vo formulárových poliach pre meno, priezvisko a mesto.

Viacriadkové textové pole TEXTAREA

Element `<textarea>` nemá žiadny atribút, ktorý treba nastaviť, ak v ňom chceme zobraziť nejakú hodnotu. Takáto default hodnota sa vkladá medzi otváraciu a zatváraciu značku – medzi nimi by nemali byť žiadne medzery ani nové riadky.

```
<textarea name="adresa">hodnota</textarea>
```

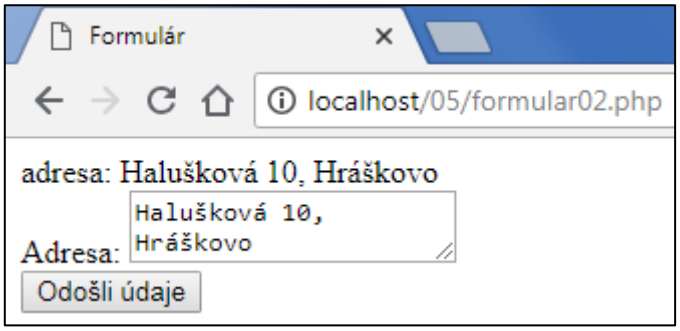
PRÍKLAD 5.3

05/formular02.php

Doplňme formulár z príkladu 3.7 (`03/formular07.php`) tak, aby sa odoslaná adresa zobrazila vo formulárovom poli, teda medzi otváraciu a koncovú značku dáme hodnotu z poľa `$_POST`.

```
<textarea name="adresa" id="adresa"><?php if
(isset($_POST['adresa'])) {echo $_POST['adresa'];} ?></textarea>
```

Výsledný formulár po odoslaní môže vyzerať nasledovne:



ÚLOHA 5.4

Upravte úlohu 5.2 tak, že prvok mesto bude element `<textarea>`. Zachovajte zobrazovanie všetkých odoslaných hodnôt vo formulári.

Výberová ponuka SELECT

Ak chceme označiť (prednastaviť) niektorú položku výberovej ponuky, nenastavujeme to elementu `<select>`, lebo ten nemá žiaden vhodný atribút. Musíme konkrétnej položke (element `<option>`), zadať atribút `selected`.

```
<select name="pocet" id="pocet">
  <option value="">0</option>
  <option value="1">1</option>
  <option value="2" selected>2</option>
  <option value="3">3</option>
  <option value="4">4</option>
</select>
```

PRÍKLAD 5.5

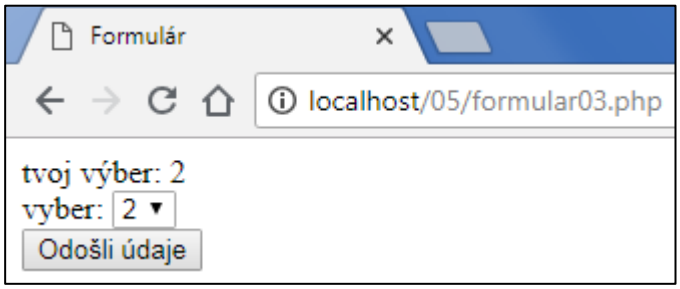
05/formular03.php

V elemente `<select>` máme možnosť výberu štyroch čísel. Doplňme do formulára kód tak, aby sa zobrazila zvolená hodnota aj po odoslaní formulára. (Mierne upravíme formulár z príkladu 3.10.).

```
<select name="pocet" id="pocet">
  <option value="">0</option>
  <option value="1"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 1)) {
     echo 'selected';} ?>
     >1</option>
  <option value="2"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 2)) {
     echo 'selected';} ?>
     >2</option>
  <option value="3"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 3)) {
     echo 'selected';} ?>
     >3</option>
  <option value="4"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 4)) {
     echo 'selected';} ?>
     >4</option>
</select>
```

Do každej položky `<option>` musíme vložiť PHP kód, ktorý skontroluje, či existuje premenná `$_POST["pocet"]` a či sa rovná hodnote príslušnej položky. Ak áno, tak sa pri nej vypíše `selected`.

Výsledný formulár po odoslaní môže vyzerať nasledovne:



Vidíme, že zobrazenie zvolenej položky pre element `<select>` je zdĺhavé a náchylné na chyby. Je výhodné, ak môžeme jednotlivé položky pre výberovú ponuku vygenerovať pomocou funkcie. Ukážeme si to na funkcii `vypis_poloziek()`, ktorá nám generuje číselné položky.

05/formular04.php

PRÍKLAD 5.6

Upravme funkciu `vypis_poloziek()` z príkladu 4.10 tak, aby vedela označiť niektorú z položiek.

Docielime to tak, že funkcii pridáme tretí parameter, ktorý bude reprezentovať položku, ktorú chceme mať označenú.

Doplnená funkcia:

```
function vypis_poloziek($od, $do, $oznac) {
  for ($i = $od; $i <= $do; $i++) {
    echo "<option value='$i'";
    if ($i == $oznac) {
      echo ' selected';
    }
    echo ">$i</option>\n";
  }
}
```

A zavolanie funkcie:

```
<select name="pocet" id="pocet">
  <option value="">0</option>
<?php
if (isset($_POST['pocet'])) {
  vypis_poloziek(1, 4, $_POST['pocet']);
} else {
  vypis_poloziek(1, 4, 0);
}
?>
</select>
```

ÚLOHA 5.7

Upravte úlohu 3.11 tak, aby sa po odoslaní formulára zobrazila zvolená pizza vo výberovej ponuke.

Prepínač - RADIOBUTTON

Ak chceme označiť (prednastaviť) niektorý prepínač, musíme mu zadať atribút `checked`.

```
<input type="radio" name="pocet" value="1" id="pocet1"><br>
<input type="radio" name="pocet" value="2" id="pocet2"><br>
<input type="radio" name="pocet" value="3" id="pocet3" checked><br>
<input type="radio" name="pocet" value="4" id="pocet4"><br>
```

PRÍKLAD 5.8

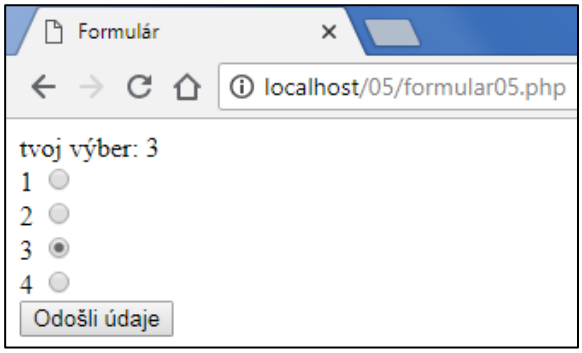
05/formular05.php

Máme štyri prepínače, pomocou ktorých môžeme vybrať jedno zo štyroch čísel. Doplňme do formulára kód tak, aby sa zobrazila zvolená hodnota aj po odoslaní formulára. (Mierne upravíme formulár z príkladu 3.12.).

```
<label for="pocet1">1</label>
<input type="radio" name="pocet" value="1" id="pocet1"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 1)) {
     echo ' checked';} ?>><br>

<label for="pocet2">2</label>
<input type="radio" name="pocet" value="2" id="pocet2"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 2)) {
     echo ' checked';} ?>><br>

<label for="pocet3">3</label>
<input type="radio" name="pocet" value="3" id="pocet3"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 3)) {
     echo ' checked';} ?>><br>

<label for="pocet4">4</label>
<input type="radio" name="pocet" value="4" id="pocet4"
<?php if (isset($_POST['pocet']) && ($_POST['pocet'] == 4)) {
     echo ' checked';} ?>><br>
```

Do každého prepínača musíme vložiť PHP kód, ktorý skontroluje, či existuje premenná `$_POST["pocet"]` a či sa rovná hodnote príslušného prepínača. Ak áno, tak sa mu pridá atribút `checked`.

Výsledný formulár po odoslaní môže vyzerať nasledovne:



ÚLOHA 5. 10

Upravte predchádzajúci príklad tak, aby prepínače generovala funkcia.

ÚLOHA 5.11

Upravte úlohu 3.13 tak, aby po odoslaní formulára bola označená zvolená pizza.

Začiarkávacie políčko - CHECKBOX

Pripomeňme si, že začiarkávacích políčok môže byť označených aj viac.

Ak chceme označiť (prednastaviť) niektoré začiarkávacie políčka, musíme im zadať atribút `checked`.

```
<input type="checkbox" name="vyber1" value="1" id="vyber1">
<input type="checkbox" name="vyber2" value="2" id="vyber2" checked>
<input type="checkbox" name="vyber3" value="3" id="vyber3" checked>
<input type="checkbox" name="vyber4" value="4" id="vyber4">
```

05/formular06.php

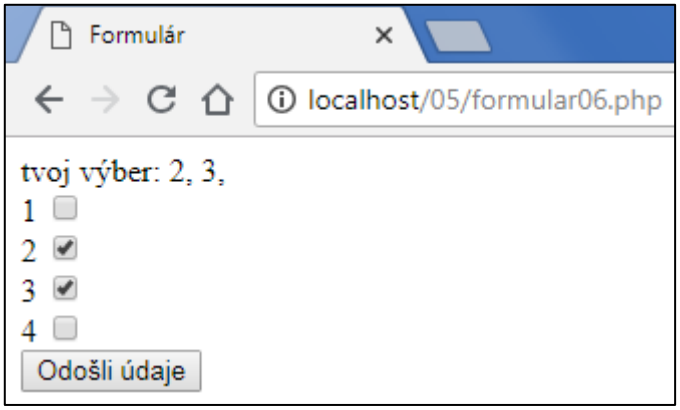
PRÍKLAD 5.12

Máme štyri začiarkávacie políčka, pomocou ktorých môžeme vybrať niektoré (aj viaceré) zo štyroch čísel. Doplňme do formulára kód tak, aby sa zobrazili zvolené hodnoty aj po odoslaní formulára. (Mierne upravíme formulár z príkladu 3.15.).

```
<label for="vyber1">1</label>
<input type="checkbox" name="vyber1" value="1" id="vyber1"
<?php if (isset($_POST['vyber1'])) {echo ' checked';} ?>
><br>

<label for="vyber2">2</label>
<input type="checkbox" name="vyber2" value="2" id="vyber2"
<?php if (isset($_POST['vyber2'])) {echo ' checked';} ?>
><br>

<label for="vyber3">3</label>
<input type="checkbox" name="vyber3" value="3" id="vyber3"
<?php if (isset($_POST['vyber3'])) {echo ' checked';} ?>
><br>

<label for="vyber4">4</label>
<input type="checkbox" name="vyber4" value="4" id="vyber4"
<?php if (isset($_POST['vyber4'])) {echo ' checked';} ?>
><br>
```

Do každého začiarkávacieho políčka musíme vložiť PHP kód, ktorý skontroluje, či existuje príslušná premenná `$_POST["vyberX"]`. Ak áno, tak sa mu pridá atribút `checked`.

Výsledný formulár po odoslaní môže vyzerať nasledovne:



ÚLOHA 5.13

Upravte predchádzajúci príklad tak, aby začiarkávacie políčka generovala funkcia.

ÚLOHA 5.14

Upravte úlohu 3.16 tak, aby po odoslaní formulára boli označené zvolené programovacie jazyky.

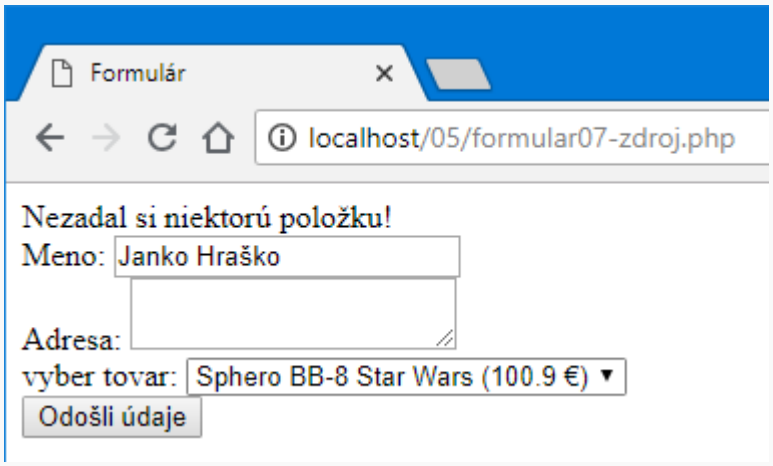
5.2 Skrývanie formulára

Doteraz sa nám formulár zobrazoval vždy, aj po odoslaní vyplneného formulára. Ale sú situácie, keď to nie je žiadúce. Napr. keď odošleme registráciu, nepotrebujeme znovu vidieť formulár (s vyplnenými hodnotami), skôr očakávame potvrdenie registrácie. Alebo pri vypĺňaní testu cez webový formulár tiež nečakáme, že sa nám znovu zobrazí formulár, ale skôr výsledný počet bodov, resp. správnych odpovedí.

PRÍKLAD 5.15

05/formular07-zdroj.php

Majme formulár s položkami *meno*, *adresa*, *tovar* (príklady 5.1, 5.3, 4.18), pričom použijeme dve polia s názvami a cenami tovarov (`$tovar_nazvy`, `$tovar_ceny` z príkladu 4.22). Ak budú po odoslaní formulára zadané všetky 3 hodnoty, zobrazí sa rekapitulácia objednávky, ale formulár sa už nezobrazí. Ak nebude niektorá z hodnôt zadaná, zobrazíme upozornenie, že neboli vyplnené všetky údaje a zobrazíme formulár, v ktorom už budú predvyplnené zadané údaje.



Využijeme kódy z uvedených príkladov, ktoré však spojíme. Upravený zdrojový kód je v súbore `05/formular07-zdroj.php`.

Formulár potrebujeme zobraziť len v dvoch prípadoch:

1. keď prvýkrát zobrazujeme stránku (nebolo odoslané tlačidlo),
2. keď bol odoslaný nekompletne vyplnený formulár.

riešenie je v 05/formular07.php

V zdrojovom kóde sme doplnili komentáre, kde by sme potrebovali zobraziť ten istý formulár. Mohli by sme celý formulár nakopírovať na obidve miesta a všetko by fungovalo. Ale to nie je veľmi efektívne riešenie. Kód by sme mali mať napísaný iba raz a len ho nejako využívať.

```
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && ($_POST['meno'] != '') &&
      isset($_POST['adresa']) && ($_POST['adresa'] != '') &&
      isset($_POST['vyber']) && ($_POST['vyber'] != '')) {
    // boli odoslané všetky údaje a zobrazíme rekapituláciu
    echo "meno: {$_POST['meno']}<br>\n";
    echo "adresa: {$_POST['adresa']}<br>\n";
    echo "tovar: {$tovar_nazvy[$_POST['vyber']]}<br>\n";
    // tu formulár nezobrazíme
  } else {
    echo 'Nezadal si niektorú položku!<br>';
    // tu treba zobraziť formulár
  }
} else {
  // tu treba zobraziť formulár
}
```

Riešením môže byť použitie premennej (napr. `$zobraz_form`), ktorá keď bude `TRUE`, tak kód s formulárom zobrazíme, keď bude `FALSE`, tak formulár nezobrazíme. Pred testom stlačenia odosielacieho tlačidla premennú `$zobraz_form` nastavíme na `TRUE`. A pri teste kompletne vyplneného formulára ju nastavíme na `FALSE`.

ÚLOHA 5.16

Upravte úlohu 4.23 tak, že keď používateľ zadá niektorý z tovarov (nenulový počet), tak sa už formulár znovu nezobrazí. Inak sa formulár bude zobrazovať.

formular07.php

```php
<?php
$tovar_nazvy = ['MZ' => 'Made Zigy', 'SSW' => 'Sphero BB-8 Star Wars', 'LB' => 'LEGO Boost 17101', 'OB' => 'OZOBOT 2.0 BIT'];
$tovar_ceny = ['MZ' => 38.90, 'SSW' => 100.90, 'LB' => 141.90, 'OB' => 65.90];

function vypis_tovarov($nazvy, $ceny, $oznac) {

$zobraz_form = true;
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && ($_POST['meno'] != '') &&
      isset($_POST['adresa']) && ($_POST['adresa'] != '') &&
      isset($_POST['vyber']) && ($_POST['vyber'] != '')) {
    // boli odoslané všetky údaje a zobrazíme rekapituláciu
    echo "meno: {$_POST['meno']}<br>\n";
    echo "adresa: {$_POST['adresa']}<br>\n";
    echo "tovar: {$tovar_nazvy[$_POST['vyber']]}<br>\n";
    // tu formulár nezobrazíme
    $zobraz_form = false;
  } else {
    echo 'Nezadal si niektorú položku!<br>';
    // tu treba zobraziť formulár
  }
} else {
  // tu treba zobraziť formulár
}

if ($zobraz_form) {
?>
<form method="post">
  <label for="meno">Meno:</label> <input type="text" name="meno" id="meno" value="<?php if (isset($_POST["meno"])) { echo $_POST["meno"]; } ?>"><br>
  <label for="adresa">Adresa:</label> <textarea name="adresa" id="adresa"><?php if (isset($_POST["adresa"])) { echo $_POST["adresa"]; } ?></textarea><br>
  <label for="vyber">vyber tovar:</label>
  <select name="vyber" id="vyber">
    <option value=''>--------------</option>
<?php
if (isset($_POST["vyber"])) {
  vypis_tovarov($tovar_nazvy, $tovar_ceny, $_POST["vyber"]);
} else {
  vypis_tovarov($tovar_nazvy, $tovar_ceny, '');
}
?>
  </select><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
<?php
}
?>
```

5.3 Kontrola vstupov od používateľa

Pri spracovaní formulárov je veľmi dôležité kontrolovať vstupy od používateľa (teda hodnoty, ktoré používateľ zadá do formulára, resp. môže zadať). Tejto problematike treba venovať zvýšenú pozornosť, lebo práve nedostatočnou kontrolou hodnôt vznikajú časté bezpečnostné problémy.

Ukážeme si dve rôzne kontroly:

1. či hodnoty spĺňajú nejaké pravidlá (správna dĺžka, číslo, korektná e-mailová adresa, ...) - kontrola textových polí
2. či hodnoty neobsahujú nežiadúce kódy (html značky, javascript, ...) - kontrola všetkých formulárových prvkov, aj tých, ktoré to na prvý pohľad nepotrebujú

Kontrola formátov vstupov

V textovom poli môžu používatelia zadávať rôzne údaje, väčšinou sú to textové informácie - meno, priezvisko, adresa, e-mailová adresa, ale môžu byť aj číselné, napr. dátum narodenia.

05/kontrola01.php

PRÍKLAD 5.17

Vytvorme funkciu `spravne_meno($m)`, ktorá vráti `TRUE`, ak zadané meno bude mať dĺžku minimálne 3 znaky a maximálne 30 znakov. Inak vráti `FALSE`.

Na zistenie dĺžky reťazca použijeme reťazcovú funkciu `strlen()`.

Funkcia `strlen(reťazec)` vráti dĺžku reťazca alebo 0, ak je reťazec prázdny.

```
function spravne_meno($m) {
  return (strlen($m) >= 3) && (strlen($m) <= 30);
}
```

Upravíme aj kontrolu odosielania formulára. Už nebudeme kontrolovať, či `$_POST['meno'] != ''`, ale či má `$_POST` správny formát.

```
$zobraz_form = true;
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && spravne_meno($_POST['meno'])) {
    echo 'meno: ' . $_POST['meno'];
    $zobraz_form = false;
  } else {
    echo 'Nezadal si meno alebo meno nemá správny formát!';
  }
}
```

05/kontrola02.php

PRÍKLAD 5.18

Vytvorme funkciu `spravne_psc($psc)`, ktorá vráti `TRUE`, ak zadané PSČ bude mať presne 6 znakov a na 3. indexe je medzera. Inak vráti `FALSE`.

```
function spravne_psc($psc) {
  return (strlen($psc) == 6) && (strpos($psc, ' ') == 3);
}
```

PRÍKLAD 5.15

05/kontrola03.php

Vytvorme funkciu `spravny_vek($vek)`, ktorá vráti `TRUE`, ak zadaný vek je číslo väčšie ako 14. Inak vráti `FALSE`.

Parameter `$vek` môže byť reťazec s číslom, ak je údaj získaný z textového poľa formulára. Preto by sme potrebovali zistiť, či je daný parameter číslo alebo reťazec s číslom (teda, či sa dá daný reťazec korektne previesť na číslo). Na to slúži funkcia `is_numeric()`.

Funkcia `is_numeric($premenna)` zisťuje, či je zadaná premenná číslo alebo reťazec s číslom, teda vráti `TRUE` alebo `FALSE`.

Ešte potrebujeme pretypovať (prekonvertovať) reťazec na číslo, aby sme otestovali aj podmienku, že číslo je väčšie ako 14.

Ľubovoľnú premennú môžeme pretypovať na nami definovaný typ. Nemusí sa to vždy podariť. Napríklad, ak chceme prekonvertovať reťazec 'abc' na číslo, tak výsledkom bude 0. Pretypovanie robíme pomocou nasledujúceho zápisu: `(nový_typ) $premenna`.

Napr.:

`(int) '123'`	123
`(int) 13.4`	13
`(real) 13`	13.0

Naša funkcia by mohla vyzerať nasledovne:

```
function spravny_vek($vek) {
  return is_numeric($vek) && ((int) $vek > 14);
}
```

DISKUSIA

Čo by sa mohlo ešte kontrolovať pri PSČ? Aké podmienky by ste vymysleli na kontrolu telefónneho čísla?

ÚLOHA 5.20

Vytvorte funkciu `spravny_email($mail)`, ktorá vráti `TRUE`, ak je zadaný e-mail korektný. Inak vráti `FALSE`. E-mail je korektný, keď má minimálne 6 znakov a obsahuje znak @ a niekde v reťazci za @ je bodka.

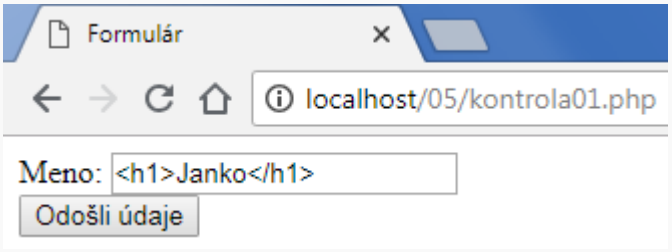
Rovnaké kontroly môžeme použiť aj pre `<textarea>`.

Ošetrenie vstupov

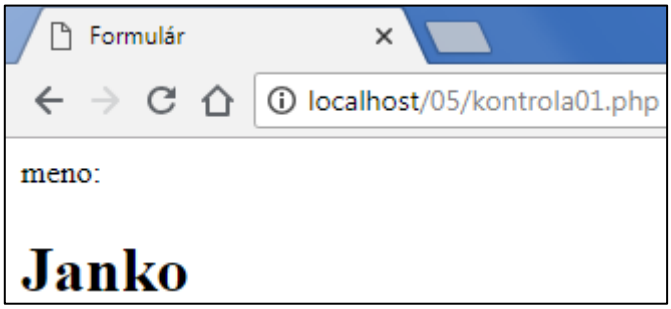
Ošetrovať používateľské vstupy môžeme viacerými spôsobmi – každý má svoje určenie.

PRÍKLAD 5.21

Zobrazme stránku z príkladu 5.17 a do textového poľa pre meno zadajme “<h1>Janko<h1>”. Čo sa zobrazí?



Zobrazí sa:



Janko sa nám zobrazil ako nadpis, čo asi nie je žiadúce, lebo to kazí vzhľad stránky. Ak by sme do textového poľa zadali len “<h1>Janko”, tak by nadpis ostal otvorený, a teda by sa takým štýlom mohli zobraziť aj ďalšie informácie na stránke, ktoré by boli vypísané po rekapitulácii.

Potrebujeme teda zamedziť zobrazeniu HTML elementov. Na to slúži funkcia `htmlspecialchars(reťazec)`. Táto funkcia zmení preddefinované znaky na HTML entity. Preddefinované znaky sú: & (&), " ("), ' ('), < (<), > (>). Napr.:

zadáme	bude v zdrojovom kóde
Janko	**<**h1**>**Janko**<**/h1**>**
<?php Janko ?>	**<**?php Janko ?**>**

To spôsobí, že zadané elementy stratia svoju funkciu a vypíšu sa len ako znaky. Táto funkcia sa využíva najmä vtedy, ak sa zadané hodnoty formulára vypisujú (hneď alebo neskôr) na stránke v rámci definovanej štruktúry.

PRÍKLAD 5.22

Upravme skript z príkladu 5.17 tak, aby sa neaplikovali prípadné HTML elementy zadané vo formulári.

05/kontrola04.php

```
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && spravne_meno($_POST['meno'])) {
    echo 'meno: ' . htmlspecialchars($_POST['meno']);
    $zobraz_form = false;
  } else {
    echo 'Nezadal si meno alebo meno nemá správny formát!';
  }
}
```

Nekontrolovanie (neodfiltrovanie) HTML elementov by nemalo spôsobiť vážne problémy na stránke (a už vôbec nie na serveri), ale môže narušiť vzhľad a niekedy aj funkčnosť stránky. Toto sa často stáva pri rôznych diskusných fórach, ktoré si autori vytvárajú sami.

Ak chceme úplne odfiltrovať HTML (aj XML a PHP) elementy, použijeme funkciu `strip_tags(reťazec)`. Táto funkcia z reťazca odstráni všetky HTML elementy. Napr.:

zadáme	v zdrojovom kóde aj na webovej stránke
Janko	Janko
<?php Janko ?>	

V druhom prípade sa nezobrazí nič, lebo sa odstráni celá PHP vsuvka.

PRÍKLAD 5.23

Upravme predchádzajúci príklad (5.22) tak, aby sa odstránili prípadné html elementy zadané vo formulári.

05/kontrola05.php

```
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && spravne_meno($_POST['meno'])) {
    echo 'meno: ' . strip_tags($_POST['meno']);
    $zobraz_form = false;
  } else {
    echo 'Nezadal si meno alebo meno nemá správny formát!';
  }
}
```

Ďalšia kontrola je už na inej úrovni. Používateľ totiž môže okrem HTML elementov zadať aj rôzne špeciálne znaky, ktoré síce nemajú v jazyku HTML žiadny význam, ale majú význam pri

príkazoch vykonávaných na serveri. Neodfiltrovanie týchto špeciálnych znakov spolu so slabším zabezpečením servera, môže mať až katastrofálne následky. Niektoré špeciálne znaky môžeme eliminovať pomocou funkcie `addslashes(reťazec)`. Táto funkcia pridá (ako vyplýva z názvu) pred preddefinované znaky spätné lomítko \, čím znefunkční ich pôvodný význam a stanú sa z nich len bežné znaky. Preddefinované znaky sú: ‘ (apostrof), “ (úvodzovka), \ (spätné lomítko) a NULL.

05/kontrola06.php

PRÍKLAD 5.24

Upravme príklad 5.22 tak, aby sa znefunkčnili špeciálne znaky.

```
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && spravne_meno($_POST['meno'])) {
    echo 'meno: ' . addslashes($_POST['meno']);
    $zobraz_form = false;
  } else {
    echo 'Nezadal si meno alebo meno nemá správny formát!';
  }
}
```

Silnejšou funkciou je `quotemeta(reťazec)`, ktorá pridáva spätné lomítka pred nasledujúce znaky: `. \ + * ? [ ^ ] ( $ )`.

Diskuisia: Pri kontrole akých údajov by nám mohli tieto funkcie narobiť problémy?

Ako skĺbiť kontrolu hodnôt vstupov s ošetrením vstupov, teda, ako robiť kontrolu hodnôt na ošetrených vstupoch?

PRÍKLAD 5.25

Vytvorme funkciu `osetri()`, ktorá ako parameter dostane reťazec a aj výsledkom bude reťazec, ktorý nebude obsahovať HTML elementy a pred špeciálnymi znakmi budú spätné lomítka.

```
function osetri($ret) {
  return quotemeta(addslashes(trim(strip_tags($ret))));
}
```

Na reťazec sme aplikovali predchádzajúce funkcie a okrem toho sme pridali aj funkciu `trim()`.

Funkcia `trim(reťazec)` zo začiatku a konca reťazca odstráni medzery, tabulátory, konce riadkov a ďalšie “biele znaky” (znaky, ktoré sa nezobrazujú).

PRÍKLAD 5.26

05/kontrola07.php

Dorobme kontrolu formulára a upravme aj formulár.

```
$zobraz_form = true;
if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) {
    $meno = osetri($_POST['meno']);
  } else {
    $meno = '';
  }
  if (spravne_meno($meno)) {
    echo "meno: $meno<br>\n";
    $zobraz_form = false;
  } else {
    echo "Nezadal si meno alebo meno nemá správny formát!<br>\n";
  }
}
```

Najskôr si do premennej `$meno` dáme ošetrený vstup (`osetri($_POST['meno'])`) alebo prázdny reťazec a potom overíme, či spĺňa požadované podmienky (`spravne_meno($meno)`).

Vo formulári už nebudeme kontrolovať **`$_POST['meno']`**, ale premennú **`$meno`**.

```
if ($zobraz_form) {
?>
<form method="post">
  <label for="meno">Meno:</label> <input type="text" name="meno"
id="meno" value="<?php if (isset($meno)) {echo $meno;} ?>"><br>
  <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
<?php
}
?>
```

Možno sa zdá, že položky, ktoré používateľ vyberá (tie, do ktorých nič nevypisuje, napr. výberová ponuka, začiarkávacie políčka, ...), netreba kontrolovať. To je však veľký omyl. Ak bude chcieť útočník napadnúť našu aplikáciu, vytvorí si kópiu nášho formulára, kde namiesto výberových prvkov vloží napr. textové polia (aby mohol zadávať príkazy). Útočník sa pokúsi odhadnúť, ako asi prebiehajú kontroly nášho formulára (prípadne ju vyskúša), zistí si povinné údaje a podľa toho si prispôsobí svoj formulár, ktorého spracovanie ale nasmeruje na náš formulár. Tým nám môže spôsobiť veľké škody.

PRÍKLAD 5.27

05/kontrola08.php

Majme formulár ako v príklade 5.15. Dorobme doňho overovanie hodnôt a kontroly. Položka meno (meno a priezvisko) musí mať minimálne 5 znakov, maximálne 50 znakov a musí obsahovať medzeru. Adresa musí mať minimálne 5 znakov, maximálne 100 znakov a tiež musí obsahovať medzeru.

Okrem funkcie `osetri()` (z príkladu 5.25) pridáme funkciu `spravne_meno()` a `spravna_adresa()`.

```
function spravne_meno($m) {
  return (strlen($m) >= 5) && (strlen($m) <= 50) &&
         (strpos($m, ' ') > 0);
}

function spravna_adresa($m) {
  return (strlen($m) >= 5) && (strlen($m) <= 100) &&
         (strpos($m, ' ') > 0);
}
```

OTÁZKA

Vedeli by ste tieto dve funkcie nahradiť jednou?

Upravme overovanie formuláru.

```
if (isset($_POST['tlacidlo'])) {
  // ošetrenie vstupov a nastavenie premenných
  if (isset($_POST['meno'])) {
    $meno = osetri($_POST['meno']);
  } else {
    $meno = '';
  }
  if (isset($_POST['adresa'])) {
    $adresa = osetri($_POST['adresa']);
  } else {
    $adresa = '';
  }
  if (isset($_POST['vyber'])) {
    $vyber = osetri($_POST['vyber']);
  } else {
    $vyber = '';
  }
  if ((spravne_meno($meno))  && (spravna_adresa($adresa)) &&
      ($vyber != '')) {
    // boli odoslané všetky údaje a zobrazíme rekapituláciu
    echo "meno: $meno<br>\n";
    echo "adresa: $adresa<br>\n";
    echo "tovar: {$tovar_nazvy[$vyber]}<br>\n";
    $zobraz_form = false;     // tu formulár nezobrazíme
  } else {
    echo "Nezadal si niektorú položku alebo nemá správny
          formát!<br>\n";
  }
}
```

A na záver upravíme zobrazovanie nastavených hodnôt vo formulári.

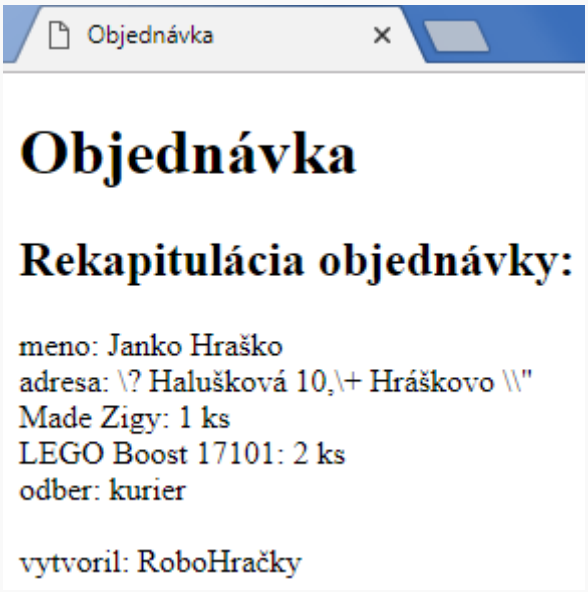
```
<form method="post">
  <label for="meno">Meno:</label>
     <input type="text" name="meno" id="meno" value="
     <?php if (isset($meno)) {echo $meno;} ?>"><br>

     <label for="adresa">Adresa:</label>
     <textarea name="adresa" id="adresa" cols="30" rows="4">
     <?php if (isset($adresa)) {echo $adresa;} ?>
     </textarea><br>

     <label for="vyber">vyber tovar:</label>
     <select name="vyber" id="vyber">
     <option value=''>--------------</option>
     <?php
     if (isset($vyber)) {
           vypis_tovarov($tovar_nazvy, $tovar_ceny, $vyber);
     } else {
           vypis_tovarov($tovar_nazvy, $tovar_ceny, '');
     }
     ?>
     </select><br>
     <input type="submit" name="tlacidlo" value="Odošli údaje">
</form>
```

ÚLOHA 5.28

Vyrobte nasledujúci formulár.

Ošetrite v ňom všetky odosielané položky a skontrolujte meno, adresu (vymyslite si vlastné kritériá) a skontrolujte, či bol zvolený aspoň 1 kus niektorého tovaru. Všetky položky formulára sú povinné.

Po korektnom odoslaní formulára sa zobrazí rekapitulácia:



Spätná väzba pri odosielaní formulára

DISKUSIA

Aké správy by sa mali zobrazovať pri nekorektne odoslanom formulári?

Ak chceme používateľom pomôcť pri vypĺňaní formulárov, nestačí oznámiť, že formulár bol nesprávne vyplnený. Mali by sme ich upozorniť aj na chyby. Má to význam predovšetkým vtedy, ak formulár obsahuje viac položiek.

PRÍKLAD 5.29

05/kontrola09.php

Vychádzajme z riešenia príkladu 5.27 a doplňme zobrazovanie informácií o chybách vo formulári.

Najskôr vytvorme funkciu, ktorá skontroluje, či je výber neprázdny.

```
function spravny_vyber($v) {
  return !empty($v);
}
```

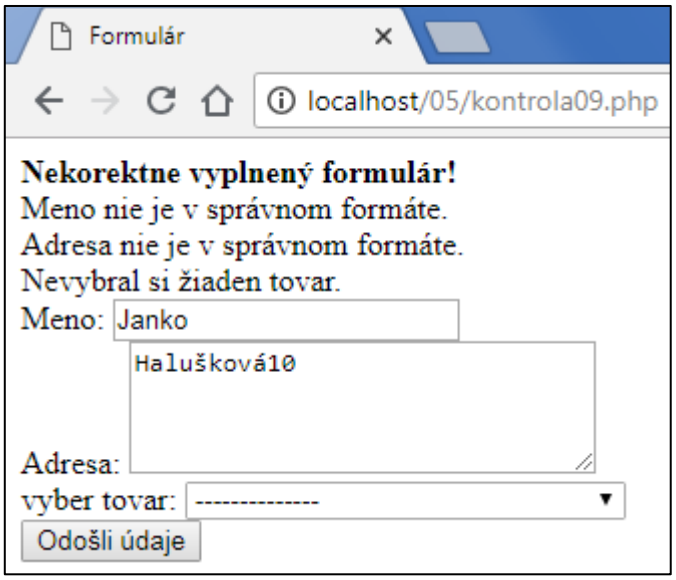
Použili sme funkciu `empty($premenna)`, ktorá zisťuje, či je premenná prázdna. Premenná je prázdna, ak nie je definovaná alebo sa jej hodnota rovná `FALSE` (`""` - prázdny reťazec, `0` - 0 ako celé číslo, `0.0` - 0.0 ako reálne číslo, `"0"` - 0 ako reťazec, `[]` - prázdne pole, `FALSE`, `NULL`). Pripomeňme si, že výkričník pred funkciou znamená negáciu.

Zoznam chýb si budeme ukladať do asociatívneho poľa `$chyby`, ktorého kľúčmi budú názvy prvkov formulára (tých, v ktorých sa vyskytli chyby). Ak po skončení overovania formulára bude toto pole prázdne, tak sa vo formulári žiadne chyby nenachádzajú, formulár nezobrazíme a môžeme vypísať napr. rekapituláciu, inak vypíšeme chyby a znovu zobrazíme formulár.

Kontrolu existencie položky, jej ošetrenie a kontrolu správnosti hodnoty sme spojili do jednej podmienky (spolu s else časťou), aby sme lepšie vedeli vypisovať chybu, kedy používateľ nezadal nič a kedy zadal hodnotu v nesprávnom formáte.

```
$zobraz_form = true;
$chyby = [];

if (isset($_POST['tlacidlo'])) {
  if (isset($_POST['meno']) && $_POST['meno']) {
    $meno = osetri($_POST['meno']);
    if (!spravne_meno($meno)) {
      $chyby['meno'] = 'Meno nie je v správnom formáte.';
    }
  } else {
    $meno = '';
    $chyby['meno'] = 'Nezadal si meno.';
  }

  if (isset($_POST['adresa']) && $_POST['adresa']) {
    $adresa = osetri($_POST['adresa']);
    if (!spravna_adresa($adresa)) {
      $chyby['adresa'] = 'Adresa nie je v správnom formáte.';
    }
  } else {
    $adresa = '';
    $chyby['adresa'] = 'Nezadal si adresu.';
  }

  if (isset($_POST['vyber']) && $_POST['vyber']) {
    $vyber = osetri($_POST['vyber']);
    if (!spravny_vyber($vyber)) {
      $chyby['vyber'] = 'Nevybral si žiaden tovar.';
    }
  } else {
    $vyber = '';
    $chyby['vyber'] = 'Nevybral si žiaden tovar.';
  }

  if (empty($chyby)) {
    echo "meno: $meno<br>\n";
    echo "adresa: $adresa<br>\n";
    echo "tovar: {$tovar_nazvy[$vyber]}<br>\n";
    $zobraz_form = false;
  } else {
    echo "<b>Nekorektne vyplnený formulár!</b><br>\n";
    foreach($chyby as $hodnota) {
      echo "$hodnota<br>\n";
    }
  }
}
```

Nesprávne vyplnený formulár by mohol vyzerať nasledovne:



ÚLOHA 5.30

Do úlohy 5.28 dorobte zobrazovanie chybových správ.

5.4 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto časti je oboznámiť študentov s vylepšenou prácou s formulármi, ako je zobrazovanie už odoslaných hodnôt vo formulári a zobrazovanie/nezobrazovanie formulára. Okrem toho sa zoznámia so základnými funkciami, ktoré súvisia s bezpečnosťou pri kontrole formulárov.

VÝKLAD

Táto časť je zameraná na:

- zobrazovanie odoslaných údajov vo formulári,
- skrývanie (nezobrazovanie) formulára,
- základnú kontrolu a overovanie vstupov od používateľa.

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

Zobrazenie odoslaných údajov znovu vo formulári

K diskusii: Naviesť študentov na to, že používatelia očakávajú, že to, čo už raz vo formulári vyplnili, ostane vyplnené, aj keď formulár odošlú neúplný alebo nekorektne vyplnený.

K textarea: Treba si dávať pozor na medzery medzi značkami elementu textarea, lebo sa zachovávajú- Ak medzi ne vkladáme PHP kód tak medzi otváracou značkou elementu textarea a PHP kódom by nemala byť medzera, rovnako aj na konci, teda `<textarea><?php ... ?></textarea>`.

K výberovej ponuke: Treba si dávať pozor na to, že medzi atribútmi v `<option>` musí byť medzera. Teda buď bude medzera priamo v HTML kóde (za atribútom value a pred PHP vsuvkou), alebo v PHP vsuvke sa bude vypisovať text `' selected'` (s medzerou na začiatku). To isté platí aj pre prepínače a začiarkávacie políčka.

Skrývanie formulára

K príkladu 5.15: So študentami diskutujte, prečo nie je dobré mať úplne rovnaký kód na rôznych miestach. Aké problémy to môže spôsobiť?

Kontrola vstupov od používateľa

Pri práci s formulármi platí zásadné pravidlo – **Nesmieme dôverovať vstupom od používateľa**. A preto treba každý prvok formulára testovať. Nesmieme sa totiž spoliehať na obmedzenia, ktoré sú vo formulári, napr. max. počet zadávaných znakov. Väčšina (nielen) začínajúcich tvorcov webových aplikácií (pri spracovaní formulárov) si myslí, že ak vo formulári môže používateľ zvoliť nejaké hodnoty, tak túto položku nemusí kontrolovať. To je však veľký omyl.

Ak bude chcieť útočník napadnúť našu aplikáciu, vytvorí si kópiu nášho formulára, kde namiesto výberových prvkov vloží napr. textové polia (aby mohol zadávať príkazy). Útočník sa pokúsi odhadnúť, ako asi prebiehajú kontroly nášho formulára (prípadne ju vyskúša) zistí si povinné údaje a podľa toho si prispôsobí svoj formulár, ktorý ale nasmeruje na náš formulár (atribút `action` v útočníkovom formulári bude obsahovať webovú adresu nášho formulára).

Funkcia `strip_tags(reťazec, povolene_elementy)` má aj voliteľný parameter povolene_elementy, v ktorom definujeme, ktoré elementy sa majú v reťazci zachovať.

Diskusia pri spätnej väzbe pri odosielaní formulára: Študentov treba naviesť na to, že nestačí správa o nekorektnom odoslaní formulára, ale aj informácia, pri ktorých položkách bola chyba. Môžete diskutovať aj o tom, kde a kedy by mali byť zobrazené informácie o požadovaných formátoch vypĺňaných údajov.

6 PHP - VKLADANIE KÓDU Z RÔZNYCH SÚBOROV

Predstavme si situáciu, že máme niekoľko stránok, ktoré majú rovnaké časti, napríklad navigáciu alebo pätičku. V prípade statických HTML stránok musíme tieto časti vložiť do všetkých stránok. Ak však v niektorej časti chceme zmeniť text, musíme to urobiť na všetkých stránkach. Podobne je to napríklad pri spoločných premenných, konštantách alebo funkciách, ktoré chceme využívať na viacerých stránkach. V takýchto prípadoch si môžeme spoločné informácie uložiť do samostatného súboru a do konkrétnej stránky ich len vložiť (pripojiť) podobne ako napr. pri kaskádových štýloch.

6.1 Pripojenie súboru - INCLUDE

Na pripojenie (vloženie) obsahu súboru (prakticky ľubovoľného typu) k PHP súboru môžeme využiť funkciu **`include(subor)`**, kde parameter **`subor`** je kompletná (relatívna alebo absolútna) cesta k pripájanému súboru. Ten istý súbor nesmieme vložiť viackrát (priamo alebo z iného súboru), inak PHP interpreter vyhlási chybu.

PRÍKLAD 6.1

Upravme stránky IT Pizza tak, aby hlavička, navigácia a päta boli v samostatných súboroch. Vychádzajme zo stránky `index.html` v priečinku `zdroj`.

06/zdroj/index.html

Aby sme však doň mohli "vložiť" obsah iného súboru, musíme ho premenovať na `index.php`.

Hlavičku spravíme ako PHP súbor, aby sme dynamicky mohli meniť titulok stránky. Súbor `hlavicka.php` bude obsahovať všetko od začiatku stránky po koniec elementu `<header>` zo súboru `index.php`:

06/riesenie1/hlavicka.php

```
<!doctype html>
<html>
<head>
...
</head>
<body>
  <header>
  ...
  </header>
```

Aj navigáciu spravíme ako PHP súbor, aby sme v nej mohli neskôr vypisovať aktuálny počet tovaru v nákupnom košíku. Súbor `navigacia.php` bude obsahovať celý element `<nav>`. V navigácii musíme zmeniť odkaz `index.html` na `index.php`:

06/riesenie1/navigacia.php

```
  <nav>
    <div id="kosik_nav">
    ...
    </div>
    <a href="index.php">ponuka</a>
    <a href="galeria.html">fotogaléria</a>
    <a href="rezervacia.html">rezervácia</a>
    <a href="praca.html">práca</a>
  </nav>
```

06/riesenie1/pata.html

Súbor `pata.html` bude obsahovať všetko od elementu `<footer>` po koniec súboru:

```
  <footer id="kontakt">
    <h2>Kontakt</h2>
    ...
  </footer>
</body>
</html>
```

Teraz môžeme upraviť súbor `index.php`. Zmažeme z neho celú časť, ktorá reprezentuje hlavičku, navigáciu a pätu a namiesto nich vložíme súbory `hlavicka.php`, `navigacia.php` a `pata.html`:

```
<?php
include('hlavicka.php');
include('navigacia.php');
?>
  <section id="ponuka">
    <h2>Ponuka</h2>
    ...
  </section>
  <aside id="akcia">
    ...
  </aside>
<?php
include('pata.html');
?>
```

Keď teraz zobrazíme `index.php`, stránka by mala vyzerať rovnako ako HTML verzia, z ktorej sme vychádzali.

Keď sa pozrieme na zdrojový kód stránky, tak budeme vidieť, že obsahuje všetky informácie z vkladaných súborov a aj zo súboru `index.php`.

To, čo sme spravili v súbore `index.php`, je akási šablóna:

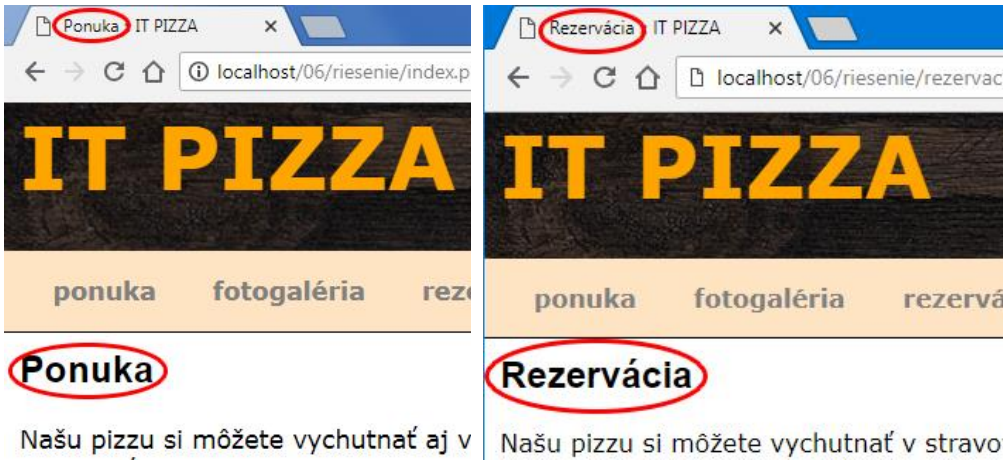
```
<?php
include('hlavicka.php');
include('navigacia.php');
?>
<!-- obsah hlavnej časti -->
<?php
include('pata.html');
?>
```

Túto šablónu môžeme použiť na každú stránku webového sídla IT pizza. Meniť sa bude len obsah hlavnej časti.

ÚLOHA 6.2

Upravte stránku `galeria.html` tak, aby ste na hlavičku, navigáciu a pätu použili súbory ako v príklade 6.1. Nezabudnite upraviť aj súbor `navigacia.php`.

Chceli by sme, aby sa titulok stránky menil podľa aktuálnej stránky. Okrem toho chceme zabezpečiť, aby sa rovnaký nadpis zobrazoval aj v obsahu hlavnej stránky, napr.:



PRÍKLAD 6.3

Upravme stránku ponuky tak, aby sa zobrazoval rovnaký text v titulku stránky aj v hlavnom nadpise obsahu stránky.

Musíme upraviť súbory `index.php` a aj `hlavicka.php`.

Na začiatok stránky `index.php` pridáme premennú `$nadpis`, ktorú potom vypíšeme v elemente `<h2></h2>`.

06/riesenie2/index.php

```
<?php
$nadpis = 'Ponuka';
include('hlavicka.php');
include('navigacia.html');
?>
  <section id="ponuka">
    <h2><?php echo $nadpis; ?></h2>
```

V súbore `hlavicka.php` doplníme do titulku vypisovanie premennej `$nadpis` (ktorú sme definovali v súbore `index.php` a definovali sme ju ešte pred vložením súboru `hlavicka.php`).

06/riesenie2/hlavicka.php

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title><?php echo $nadpis; ?> :: IT PIZZA</title>
  <link href="styly.css" rel="stylesheet">
```

Prečo to bude fungovať, keď premennú máme definovanú len v súbore `index.php`?

Súbor `hlavicka.php` nespúšťame v prehliadači samostatne, ale je súčasťou `index.php` (lebo sme ho do `index.php` vložili). Vykonáva sa len `index.php`, ktorý má prístup k premennej aj k obsahu hlavičky.

ÚLOHA 6.4

Do stránky `galeria.php` dorobte správne zobrazovanie titulku a nadpisu v hlavnej obsahovej časti.

6.2 Oddelenie obsahu stránky od funkcií

Väčšinou to budú údaje získané z databázy. My ich však budeme mať uložené v poliach.

Ponuku pízz na stránke `index.php` by sme chceli generovať z nejakých údajov. Údaje o pizziach by sa nám hodili aj na stránke nákupného košíka. Aby sme ich mali na jednom mieste, uložíme si ich do súboru `udaje.php`.

PRÍKLAD 6.5

Vytvorme súbor s údajmi o pizziach a pripojme ho tak, aby sme ho mohli využiť na stránke `index.php`. Potom na stránke `index.php` vygenerujme jednotlivé elementy `<article>`, ktoré reprezentujú pizze.

06/riesenie3/udaje.php

V súbore `udaje.php` údaje o pizziach rozdelíme do piatich polí.

```
<?php
$nazvy = ['Margherita', 'Cardinale', 'Funghi', 'Hawai',
'Prosciutto', 'Quattro Formaggi', 'Tonno'];
$popisy = ['paradajková omáčka, syr', 'paradajková omáčka, syr,
šunka', 'paradajková omáčka, syr, šampiňony', 'paradajková omáčka,
syr, šunka, ananás', 'paradajková omáčka, mozarella, šunka',
'paradajková omáčka, 4 druhy syra', 'paradajková omáčka, mozarella,
tuniak, cibuľa'];
$ceny = [3.15, 3.99, 5.05, 4.10, 5.89, 4.50, 4.05];
$alergeny = ['1, 7', '1, 7', '1, 7', '1, 7', '1, 7', '1, 7', '1, 4,
7'];
$subory = ['margerita.jpg', 'cardinale.jpg', 'funghi.jpg',
'hawai.jpg', 'prosciutto.png', 'formaggi.png', 'tonno.png'];
?>
```

Rovnaký index v každom poli reprezentuje údaje o jednej pizzi.

06/riesenie3/hlavicka.php

Súbor `udaje.php` vložíme na začiatok súboru `hlavicka.php`. Tým zabezpečíme, že ho nemusíme vkladať do ďalších súborov, v ktorých by sme tieto údaje mohli potrebovať. Budú prístupné vo všetkých súboroch, ktoré majú vložený súbor `hlavicka.php`.

```
<?php
include('udaje.php');
?>
<!doctype html>
<html>
```

Na stránke `index.php` môžeme teraz vygenerovať informácie o jednotlivých pizziach.

06/riesenie3/index.php

```
<?php
foreach($nazvy as $kluc => $hodnota) {
  echo "<article>\n";
  echo "<h3>$hodnota ({$ceny[$kluc]} &euro;)</h3>{$popisy[$kluc]}
<sub>{$alergeny[$kluc]}</sub><br>\n";
  echo '<form method="post">';
  echo '<button type="submit" name="pizza" value="' . $kluc .
'"><img src="obrazky/kosik.png" alt="nákupný košík" width="16"
height="16" title="do košíka">' . "</button>\n";
  echo "</form>\n";
  echo '<img src="obrazky/' . $subory[$kluc] . '" alt="pizza ' .
$hodnota . '" width="165" height="124"><br>';
  echo "</article>\n";
}
?>
```

Rovnako, ako sme údaje vyčlenili do samostatného súboru, môžeme tak spraviť aj s funkciami. Mnohé funkcie vieme využiť na viacerých stránkach, a tiež naše súbory budú prehľadnejšie, keď nebudú obsahovať funkcie.

PRÍKLAD 6.6

Upravme stránku `rezervacia.html` tak, aby sa hlavička, navigácia a päta vkladala z príslušných súborov. Potom upravme formulár - jeho spracovanie aj overovanie. Funkcie, ktoré budeme potrebovať, uložíme do súboru `funkcie.php`, ktorý, rovnako ako `udaje.php`, vložíme do hlavičky.

Najskôr premenujme súbor na `rezervacia.php` a odstráňme spoločné informácie pre všetky stránky. Pridajme aj premennú `$nadpis`.

```
<?php
$nadpis = 'Rezervácia';
include('hlavicka.php');
include('navigacia.php');
?>
  <section id="rezervacia">
    <h2><?php echo $nadpis; ?></h2>
    <div>
      ...
    </div>
    <form method="post">
      ...
    </form>
  </section>
<?php
include('pata.html');
?>
```

Nesmieme zabudnúť upraviť aj stránku `navigacia.php`.

06/riesenie4/navigacia.php

```
  <nav>
    ...
    <a href="rezervacia.php">rezervácia</a>
    <a href="praca.html">práca</a>
  </nav>
```

06/riesenie4/funkcie.php

Vráťme sa k súboru `rezervacia.php`. Upravme elementy `<select>` pre počet osôb, deň, rok a hodiny tak, aby jednotlivé elementy `<option>` vygenerovala funkcia `vypis_poloziek()` z príkladu 5.6. Vytvorme súbor `funkcie.php`, skopírujme doň funkciu `vypis_poloziek()` a pridáme nastavenie časovej zóny.

```
<?php
date_default_timezone_set('Europe/Bratislava');
function vypis_poloziek($od, $do, $oznac) {
...
}
?>
```

06/riesenie4/hlavicka.php

Súbor `funkcie.php` pripojme do `hlavicka.php`.

```
<?php
include('udaje.php');
include('funkcie.php');
?>
<!doctype html>
<html>
...
</html>
```

06/riesenie4/reyervacia.php
06/riesenie4/...

Teraz upravme súbor `rezervacia.php`. Nahraďme vypisovanie elementov `<option>` funkciou. Počet osôb bude mať hodnoty od 0 do 10, deň od 1 do 31, rok aktuálny rok a nasledujúci rok a hodiny budú od 10 do 20. Pre počet osôb nebude prednastavená žiadna hodnota (teda 0), pre ostatné aktuálny deň, rok a hodina.

```
<form method="post">
  ...
  <select name='pocet'>
    <?php echo vypis_poloziek(0, 10, 0); ?>
  </select><br>
  ...
  Dátum: <select name='den'>
    <?php echo vypis_poloziek(1, 31, date("j")); ?>
  </select>
  <select name='mesiac'>
    ...
  </select>
  <select name='rok'>
    <?php
    $rok = date("Y");
    echo vypis_poloziek($rok, $rok + 1, $rok);
    ?>
  </select><br>
  Čas: <select name='hod'>
    <?php echo vypis_poloziek(10, 20, date("G")); ?>
  </select>
  ...
</form>
```

ÚLOHA 6.7

Upravte vypisovanie mesiacov (z príkladu 6.6) tak, aby bol vo formulári prednastavený aktuálny mesiac. Môžete si na výpis mesiacov vytvoriť funkciu napr. `vypis_mesiacov($oznac)`, podobne ako je funkcia `vypis_poloziek()`. Na začiatku funkcie vytvorte pole, ktoré bude obsahovať názvy mesiacov. Hodnotami môžu byť čísla.

POZNÁMKA

Užitočná môže byť aj funkcia `vypis($hodnoty, $texty, $oznac)`, kde `$hodnoty` a `$texty` sú polia. Prípadne `vypis1($texty, $oznac)`, ak by nám stačili číselné hodnoty v elementoch `<option>`. Touto funkciou by sme mohli nahradiť funkciu `vypis_mesiacov()`, kde pole názvov mesiacov by sme mohli doplniť do súboru `udaje.php` a ako parameter ho zadať tejto funkcii.

PRÍKLAD 6.8

Do súboru `rezervacia.php` doplňme ešte odosielanie a overovanie formulára. Overovať budeme len položky meno, priezvisko a telefón a či používateľ zadal nenulový počet osôb a vybral mesto.

06/riesenie5/funkcie.php

Do súboru `funkcie.php` skopírujme funkcie `osetri()` a `spravne_meno()` z príkladu 5.29. Z funkcie `spravne_meno()` odstránime poslednú podmienku. Ďalej dopíšme funkciu `spravny_telefon($t)`. V tejto funkcii skontrolujeme, či premenná začína reťazcom "`+42`" a či má dĺžku aspoň 13 znakov.

```
function osetri($ret) {
  return quotemeta(addslashes(trim(strip_tags($ret))));
}
function spravne_meno($m) {
  return (strlen($m) >= 5) && (strlen($m) <= 50);
}
function spravny_telefon($t) {
  return (strlen($t) >= 13) && (strpos($t, '+42') === 0);
}
```

Keďže chceme, aby telefón začínal presne znakmi "`+42`" použijeme porovnanie `===`. Ak by sme použili porovnanie `==`, tak by stačilo zadať dostatočne dlhý reťazec a funkcia by dala hodnotu `TRUE`. Pri neexistencii tohto podreťazca by funkcia vrátila `FALSE`, ale vďaka automatickej konverzii pri porovnaní by porovnanie s hodnotou `0` bolo `TRUE`, čo nechceme.

V súbore `rezervacia.php` doplňme odosielanie a overovanie formulára.

```
<div>
  ...
<?php
$zobraz_form = true;
$chyby = [];
if (isset($_POST['tlacidlo'])) {
  // kontrola mena
  if (isset($_POST['meno']) && $_POST['meno']) {
    $meno = osetri($_POST['meno']);
    if (!spravne_meno($meno)) {
      $chyby['meno'] = 'Meno nie je v správnom formáte.';
    }
  } else {
    $meno = '';
    $chyby['meno'] = 'Nezadal si meno.';
  }
  // kontrola priezviska
  if (isset($_POST['priezvisko']) && $_POST['priezvisko']) {
    $priezvisko = osetri($_POST['priezvisko']);
    if (!spravne_meno($priezvisko)) {
      $chyby['priezvisko'] = 'Priezvisko nie je v správnom
formáte.';
    }
  } else {
    $priezvisko = '';
    $chyby['priezvisko'] = 'Nezadal si priezvisko.';
  }
  // kontrola telefónu
  if (isset($_POST['tel']) && $_POST['tel']) {
    $tel = osetri($_POST['tel']);
    if (!spravny_telefon($tel)) {
      $chyby['tel'] = 'Telefón nie je v správnom formáte.';
    }
  } else {
    $tel = '';
    $chyby['tel'] = 'Nezadal si telefón.';
  }
  // kontrola počtu
  if (!$_POST['pocet']) {
    $chyby['pocet'] = 'Nevybral si počet osôb.';
  }
  // kontrola mesta
  if (!isset($_POST['mesto'])) {
    $chyby['mesto'] = 'Nevybral si mesto.';
  }
  // ak nie sú žiadne chyby, zobrazíme rekapituláciu
  if (empty($chyby)) {
    echo "meno: $meno<br>\n";
    echo "priezvisko: $priezvisko<br>\n";
    echo "telefón: $tel<br>\n";
    echo "počet osôb: {$_POST['pocet']}<br>\n";
    echo "mesto: ";
    if ($_POST['mesto'] == 'ba') {
      echo "Bratislava<br>\n";
    } else {
      echo "Banská Bystrica<br>\n";
    }
    $zobraz_form = false;     // tu formulár nezobrazíme
  } else {
    ...
  }
```

```
}
?>
</div>

<?php
if ($zobraz_form) {
?>
<form method="post">
  ...
  <select name='pocet'>
    <?php
    if (isset($_POST['pocet'])) {
      echo vypis_poloziek(0, 10, $_POST['pocet']);
    } else {
      echo vypis_poloziek(0, 10, 0);
    }
    ?>
  </select><br>
  ...
  Dátum: <select name='den'>
    <?php
    if (isset($_POST['den'])) {
      echo vypis_poloziek(1, 31, $_POST['den']);
    } else {
      echo vypis_poloziek(1, 31, date("j"));
    }
    ?>
  </select>
  ...
  <select name='rok'>
    <?php
    $rok = date("Y");
    if (isset($_POST['rok'])) {
      echo vypis_poloziek($rok, $rok + 1, $_POST['rok']);
    } else {
      echo vypis_poloziek($rok, $rok + 1, $rok);
    }
    ?>
  </select><br>

  Čas: <select name='hod'>
    <?php
    if (isset($_POST['hod'])) {
      echo vypis_poloziek(10, 20, $_POST['hod']);
    } else {
      echo vypis_poloziek(10, 20, date("G"));
    }
    ?>
  </select>
  ...
</form>
<?php
}
?>
```

Aj napriek tomu, že formulár odošleme so správnym telefónnym číslom, vypíše sa chybová správa, že telefón nie je v správnom formáte. Problémom je znak `+` na začiatku telefónu, pred ktoré dá funkcia `quotemeta()` spätné lomítko. Tento problém vyriešime tak, že do premennej `$tel` dáme ošetrený reťazec bez prvého znaku (teda v prípade správne zadaného telefónu funkcia `quotemeta()` pridá spätné lomítko `\`, ktoré odstránime).

```
if (isset($_POST['tel']) && $_POST['tel']) {
  $tel = substr(osetri($_POST['tel']), 1);
  if (!spravny_telefon($tel)) {
    $chyby['tel'] = 'Telefón nie je v správnom formáte.';
  }
} else {
  $tel = '';
  $chyby['tel'] = 'Nezadal si telefón.';
}
```

ÚLOHA 6.9

Do predchádzajúceho príkladu doplňte zobrazovanie všetkých vyplnených údajov vo formulári a v rekapitulácii vypisovanie ostatných údajov.

ÚLOHA 6.10

Doplňte stránku `praca.html` o funkčnosť formulára aj s jeho ošetrením a vypisovaním chýb pri nekorektne zadaných údajoch. Nezabudnite na správne zobrazovanie titulku a nadpisu stránky.

ÚLOHA 6.11

Upravte ostatné stránky IT pizza tak, aby sa na spoločné údaje použilo vkladanie príslušných súborov.

6.3 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto časti je oboznámiť študentov s možnosťami vkladania rôznych súborov do PHP kódu. Dokážeme tak sprehľadniť a zefektívniť kód. Zameriame sa predovšetkým na vkladanie PHP kódu prípadne HTML kódu.

VÝKLAD

Táto časť je zameraná na:

- vkladanie iného kódu (HTML alebo PHP) z externých súborov do PHP kódu.

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

K príkladu 6.1:

Pri všetkých súboroch, ktoré vkladme do PHP súborov treba kontrolovať kódovanie, aby bolo rovnaké ako je PHP súbor, do ktorého vkladáme (UTF-8).

Hlavičku sme uložili do súboru `hlavicka.php`, pričom by súbor mohol zostať aj `hlavicka.html`. Pre študentov by to však mohlo byť mätúce, že v HTML súbore je PHP vsuvka. Na prvý pohľad by to nemalo fungovať. Ale keďže súbor `hlavicka.php` (resp. `hlavicka.html`) nevoláme priamo, ale vkladáme do `index.php`, tak PHP kód sa vykonáva až v súbore `index.php`.

K príkladu 6.8:

Odstránenie spätných lomítok z funkcie `quotemeta()` môžeme aj pomocou funkcie `stripslashes()`.

7 PHP - TRVALÉ UKLADANIE ÚDAJOV

Doteraz sme si ukázali ako spracovať formulár, ale zatiaľ sme odoslané údaje z neho nikde neukladali. To si ukážeme v tejto kapitole. Možností, ako trvale ukladať údaje, je niekoľko - do súborov, do databáz,… Dnešným štandardom je ukladanie údajov do databáz, ale problematika PHP a databázy je príliš rozsiahla, preto sa jej v tejto učebnici nebudeme venovať.

So súbormi budeme pracovať podobne ako s textovými súbormi v nejakom inom programovacom jazyku. Do súboru teda môžeme zapisovať len čísla a reťazce. Ak chceme do súboru zapísať inú štruktúru, musíme ju prekonvertovať na tieto typy.

Pri zapisovaní do súborov si musíme stanoviť, v akom formáte si budeme jednotlivé údaje v súbore pamätať. Predpokladajme, že máme objednávkový formulár s údajmi meno, adresa a tovar. Máme niekoľko možností:

- pamätať si každý údaj v samostatnom riadku, napr.
  Janko Hraško
  Cícerová 27 940 01 Košice
  Lego Mindstorms,
- pamätať si všetky údaje z jednej objednávky v jednom riadku, oddelené konkrétnym oddeľovačom, napr.
  Janko Hraško#Cícerová 27 940 01 Košice#Lego Mindstorms
- pamätať si údaje v nejakej štruktúre, napr. XML, JSON, ...,
- …

Každý zo spôsobov má svoje výhody aj nevýhody.

DISKUTUJTE

Diskutujte o výhodách a nevýhodách rôznych spôsobov uchovávania údajov.

7.1 Uloženie odoslaných údajov do súboru

PRÍKLAD 7.1

Budeme dopĺňať kód, ktorý sme vytvorili v rámci príkladu 5.29. Po odoslaní formulára zapíšeme *meno* do súboru `objednavka.txt`.

07/objednavka-zapis01.php

Zapísanie údajov urobíme v časti `if (empty($chyby)) { }`, kde sme vypísali rekapituláciu objednávky. Tá zostane, ale pridáme zapísanie do súboru.

Na zapísanie údajov do súboru môžeme využiť funkciu **`file_put_contents(subor, udaje)`**. Táto funkcia zapíše reťazec alebo pole do súboru. Ak súbor neexistuje, vytvorí sa. Ak už súbor existoval, vymaže sa všetok obsah a nahradí sa novým (prepíše sa). Súbor zadávame ako názov súboru, prípadne spolu s cestou.

Neodporúčame zapisovať pole, lebo všetky prvky sa zapíšu do jedného riadka bez akéhokoľvek oddeľovača.

```php
if (empty($chyby)) {
  echo "meno: $meno<br>\n";
  echo "adresa: $adresa<br>\n";
  echo "tovar: {$tovar_nazvy[$vyber]}<br>\n";
  $zobraz_form = false;     // tu formulár nezobrazíme
  // zapíš údaje do súboru
  file_put_contents('objednavka.txt', $meno);
} else {
...
}
```

Súbor `objednavka.txt` bude vyzerať nasledovne:

objednavka-zapis01.php | objednavka.txt

```
1  Janko Hraško
```

07/objednavka-zapis02.php

PRÍKLAD 7.2

Zapíšme do súboru `objednavka.txt` všetky údaje z objednávky (*meno*, *adresa*, *tovar*). Každý údaj zapíšme na samostatný riadok.

Musíme si uvedomiť, že pomocou funkcie `file_put_contents()` nemôžeme zapísať každý údaj samostatne, lebo funkcia súbor najprv vymaže. Teda by sa nám zachoval len posledný údaj. Preto si najprv pripravíme reťazec, v ktorom sú jednotlivé údaje oddelené znakmi konca riadku a až ten nakoniec zapíšeme do súboru.

```php
if (empty($chyby)) {
...
  // zapíš údaje do súboru
  $udaje = $meno . "\n" . $adresa . "\n" . $vyber;
  file_put_contents('objednavka.txt', $udaje);
} else {
...
}
```

Súbor `objednavka.txt` bude vyzerať nasledovne:

objednavka-zapis02.php | objednavka.txt

```
1  Janko Hraško
2  Halušková 10, Hráškovo
3  MZ
```

ÚLOHA 7.3

Urobte niekoľko objednávok z aplikácie v príklade 7.2 a po každej si pozrite súbor `objednavky.txt`. Skúšajte rôzne dlhé texty v položke *adresa*. Platí pri všetkých pravidlo, že jeden riadok obsahuje informácie o jednej položke?

Ak budeme do položky *adresa* vkladať len údaje v jednom riadku, bude všetko v poriadku. Ak však do adresy vložíme niekoľko skutočných riadkov (stlačíme kláves *Enter*), potom vo

výstupnom súbore budeme mať viac ako tri riadky (adresa bude zaberať viac riadkov). Toto však môže spôsobiť problém pri čítaní súboru, lebo nebudeme vedieť, koľko riadkov máme prečítať. Riešení je niekoľko. Jedným z nich je, že pred zápisom adresy do súboru konce riadkov odstránime, resp. ich nahradíme iným znakom (napr. čiarkou alebo bodkočiarkou).

My si ukážeme iný spôsob, ktorý zachová pôvodné formátovanie údajov. Riešením je využiť štruktúru **JSON** (JavaScript Object Notation), ktorá je dnes bežná v rôznych webových aplikáciách a vieme s ňou pracovať v rôznych programovacích jazykoch.

7.2 JSON

JSON (JavaScript Object Notation) je jeden z najpoužívanejších formátov pre výmenu údajov (aj rôznych typov) na webe. Ľudia ho dokážu ľahko čítať a aj doň zapisovať. Programy ho dokážu ľahko analyzovať aj generovať.

Má veľmi dobrú podporu v programovacích jazykoch, napr. C, C++, C#, Delphi, Java, JavaScript, Lisp, Matlab, Perl, PHP (od 5.2 natívne), Python, Ruby...

Do JSON môžeme uložiť tieto typy: číslo (celé, reálne), textový reťazec v úvodzovkách, logickú hodnotu, pole, objekt. Príklady:

- [0, "PHP", false] - jednoduché pole
- ['MZ':38.90, 'SSW':100.90, 'LB': 141.90, 'OB':65.90] - asociatívne pole
- {"osoba": {"name": "Web", "age": 25}, "login": false, "opravnenia": [1,2,8]} – objekt

7.2.1 Využitie JSON v PHP

V PHP máme funkciu `json_encode()`, ktorá dokáže pretransformovať ľubovoľnú premennú do formátu JSON - vytvorí jeden reťazec.

Všetky tri údaje z objednávky (meno, adresa, tovar) chceme zapísať do jedného riadku. Preto z nich najskôr vytvoríme pole, ktoré potom prevedieme do JSON formátu a zapíšeme do súboru.

PRÍKLAD 7.4

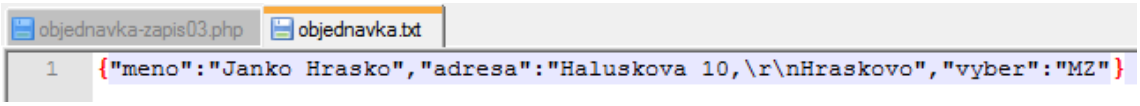
Zapíšme do súboru `objednavka.txt` všetky údaje z objednávky (*meno*, *adresa*, *tovar*) vo formáte JSON.

07/objednavka-zapis03.php

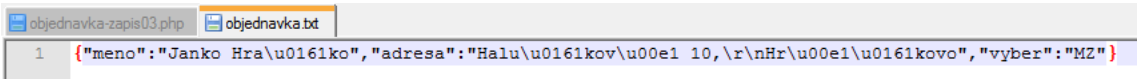
Všetky údaje si najprv vložíme do poľa `$udaje`. To následne pretransformujeme do formátu JSON pomocou funkcie `json_encode($udaje)` a výsledný reťazec zapíšeme do súboru.

```
if (empty($chyby)) {
...
  // zapíš údaje do súboru
  $udaje = ['meno' => $meno, 'adresa' => $adresa, 'vyber' =>
$vyber];
  $udaje_json = json_encode($udaje);
  file_put_contents('objednavka.txt', $udaje_json);
} else {
...
```

Súbor `objednavka.txt` bude vyzerať nasledovne:



Ak píšeme aj s diakritikou, tak súbor bude vyzerať:



Všimnite si, že diakritické znamienka sú kódované.

7.3 Načítanie údajov zo súboru

Ukázali sme si, ako údaje zapisovať do súboru. Poďme ich teraz načítať a zobraziť na stránke.

07/objednavka-vypis01.php

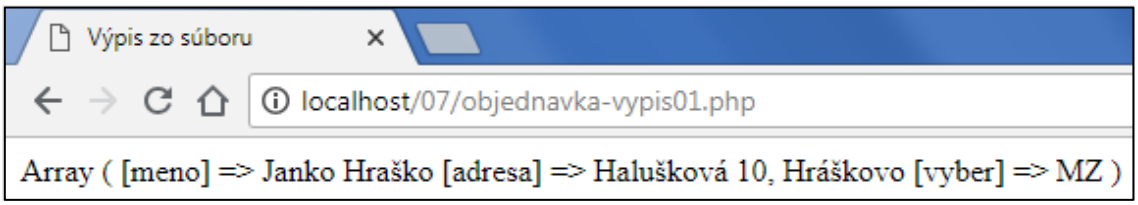
PRÍKLAD 7.5

Vypíšme objednávku, ktorá je v súbore `objednavka.txt` uložená vo formáte JSON. Z pôvodnej objednávky potrebujeme len polia s názvami a cenami tovaru.

Obsah súboru načítame pomocou funkcie **`file_get_contents(subor)`**, ktorá načíta celý obsah súboru ako jeden reťazec. Keďže v súbore je len JSON reťazec, tak ho dekódujeme pomocou funkcie **`json_decode(reťazec)`**. Táto funkcia reťazec štandardne konvertuje na objekt, čo nám nevyhovuje. Avšak funkcii `json_decode()` môžeme dať druhý parameter (typu *Boolean*) **`json_decode(reťazec, TRUE / FALSE)`**, ktorý keď nastavíme na `TRUE`, tak funkcia dekóduje reťazec ako asociatívne pole. S takým poľom už môžeme ďalej pracovať. Mali by sme dostať rovnaké pole, ktoré sme si pripravili pred zápisom do súboru. Zatiaľ ho vypíšeme pomocou funkcie `print_r()`.

```
<body>
<?php
// vypíš údaje zo súboru
$udaje_json = file_get_contents('objednavka.txt');
$udaje = json_decode($udaje_json, true);
print_r($udaje);
?>
</body>
```

Vidíme, že funkcia pole vypíše (dostala korektné pole) a obsahuje 3 kľúče - *meno*, *adresa*, *vyber*.



Čo by sa však stalo, keby súbor neexistoval? Funkcia `file_get_contents()` by vypísala upozornenie (*Warning*). Preto by sme nemali zo súboru čítať, pokým si nie sme istí, že existuje.

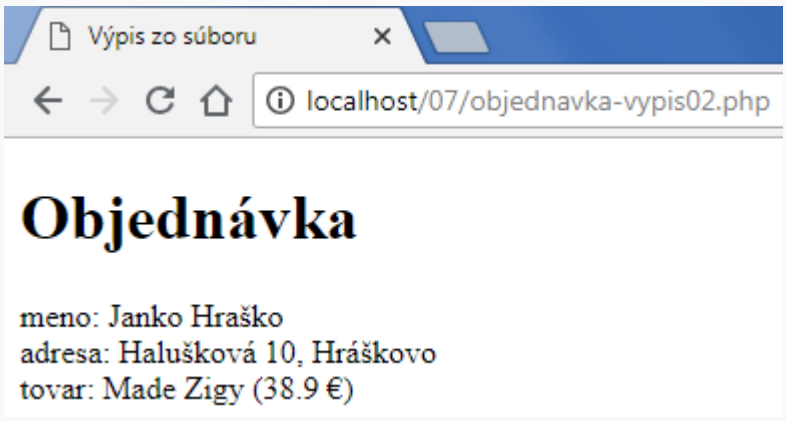
Na otestovanie, či súbor existuje, môžeme využiť funkciu **`file_exists(súbor)`**, ktorá vráti `TRUE`, ak súbor existuje, resp. `FALSE`, ak súbor neexistuje.

```
<?php
// test, či súbor existuje
if (file_exists('objednavka.txt')) {
  // vypíš údaje zo súboru
  $udaje_json = file_get_contents('objednavka.txt');
  $udaje = json_decode($udaje_json, true);
  print_r($udaje);
} else {
  echo 'Vstupný súbor neexistuje.';
}
?>
```

PRÍKLAD 7.6

07/objednavka-vypis02.php

Upravme výpis objednávky, aby vyzeral ako na nasledujúcom obrázku.



Meno a adresu vypíšeme štandardne cez premenné `$udaje['meno']` a `$udaje['adresa']`. Názov tovaru a jeho cenu však musíme zistiť podľa kľúča z polí `$tovar_nazvy` a `$tovar_ceny`. Kľúčom bude hodnota `$udaje['tovar']`.

```
echo "<h1>Objednávka</h1>\n";
echo "meno: {$udaje['meno']}<br>\n";
echo "adresa: {$udaje['adresa']}<br>\n";
echo "tovar: {$tovar_nazvy[$udaje['vyber']]}
({$tovar_ceny[$udaje['vyber']]} €)<br>\n";
```

Objednávka sa nám pekne vypíše. Formátovať ju môžeme ľubovoľne pomocou HTML elementov.

Avšak adresa je v jednom riadku. Pričom, keď sme ju odosielali a ukladali do súboru, bola v dvoch riadkoch. Využitím JSON formátu sme chceli toto riadkovanie zachovať, čo sa nám zatiaľ nepodarilo. Ak si však pozrieme zdrojový kód výpisu objednávky, zistíme, že v zdrojovom kóde je *adresa* v dvoch riadkoch.

```
8  <h1>Objednávka</h1>
9  meno: Janko Hraško<br>
10 adresa: Halušková 10,
11 Hráškovo<br>
12 tovar: Made Zigy (38.9 €)<br>
```

Teda zalomenie riadka (`\n`) sa zachovalo, ale aby sme to videli aj v prehliadači, musíme využiť štandardnú PHP funkciu **`nl2br(reťazec)`**, ktorá v zadanom reťazci nahradí všetky znaky `\n` reťazcami `<br>`. Takto upravený reťazec bude výsledkom funkcie `nl2br()`.

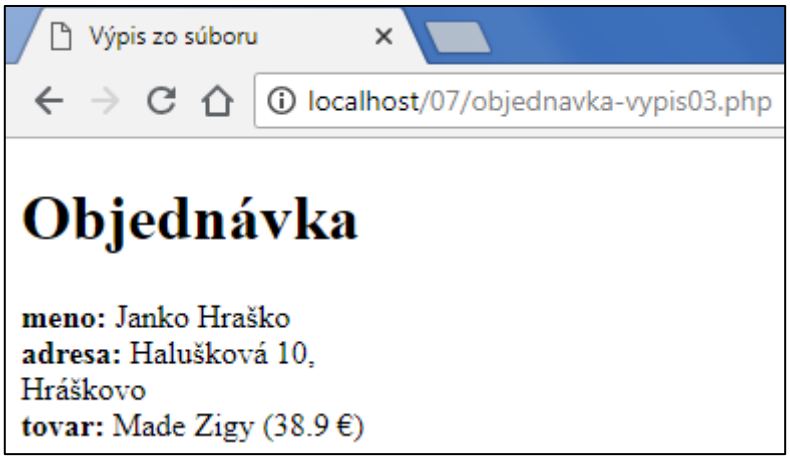
07/objednavka-vypis03.php

PRÍKLAD 7.7

Upravme výpis adresy objednávky pomocou funkcie `nl2br()`.

```
echo "<h1>Objednávka</h1>\n";
echo "meno: {$udaje['meno']}<br>\n";
echo "<strong>adresa:</strong> " . nl2br($udaje['adresa']) .
"<br>\n";
echo "tovar: {$tovar_nazvy[$udaje['vyber']]}
({$tovar_ceny[$udaje['vyber']]} €)<br>\n";
```

Výpis môže vyzerať nasledovne:



ÚLOHA 7.8

Do formulára z úlohy 5.28 dorobte zapisovanie do súboru a vytvorte aj stránku, ktorá zobrazí zapísané údaje..

7.4 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto časti je oboznámiť študentov s jednoduchou prácou so súbormi (čítanie a zapisovanie). Takisto sa oboznámia s formátom JSON.

VÝKLAD

Táto časť je zameraná na:

- zápis do súboru
- čítanie zo súboru
- kódovanie údajov do JSON formátu a jeho spätné dekódovanie do PHP premenných

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

Problematike samotných databáz sa venuje predmet *Databázy*. Uploadom súborov sa v tejto učebnici nezaoberáme.

Všetky príklady v kapitole zapisujú len 1 objednávku do súboru. Každé odoslanie objednávky vymaže zo súboru predchádzajúce údaje. V tejto časti uvádzame príklad, kde sa do súboru zapisujú všetky objednávky postupne (ako pole polí). K tomu je upravený aj výpis súboru, kde sa vypisujú všetky objednávky. V kapitole 4 sme pre študentov neuvádzali túto štruktúru (pole polí), preto aj v tejto kapitole použitie tejto štruktúry štandardne neuvádzame.

Súbory `objednavka-zapis01.php` a `objednavka-zapis02.php` zapisujú do súboru údaje nie vo formáte JSON, preto nebudú fungovať súbory s výpismi (`objednavka-vypis01---03`).

Uloženie odoslaných údajov do súboru

Funkcia `file_put_contents()` pred zapísaním údajov štandardne vymaže obsah súboru. V prípade potreby môžeme funkcii zadať ďalší parameter (konštantu **`FILE_APPEND`**), ktorým sa pôvodný obsah nevymaže, ale zapisované údaje sa pridajú na koniec súboru.

```
file_put_contents(subor, udaje, FILE_APPEND)
```

JSON

Do JSON môžeme uložiť tieto typy:

- JSONString – textový reťazec
  - Musí byť v úvodzovkách, nie apostrofoch
- JSONNumber – číslo (celočíselné alebo reálne)
  - Desatinné číslo nemôže začínať bodkou, napr. .7 nie je platný zápis, musíme napísať 0.7

- JSONBoolean – logická hodnota
  - true, false
- JSONNull – hodnota null
- JSONArray – pole
  - Uzatvorené v hranatých zátvorkách [ ]
  - Jednotlivé hodnoty oddeľujeme čiarkou
  - Môžeme vložiť ďalšie pole či objekt
  - Príklady:
    - [1, 2, 3]
    - [0, "PHP", false]
    - [[1, 2], null, "karta"]
- JSONObject – objekt
  - Nie je to plnohodnotný objekt, obsahuje len údaje, nie metódy.
  - Kľúče sú typu string, musia byť v úvodzovkách
  - Môžeme vkladať ďalšie objekty
  - Príklady:
    - {"x": 80, "y": 120}
    - {"osoba": {"name": "Web", "age": 25}, "login": false, "zaluby": null, "opravnenia": [1,2,8]}

Užitočné zdroje a pomôcky:

- domovská stránka JSON – json.org
- JSON validátor s formátovaním kódu – jsonlint.com
- JSON Editor Online – jsoneditoronline.org
- Code Beautify (online JSON editor) – codebeautify.org/online-json-editor
- JSON editor priamo v Google Chrome – https://chrome.google.com/webstore/detail/json-editor/lhkmoheomjbkfloacpgllgjcamhihfaj

Zapisovanie viacerých objednávok do súboru a čítanie z tohto súboru

07/objednavka-zapis04-MP.php

PRÍKLAD

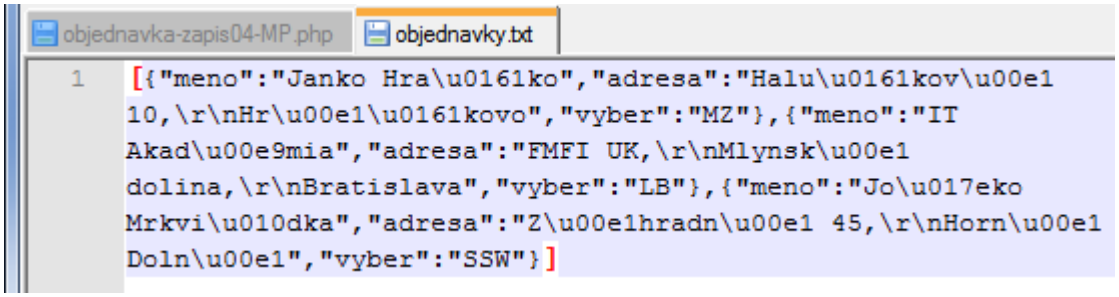
Upravme zapisovanie objednávky do súboru `objednavky.txt` tak, aby sa uchovávali všetky objednávky. Využijeme štruktúru pole polí. Tá sa nám ľahko zakóduje do JSON.

Pred zapísaním novej objednávky do súboru si najprv načítame všetky objednávky zo súboru (`$udaje_json`), dekódujeme do poľa polí (`$udaje_vsetky`), pridáme do neho pole nových údajov (`$udaje`) a až potom zapíšeme nové pole všetkých objednávok zakódované do JSON do súboru.

```
// test, či súbor existuje
if (file_exists('objednavky.txt')) {
  // načítaj pôvodné údaje zo súboru (ako JSON reťazec)
  $udaje_json = file_get_contents('objednavky.txt');
```

```
  // dekóduj JSON reťazec do poľa
  $udaje_vsetky = json_decode($udaje_json, true);
} else {
  // vytvor prázdne pole
  $udaje_vsetky = [];
}
// vytvor pole s údajmi o objednávke
$udaje = ['meno' => $meno, 'adresa' => $adresa, 'vyber' => $vyber];
// pole s údajmi o objednávke vložíme do poľa všetkých objednávok
$udaje_vsetky[] = $udaje;
// zakóduj všetky objednávky z poľa do JSON reťazca
$udaje_json = json_encode($udaje_vsetky);
// zapíš všetky objednávky do súboru
file_put_contents('objednavky.txt', $udaje_json);
```

Súbor `objednavky.txt` bude vyzerať nasledovne:



Všimnite si, že diakritické znaky sú kódované, preto sa texty trochu ťažšie čítajú. Pri spätnom načítaní a zobrazení na stránke sa však zobrazujú správne. V prípade potreby to treba študentom vysvetliť.

PRÍKLAD

Vypíšme všetky objednávky zo súboru `objednavky.txt`. Pred každou objednávkou vypíšme jej číslo.

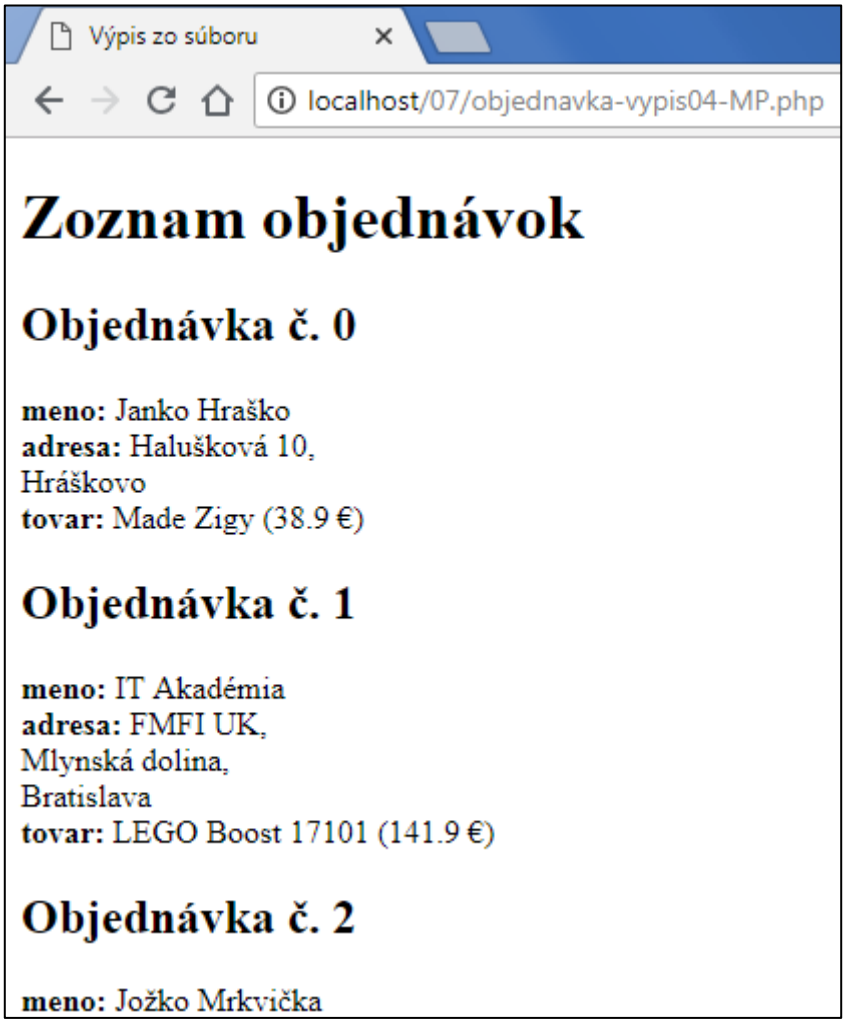
07/objednavka-vypis04-MP.php

Po načítaní súboru vo formáte JSON dekódujeme obsah do poľa polí (`$udaje_vsetky`). Toto pole potom postupne prejdeme a vypíšeme. Využijeme pôvodný výpis objednávky, ktorý len vnoríme do cyklu. Pridáme aj výpis čísla objednávky.

```
// načítaj pôvodné údaje zo súboru (ako JSON reťazec)
$udaje_json = file_get_contents('objednavky.txt');
// dekóduj JSON reťazec do poľa
$udaje_vsetky = json_decode($udaje_json, true);
echo "<h1>Zoznam objednávok</h1>\n";
foreach($udaje_vsetky as $kluc => $udaje) {
// v poli $udaje bude 1 objednávka
//   $kluc++;    // Objednávky sú číslované od 0, týmto začneme
číslovať od 1
  echo "<h2>Objednávka č. $kluc</h2>\n";
  echo "<strong>meno:</strong> {$udaje['meno']}<br>\n";
  echo "<strong>adresa:</strong> " . nl2br($udaje['adresa']) .
"<br>\n";
  echo "<strong>tovar:</strong> {$tovar_nazvy[$udaje['vyber']]}
({$tovar_ceny[$udaje['vyber']]} €)<br>\n";
}
```

Keďže objednávky sú číslované od 0 (počiatočný index v poli), môžeme pridať príkaz `$kluc++;`, ktorým začneme číslovať od 1.

Výpis môže vyzerať nasledovne:



8 PHP - PAMÄTANIE ÚDAJOV

V predchádzajúcich kapitolách sme vytvorili základ internetového obchodu s rôznymi vylepšeniami. Ukázali sme si, ako ukladať údaje z objednávky, ale doteraz sme neriešili tzv. košík - teda možnosť priebežne si budovať objednávku pomocou nákupného košíka, ktorý až na záver odošleme. Budovanie a prácu s košíkom si ukážeme v tejto kapitole. A takisto, ako tento mechanizmus využiť aj iným spôsobom.

Pamätanie si údajov v košíku vlastne znamená, že údaje si musíme pamätať v rámci rôznych stránok - napr. na stránke *Ponuka*, *Košík*, prípadne ďalších. Toto môžeme dosiahnuť napr. pomocou tzv. **SESSION**.

V SESSION si môžeme ukladať rôzne informácie (udržiavať nejaký "stav"), napr. kto je prihlásený, obsah nákupného košíka, atď. Pri spustení SESSION na našej stránke vygeneruje PHP unikátne tzv. SESSION ID, ktoré je uložené na serveri, ale kópia je aj na strane prehliadača. Ak však už v danom prehliadači existuje SESSION ID pre danú adresu, nevygeneruje sa nové SESSION ID, ale PHP zistí, že už existuje, a tak len obnoví platnosť existujúceho ID. Ak by sme však rovnakú stránku otvorili v inom prehliadači na tom istom počítači, PHP vygeneruje iné nové SESSION ID, lebo z pohľadu PHP je to iný klient.

SESSION zaniknú zrušením alebo zatvorením prehliadača.

Pri práci so SESSION platí základné pravidlo: spustenie SESSION musí byť pred ľubovoľným výpisom (PHP, HTML,...). Výpisom môže byť napríklad aj prázdny riadok v zdrojovom kóde stránky (častá nepozornosť začiatočníkov).

SESSION je v skutočnosti asociatívne pole, s ktorým pracujeme ako s iným asociatívnym poľom. Pole **`$_SESSION`** sa vytvorí, keď spustíme príkaz **`session_start()`**.

Príkaz `session_start()` naštartuje (obnoví) používanie SESSION na stránke. Musí byť pred ľubovoľným výpisom (PHP alebo HTML) na stránke aj prázdnym riadkom v zdrojovom HTML kóde.

Po naštartovaní SESSION si môžeme do poľa `$_SESSION` ukladať údaje priradením **`$_SESSION[kluc] = hodnota`**.

PRÍKLAD 8.1

08/session1/index.php

Uložme si do SESSION číslo pizze, ktoré bude odoslané vo formulári (`$_POST['pizza']`). Aby sme čísla pízz mali na jednom mieste, uložíme si všetky čísla do `$_SESSION['kosik']`, čo bude pole všetkých odoslaných indexov.

Pri každej pizzi máme jeden formulár, v ktorom je len tlačidlo `<button>`. Všetky tlačidlá (pri všetkých pizziach) sú pomenované rovnako (`name="pizza"`), ale líšia sa hodnotou (atribút `value`). Toto riešenie nám uľahčuje kontrolu odoslania formulára v tom, že je odoslané len 1 tlačidlo (vždy s rovnakým menom) a podľa hodnoty vieme zistiť konkrétnu pizzu.

```
if (isset($_POST['pizza'])) {
  $_SESSION['kosik'][] = $_POST['pizza'];
}
```

Ak by sme si chceli dočasne vypísať, čo máme uložené v SESSION, môžeme využiť funkciu `print_r($_SESSION)`.

Samozrejme, že použitie funkcie `print_r()`, je len dočasné. Obsah košíka (SESSION) si vypíšeme v peknom tvare. Budeme ho vypisovať na samostatnej stránke *Košík* (`kosik.php`).

08/session1/
kosik.php

PRÍKLAD 8.2

Vypíšme obsah nákupného košíka uloženého v SESSION (`$_SESSION['kosik']` - stačí, keď je neprázdny) ako zoznam tovarov (názvy z poľa `$nazvy`) s ich cenami (z poľa `$ceny`) na stránke `kosik.php`. Počas vypisovania tovarov si zároveň budeme počítať celkovú cenu košíka (v premennej `$celk_cena`). V prípade, že v košíku nie je uložený žiaden tovar, vypíšeme správu.

```
if (isset($_SESSION['kosik'])) {
  $celk_cena = 0;
?>
  <h3>V košíku máte:</h3>
  <ol>
<?php
  foreach($_SESSION['kosik'] as $h) {
    echo "<li>{$nazvy[$h]}: {$ceny[$h]} &euro;</li>\n";
    $celk_cena += $ceny[$h];
  }
  echo "</ol>\n";
  echo "<p>Celková cena objednávky: <strong>$celk_cena
&euro;</strong></p>\n";
} else {
  echo '<p>Nákupný košík je prázdny</p>';
}
```

Ak by sme však uvedený kód vyskúšali, nefungoval by. Hlavným problémom je to, že chceme využívať premennú `$_SESSION`, ale nemáme naštartované SESSION. Hneď na začiatku stránky totiž musíme mať príkaz `session_start()`. Vďaka nemu sa nám sprístupnia údaje zo SESSION, ktoré sme uložili na stránke `index.php`.

```
<?php
session_start();
$nadpis = 'Nákupný košík';
...
?>
```

Výsledná stránka by mohla vyzerať napr. ako na nasledujúcom obrázku.

Ďalším krokom v našej aplikácii by mohlo byť vymazanie košíka. Či už celého alebo vymazávanie jednotlivých tovarov. Zatiaľ vyriešime ten jednoduchší variant - kompletné vymazanie košíka.

PRÍKLAD 8.3

Pridajme na stránku nákupného košíka tlačidlo s možnosťou vymazania celého obsahu košíka uloženého v SESSION.

08/session2/kosik.php

Aby sa nám tlačidlo zobrazilo vedľa zoznamu tovaru, vložíme kód s formulárom medzi nadpis h3 a odrážky.

```
  <h3>V košíku máte:</h3>
  <form method="post" class="form_kosik">
    <input type="submit" name="zrus" value="Zruš obsah košíka"
class="odosli_velke">
  </form>
  <ol>
```

Potrebujeme však pridať aj kód na spracovanie odoslania tohto tlačidla. Tlačidlo spracujeme ako iné odosielacie tlačidlo - otestujeme hodnotu `$_POST['zrus']` a ak existuje, tak zrušíme obsah košíka v premennej `$_SESSION['kosik']`.

```
<?php
session_start();
$nadpis = 'Nákupný košík';

if (isset($_POST['zrus'])) {
  unset($_SESSION['kosik']);
}
include('hlavicka.php');
include('navigacia.php');
?>
...
```

Je dôležité kód spracovania tlačidla na zrušenie košíka vložiť na začiatok stránky. Po kliknutí na tlačidlo sa načíta stránka od začiatku, kde zrušíme obsah košíka a následne na stránke

zobrazíme informáciu o prázdnom košíku. Prvok poľa zrušíme pomocou funkcie **`unset(čosi)`**. V našom prípade `unset($_SESSION['kosik']);` zruší prvok poľa. Ale táto funkcia vymaže aj jednoduchú premennú.

V tomto momente máme hotovú základnú funkčnosť nákupného košíka. Väčšina internetových obchodov však na každej stránke priebežne zobrazuje nejakú informáciu o košíku - napr. počet tovarov alebo aktuálnu cenu objednaného tovaru. Na stránke pizzérie máme pripravenú časť na zobrazovanie stručnej informácie o košíku v navigácii.

08/session2/navigacia.php

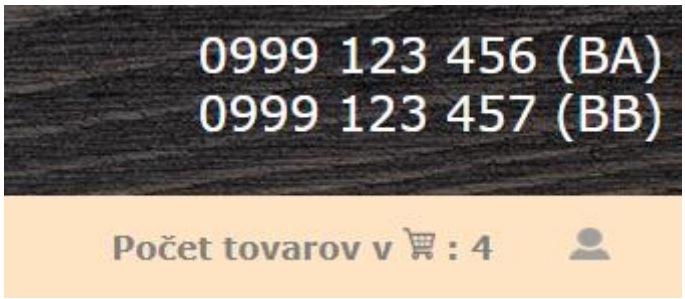
PRÍKLAD 8.4

Upravme informáciu o košíku v navigácii tak, aby sa v prípade neprázdneho košíka zobrazoval počet kusov tovaru v košíku. Využijeme funkciu **`count()`** pre `$_SESSION['kosik']`.

```
<div id="kosik_nav">
<?php
  if (isset($_SESSION['kosik'])) {
    $poc = count($_SESSION['kosik']);
    echo '<a href="kosik.php">Počet tovarov v <img
src="obrazky/kosik.png" alt="nákupný košík" width="16" height="16"
title="nákupný košík"> : ' . $poc . '</a>';
  } else {
    echo '<a href="kosik.php"><img src="obrazky/kosik.png"
alt="nákupný košík" width="16" height="16" title="nákupný košík"> je
prázdny</a>';
  }
?>
<a href=""><img src="obrazky/login.png" alt="prihlásenie" width="20"
height="16" title="prihlásenie"> </a>
...
```

V kóde sme použili funkciu **`count()`** - vráti počet prvkov poľa.

Stručné info o košíku by mohlo vyzerať ako na nasledujúcom obrázku.



ÚLOHA 8.5

Doplňte príklad 8.4 tak, že na stránku košíka pridáte tlačidlo na uloženie obsahu košíka do súboru. Tlačidlo môže byť podobné tomu na vymazanie obsahu košíka. Formát údajov v súbore si zvoľte podľa seba. Môžete sa inšpirovať príkladom 7.4.

8.1 Metodické pokyny pre učiteľa

CIEĽ

Cieľom tejto časti je oboznámiť študentov s možnosťou uchovávania údajov v SESSION.

VÝKLAD

Táto časť je zameraná na:

- vytváranie SESSION
- používanie a mazanie SESSION.

Výklad sa strieda s riešenými príkladmi a samostatnými úlohami pre študentov.

K session_start():

Častá chyba začiatočníkov je, že keď na inej stránke pridávajú niečo do SESSION, zabudnú na začiatok súboru vložiť príkaz `session_start()`.

9 ÚVOD DO JAZYKA JAVASCRIPT

JavaScript je plnohodnotný programovací jazyk, ktorý sa používa hlavne pri tvorbe webových stránok. Zaraďujeme ho po značkovacom jazyku HTML a kaskádových štýloch CSS medzi základné jazyky, ktoré je potrebné ovládať pri tvorbe webových stránok.

Prvýkrát sa tento jazyk objavil už v roku 1995 ešte pod starším označením LiveScript. Za vývojom tohto jazyka stála spoločnosť Netscape a za jeho tvorcu považujeme Brendana Eicha. Hlavným dôvodom vzniku tohto jazyka bolo optimalizovanie rýchlosti práce s webovými stránkami. Pred vznikom jazyka JavaScript sa údaje zadané používateľmi overovali na serverovej strane. Teda používateľ musel vyplniť údaje na stránke, následne musel stlačiť tlačidlo, ktoré údaje odoslalo do serverovej časti, ktorá následne vyhodnotila správnosť údajov a vrátila odpoveď klientskej časti – webovej stránke. V prípade, že používateľ zadal nesprávne údaje do formuláru, tak mu prišla odpoveď, že nevyplnil správne údaje, ktoré následne musel opraviť a znovu odoslať. Rýchlosť internetového pripojenia v tej dobe bola podstatne pomalšia ako je v súčasnosti, takže používateľ zbytočne čakal na odpoveď zo servera.

JavaScript bol v minulosti považovaný za veľmi rozporuplný jazyk. Veľká väčšina internetovej verejnosti ho považovala za nedostatočne seriózny. Hlavným dôvodom bola jeho nekonzistentnosť v prehliadačoch. Zlom nastal v roku 2004, kedy sa objavil pojem Web 2.0 a technológia Ajax sa stala dominujúcou technológiou.

POZNÁMKA

Web 2.0 – obdobie vývoja webových stránok, v ktorom sa statické stránky doplnili (resp. nahradili) dynamickými stránkami.

Jazyk JavaScript je objektovo orientovaný jazyk, ktorý obsahuje programovacie konštrukcie (napr. príkaz `if`, cyklus `while`, atď.), ktoré syntakticky pripomínajú iné programovacie jazyky ako napr. C, C++, Java. Okrem syntaxe s týmito jazykmi ich podobnosť končí.

POZNÁMKA

Jazyk JavaScript obsahuje v názve slovo Java (objektovo-orientovaný programovací jazyk), okrem názvu však s týmto programovacím jazykom nemá nič spoločné – autori sa rozhodli čerpať z veľkej popularity tohto jazyka.

9.1 Implementácia jazyka JavaScript

Jazyk JavaScript sa skladá z troch častí:

1) ECMAScript,
2) DOM (Document Object Model),
3) BOM (Browser Object Model).

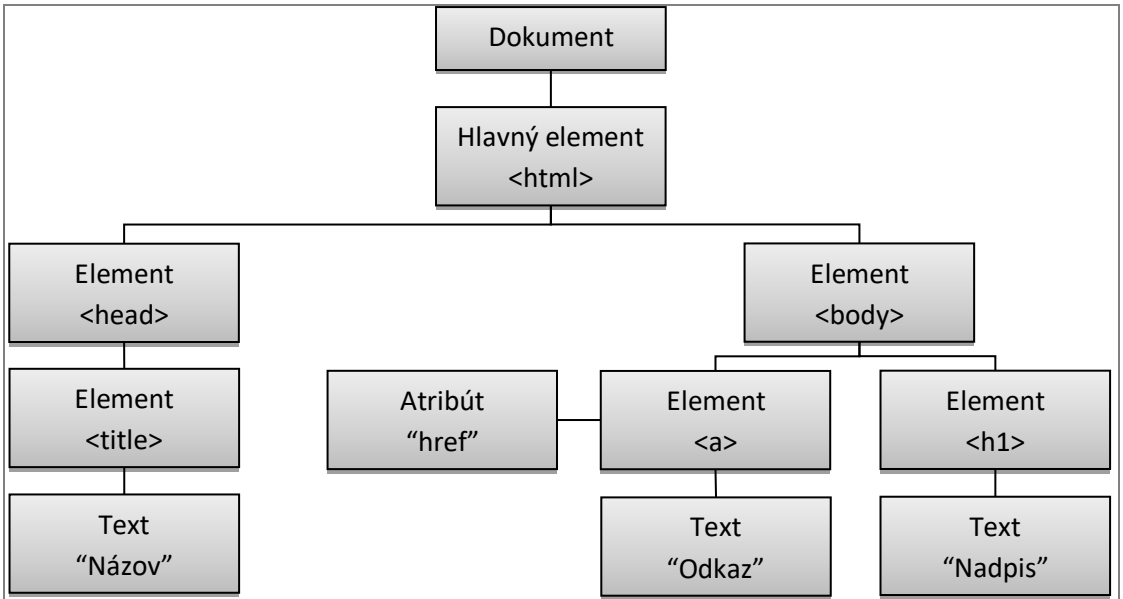
ECMAScript

Tvorí základ jazyka, na ktorom môžu stavať ďalšie jazyky. Nie je naviazaný na internetové prehliadače a teda neobsahuje žiadne metódy pre vstup a výstup. Špecifikuje:

- syntax,
- typy,
- príkazy, kľúčové slová, vyhradené slová, operátory a objekty.
- kľúčové slová, vyhradené slová, operátory a objekty.
- vyhradené slová, operátory a objekty.
- operátory,
- objekty.

Objektový model dokumentu (DOM)

DOM je API (aplikačné programovacie rozhranie), ktoré mapuje stránku do uzlov. Poskytuje nám metódy a rozhrania pre prácu s obsahom webovej stránky.



Obrázok 1 Objektový model dokumentu. Hierarchia webovej stránky.

Pomocou DOM môžeme:

- kontrolovať obsah a štruktúru stránky (uzly môžeme odstraňovať, pridávať, nahrádzať, upravovať),
- pracovať s udalosťami (event) – stlačenie klávesnice, myšky, atď.,
- aplikovať CSS štýly.

Objektový model prehliadača (BOM)

BOM umožňuje pracovať s oknom prehliadača. Môžeme napríklad zmeniť veľkosť okna prehliadača, zatvoriť prehliadač, pomocou objektu `navigator` môžeme získať informácie o prehliadači, pracovať s `cookie` prehliadača, atď.

9.1.1 Umiestnenie jazyka JavaScript do HTML

Pre použitie jazyka JavaScript v HTML stránkach musíme vložiť kód do HTML párovej značky `<script>…</script>`. Existujú dva možné spôsoby vloženia jazyka JavaScript do stránky:

POZNÁMKA

Na internete sa môžete stretnúť aj so starším označovaním JavaScriptu, ktoré používa atribúty `type="text/javascript"`- Atribút `type` je v súčasnej verzii JavaScriptu zastaralý atribút, ktorý by sme nemali používať.

Vložením JavaScript kódu priamo do stránky. Značka `<script>` môže byť vložená do časti `<body>`, alebo do časti `<head>`, alebo ju môžeme vložiť do obidvoch častí. Do webovej stránky môžeme vložiť značku `<script>` aj viackrát.

PRÍKLAD 9.1

09/01_head .html

Vytvorte HTML stránku, ktorá bude obsahovať základnú štruktúru HTML. V časti body pridajte tlačidlo s atribútom `onclick` a jeho hodnotou `mojaFunkcia()`.

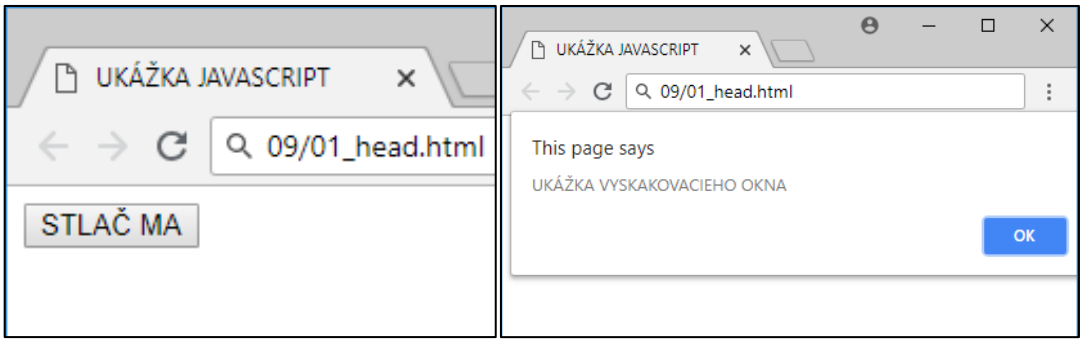
Do časti head vložte párovú značku `<script>`, ktorá bude obsahovať funkciu `mojaFunkcia()`. V tele funkcie bude zavolaná funkcia `alert()`, ktorá zobrazí dialógové okno s textom „`Ukážka vyskakovacieho okna`“.

Poznámka: Viac o JavaScript funkciách je rozpísané v kapitole 10.8.

Funkcia `alert()` vyvolá dialógové okno s textom, ktorý je poslaný ako parameter. Kľúčové slovo `function` znamená definovanie funkcie. Viac v kapitole 10.8.

```
<!DOCTYPE html>
<html>
<head>
<title>Ukážka JavaSriptu</title>
<meta charset="utf-8">
<script>
function mojaFunkcia() {
    alert("Ukážka vyskakovacieho okna");
}
</script>
</head>
<body>
  <button type="button" onclick="mojaFunkcia()">Stlač ma</button>
</body>
</html>
```

Tento kód vypíše:



Ukážka použitia jazyka JavaScript v časti `<body>`:

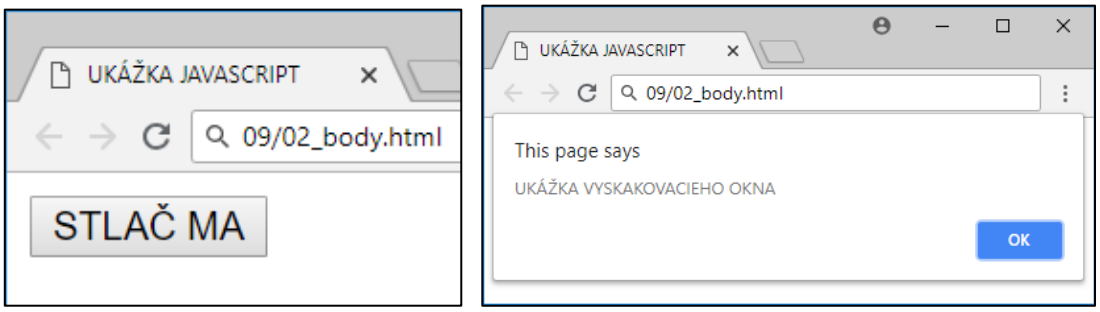
09/02_body.html

PRÍKLAD 9.2

Párovú značku `<script>` môžeme vložiť okrem hlavičky aj do časti body. Upravte predchádzajúci príklad 9.1 tak, že presuňte časť `<script>` z hlavičky do časti body.

```
<!DOCTYPE html>
<html>

<head>
<title>Ukážka JavaSriptu</title>
<meta charset="utf-8">
</head>

<body>
<script>
function mojaFunkcia() {
   alert("Ukážka vyskakovacieho okna");
}
</script>
<button type="button" onclick="mojaFunkcia()">Stlač ma</button>
```

Tento kód vypíše:



Načítavať JavaScript z externého súboru. JavaScript uložený v externom súbore oddeľuje HTML kód od jazyka JavaScript a teda zdrojový kód stránky je prehľadnejší. JavaScript v externom súbore má príponu `.js`. Pri vytváraní JavaScriptu súboru postupujeme rovnako,

ako pri vytváraní HTML alebo CSS súboru, namiesto prípony `.html` alebo `.css` ale použijeme (napíšeme) príponu `.js`. Značku `<script>` môžeme vložiť rovnako ako pri predchádzajúcom

ZAPAMÄTAJTE SI

Do súboru s príponou `.js` už nedávame párovú značku `<script>`. Obsah tohto súboru predstavuje všetko, čo sa nachádza v párovej značke `<script>`.

príklade do časti `<head>` alebo do časti `<body>`.

VYTVORTE

Predchádzajúci príklad 9.2 upravte tak, že časť `<script>` premiestnite do samostatného súboru `03_externy.js`. Do hlavičky pridajte značku `<script>` pomocou ktorej sa budete odkazovať na novovzniknutý súbor `03_externy.js`.

09/03_externy.html

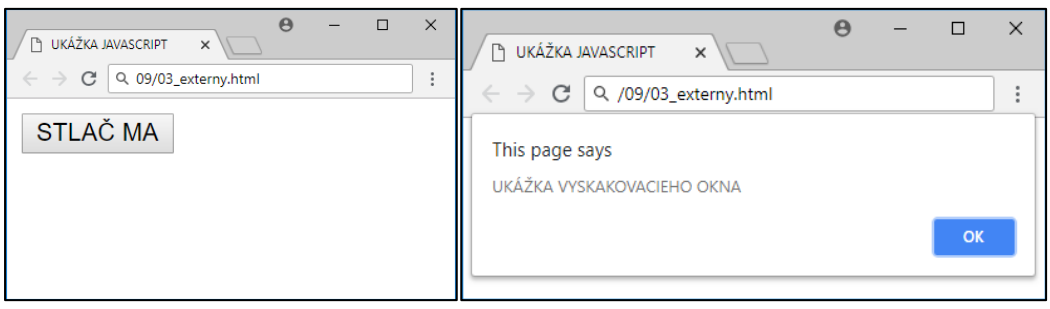
Obsah súboru `03_externy.html:`

```
<!DOCTYPE html>
<html>
<head>
<title>Ukážka JavaSriptu</title>
<script src="03_externy.js"></script>
</head>

<body>

<button type="button" onclick="mojaFunkcia()">Stlač ma</button>

</body>
</html>
```

Obsah súboru `03_externy.js`

```
function mojaFunkcia() {
    alert("Ukážka vyskakovacieho okna");
}
```

Tento kód vypíše:



Atribúty značky <script>

Do párovej značky `<script>` môžeme vložiť rovnako, ako pri iných HTML značkách rôzne atribúty.

Tabuľka 2 Atribúty značky <script>

Atribúty	Hodnota	Popis
async	*async*	*Script sa vykonáva asynchrónne. Je možné ho použiť iba pre skripty v externom súbore.*
defer	*defer*	*Načítanie obsahu odloží dovtedy, kým sa neskončí analýza. Je možné ho použiť iba pre skripty v externom súbore.*
charset	*"UTF-8"*	*Znaková sada použitá v externom štýly (ak používame diakritiku napr. v komentároch).*
src	*www.mojastranka.sk/javascript.js*	*Určuje umiestnenie súboru*

Značka <noscript>

Vzhľadom nato, že niektoré prehliadače majú možnosť zakázať používanie jazyka JavaScript bola vytvorená značka `<noscript>`, ktorá sa uplatní vtedy, keď JavaScript nie je v prehliadači povolený, alebo ho prehliadač nepodporuje. Do tejto značky môžeme napísať ľubovoľný text, ktorý sa zobrazí používateľovi.

V ukážke nižšie je pridaná párová značka `<noscript>…</noscript>`, ktorá sa uplatní iba vtedy, keď prehliadač nepodporuje, alebo má zakázané spúšťanie jazyka JavaScript.

09/04_noscript.html

PRÍKLAD 9.4

Do predchádzajúceho príkladu 9.3 doplňte párovú značku `<noscript>` s textom `"Tvoj prehliadač nepodporuje JavaScript!!!"`, ktorý sa zobrazí v prípade, ak prehliadač nepodporuje jazyk JavaScript.

```
<!DOCTYPE html>
<html>

<head>
<title>Ukážka JavaScriptu</title>
<meta charset="utf-8">
</head>

<body>
<script>
function mojaFunkcia(){
      alert("Ukážka vyskakovacieho okna");
}
</script>
<noscript>Tvoj prehliadač nepodporuje JavaScript!!!</noscript>

<button type="button" onclick="mojaFunkcia()">Stlač ma</button>
</body>
</html>
```

Značka `<noscript>`, sa môže používať v časti `<head>`, aj v `<body>`.

9.2 Metodika pre učiteľa

CIEĽ

Cieľom je ukázať možnosti uplatnenia JavaScriptu pri vytváraní dynamického obsahu webových stránok. Študenti sa naučia rôzne spôsoby vkladania JavaScript zdrojového kódu do html súboru webovej stránky a vytváranie samostatných `.js` súborov.

MOTIVÁCIA

Študent dokáže vytvoriť jednoduchú webovú stránku s vloženým JavaScript zdrojovým kódom a následne ju spustiť a zobraziť výsledok pomocou webového prehliadača.

VÝKLAD

Študentov treba oboznámiť:

- ako vyzerá samotný JavaScript zdrojový kód v html súbore,
- s implementáciou jazyka JavaScript (ECMAScript, DOM, BOM),
- akým spôsobom a kde sa vykonáva JavaScript,
- ako vložiť JavaScript kód do html súboru,
- ako vytvoriť externý súbor so zdrojovým kódom JavaScript a ako ho použiť.

Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch.

10 JAVASCRIPT – ZÁKLADNÁ SYNTAX A ŠTRUKTÚRY

V tejto kapitole budeme pre jednoduchosť uvádzať len časti zdrojového kódu zapísané medzi značkami `<script> .. </script>`.

- Identifikátory
- Premenné
- Kľúčové a vyhradené slová
- Komentáre
- Dátové typy
- Operátory
- Riadiace príkazy

10.1 Identifikátory

Identifikátor, alebo inak povedané názov, používa sa k pomenovaniu premenných, funkcií, vlastnosti, argumentov funkcie, atď. Pri vytváraní identifikátorov musíme dodržiavať nasledujúce pravidlá:

- Prvý znak musí byť písmeno, podčiarkovník (`_`) alebo znak dolára (`$`). Ostatné znaky už môžu byť čísla, písmená, znak dolára, podčiarkovník, atď.

```
prvaPremenna
druhaPremenna1
_tretiaPremenna
$stvrtaPremenna
```

Identifikátor v sebe nesmie obsahovať medzeru. Pri viacslovných identifikátoroch by sme mali začať každé slovo s veľkým písmenom (Camel case) alebo použiť namiesto medzery podčiarkovník.

```
prvaPremenna
druha_Premenna1
```

JavaScript rozlišuje veľké a malé písmená. Premenná s identifikátorom `informatika` je iná premenná ako premenná s identifikátorom `Informatika`, a teda sa bude jednať o dve rôzne premenné.

10.2 Premenné

Premenné v jazyku JavaScript sú tzv. "voľne typované", to znamená, že môžu uchovávať rôzne typy dát. Premenná sa definuje pomocou kľúčového slova `var`, za ktorým nasleduje názov premennej (identifikátor).

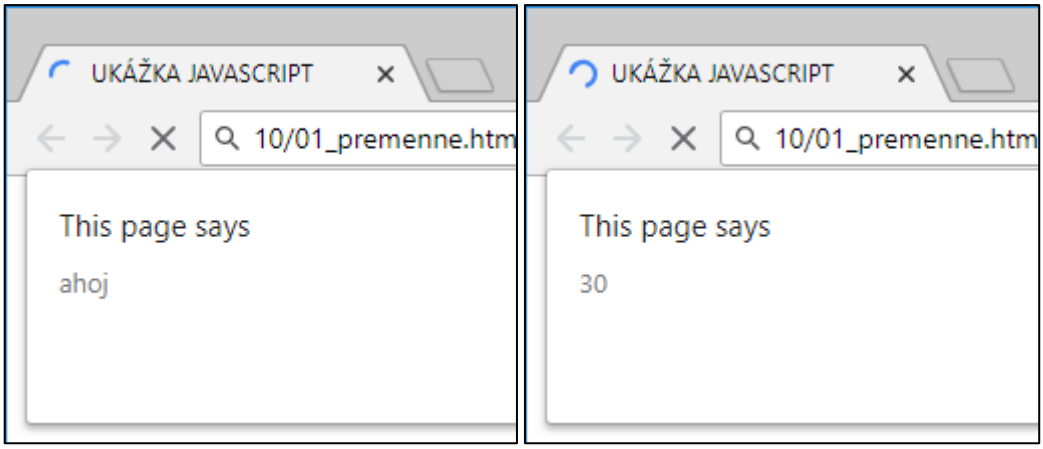
```
var nazovPremennej;
var nazovPremennej2 = hodnota_ktoru_uchovava_premenna;
nazovPremennej2 = nova_hodnota_ktoru_uchovava_premenna;
```

10/01_premennne.html

PRÍKLAD 10.1

Zadefinujeme niekoľko premenných s rôznymi hodnotami. Každú premennú zobrazíme v dialógovom okne pomocou hodnoty `alert()`.

```
//definícia premennej
var suma;

//definícia premennej s inicializáciou jej hodnoty
var sprava = "ahoj";

//zobrazenie hodnoty, ktorú uchováva premenná sprava
alert(sprava);

//nastavenie novej hodnoty premennej
sprava = 30;

//zobrazenie hodnoty, ktorú uchováva premenná sprava
alert(sprava);

//definícia viacerých ďalších premenných
var text, cislo;
```



POZNÁMKA

Premenná sprava najprv uchováva text `ahoj`, následne sa hodnota prepíše na číselnú hodnotu `30`. Avšak takéto riešenie nie je vhodné a teda sa neodporúča takto prepisovať hodnoty rôzneho údajového typu (`String` na `int`).

Ak chceme definovať viac než jednu premennú, stačí napísať raz operátor `var` a následne všetky ďalšie premenné oddelíme čiarkou:

```
var suma, sprava, text;
```

Premenná je vytvorená lokálne v rámci oboru platnosti, kde je definovaná. Napríklad, ak je premenná definovaná vo vnútri nejakej funkcie pomocou operátora `var`, táto premenná je odstránená okamžite, keď sa funkcia ukončí.

PRÍKLAD 10.2

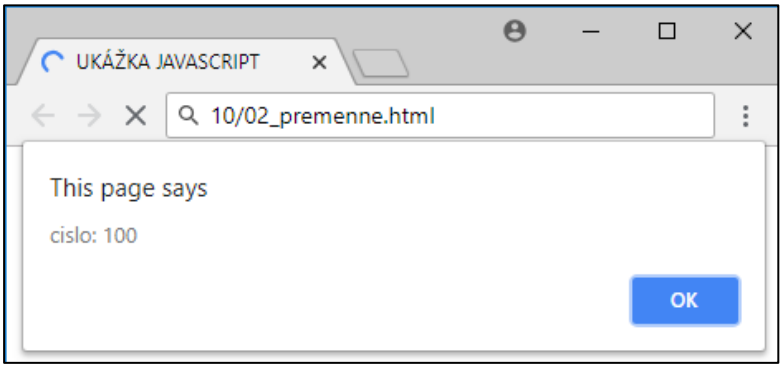
10/02_premenne.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú s názvom `cislo`. Ďalej vytvorte funkciu s názvom `nastavPremennu()`, v ktorej priraďte premennej `cislo` číselnú hodnotu `100`. Po zadefinovaní funkcie túto funkciu zavolajte. Posledný príkaz, ktorý vložte do časti `<script>` je zavolanie funkcie na zobrazenie dialógového okna `alert()` s parametrom `cislo`.

Poznámka: Takto vytvorená premenná `cislo` sa nazýva aj globálna premenná.

```
var cislo; //globálna premenná
function nastavPremennu() {
      cislo = 100;
}
nastavPremennu();
alert(cislo);
```

Tento kód zobrazí stránku:



ÚLOHA 10.1

Upravte príklad 10.2 tak, že premennú s názvom `cislo` zadefinujte priamo vo funkcii `nastavPremennu()`. Stránku spustite a pozorujte, či sa zobrazí dialógové okno.

Poznámka: Takto vytvorená premenná `cislo` sa nazýva lokálna premenná.

10.3 Kľúčové a vyhradené slová

Pri vytváraní premenných nemôžeme nazvať premenné identifikátormi, ktoré sú vyhradené pre špecifické operácie. Takýmto premenným hovoríme kľúčové, resp. vyhradené slová. V tabuľke 2 sa nachádzajú niektoré najčastejšie používané slová, ktoré nemôžeme použiť ako identifikátory. V prípade, že použijeme niektoré z vyhradených slov, napríklad ako názov premennej, môže sa vyvolať chyba "očakávaný identifikátor". To, či sa chyba objaví záleží od konkrétneho prehliadača.

Tabuľka 3 Zoznam niektorých najčastejšie používaných vyhradených slov (https://www.w3schools.com/Js/js_reserved.asp).

in	arguments	break	let
instanceof	interface	super	function
int	boolean	char	double
null	long	float	true
false	break	case	catch
throws	new	class	const
package	private	protected	static
continue	default	do	while
for	this	switch	enum
void	import	with	if

10.4 Komentáre

V jazyku JavaScript je možné používať komentáre. Spravidla sa používajú na zdokumentovanie funkcionalít jednotlivých častí zdrojového kódu, prípadne na dočasné znefunkčnenie častí zdrojového kódu počas vývoja alebo ladenia. JavaScript umožňuje používať dva typy komentárov:

1) **jednoriadkový komentár** – text, ktorý sa nachádza medzi `//` a koncom riadku, bude jazyk JavaScript ignorovať,

```
//toto je jednoriadkový komentár
```

2) **viacriadkový komentár** – text, ktorý sa nachádza medzi `/*` a `*/`, bude jazyk JavaScript ignorovať. Medzi značkami `/*` a `*/` môže byť ľubovoľný počet riadkov.

```
/*
 Toto
 je
 viacriadkový
 komentár, hviezdičky nemusíme vypisovať
 na každom riadku, stačí na začiatku a
 na konci
*/
```

POZNÁMKA

Najčastejšie používaný je tzv. jednoriadkový komentár. Viacriadkový komentár sa väčšinou používa na dokumentáciu.

PRÍKLAD 10.3

10/03_komentare.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú s názvom `cislo`. O riadok vyššie vložte jednoriadkový komentár s textom.

```
//toto je jednoriadkovy komentar
```

Za premennú (bez toho, aby ste stlačili klávesu Enter – posunuli kurzor na nový riadok) vložte ďalší komentár .

```
//aj takto moze vyzerat jednoriadkovy komentar
```

O dva riadky nižšie vložte viacriadkový komentár s textom.

```
/*viac riadkovy
komentar moze
 vyzerat takto */
```

```
//toto je jednoriadkovy komentar
  var cislo; //aj takto moze vyzerat jednoriadkovy komentar

/* viac riadkovy
   komentar moze
   vyzerat takto */
```

ZAPAMÄTAJTE SI

Jednoriadkový aj viacriadkový komentár môžeme vložiť iba do značky `<script>`, alebo do súboru `.js`. Ak chceme použiť komentáre v jazyku HTML musíme použiť komentáre s HTML syntaxou.

10.5 Dátové typy

V programovaní je dôležité uchovávať hodnoty rôznych dátových typov. JavaScript na rozdiel od iných programovacích jazykoch používa tzv. dynamické dátové typy. Každá premenná môže v sebe uchovávať rôzne dátové typy. V jazyku JavaScript existuje 5 jednoduchých dátových typov:

- Number (číslo),
- String (reťazec),

- Undefined (nedefinovaná hodnota),
- Null,
- Boolean,

10.5.1 Number

Dátový typ `Number` sa používa na reprezentáciu číselných hodnôt. Môže obsahovať celé alebo reálne čísla, pričom túto informáciu nemusíme uviesť pri inicializovaní premennej. Maximálny počet desatinných miest je 17.

10/04_number.html

PRÍKLAD 10.4

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

- `cislo` s hodnotou `5`,
- `cislo2` s hodnotou `3.14`.

Do parametra funkcie `alert()` vložte najskôr text `Celé číslo:` a premennú `cislo` a potom text `Reálne číslo` a premennú `cislo2`.

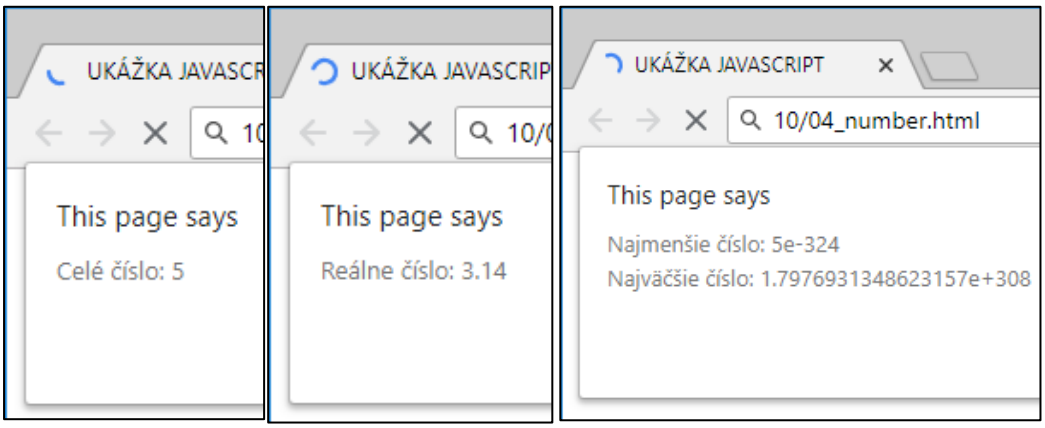
Posledný príkaz, ktorý vložte do značky `<script>` je `alert()` s hodnotou parametra `Najmenšie číslo: Number.MIN_VALUE + Najväčšie číslo: Number.MAX_VALUE.`

Poznámka: Pomocou operatora `+` je možné spojiť viacero hodnôt do jedného reťazca, ktorý sa následne použije ako parameter funkcie `alert()`.

Poznámka: Pri zadávaní reálnych čísel sa na oddelenie desatinných miest používa bodka.

```
//definícia premennej s celým číslom
var cislo = 5;
alert("Celé číslo: " + cislo);

//definícia premennej s reálnym číslom
var cislo2 = 3.14;
alert("Reálne číslo: " + cislo2);

//konštanty pre najmenšie a najväčšie číslo
alert("Najmenšie číslo: " + Number.MIN_VALUE + "\nNajväčšie číslo: " +
Number.MAX_VALUE);
```

Tento kód zobrazí nasledujúce dialógové okná:



Kvôli pamäťovým obmedzeniam nie je možné reprezentovať všetky čísla. Najmenšie kladné číslo, ktoré je možné uložiť je uložené v konštante `Number.MIN_VALUE`, najväčšie kladné číslo je uložené v konštante `Number.MAX_VALUE`.

- **Najmenšie číslo =** 5e-324
- **Najväčšie číslo =** 1.7976931348623157e+308

Čísla menšie ako `Number.MIN_VALUE` sú konvertované na hodnotu `0`. Čísla väčšie ako `Number.MAX_VALUE` sú reprezentované ako `Infinity` (nekonečno). V prípade, že nám výpočet vráti hodnotu `Infinity`, nemôžeme túto hodnotu použiť pri ďalšom výpočte. Na zistenie, či je nejaká hodnota konečná, sa používa funkcia `isFinite()`. Funkcia vracia boolean hodnotu (`true/false`).

POZNÁMKA

Dátový typ Boolean je vysvetlený v kapitole 10.5.5.

PRÍKLAD 10.5

10/05_infinity.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

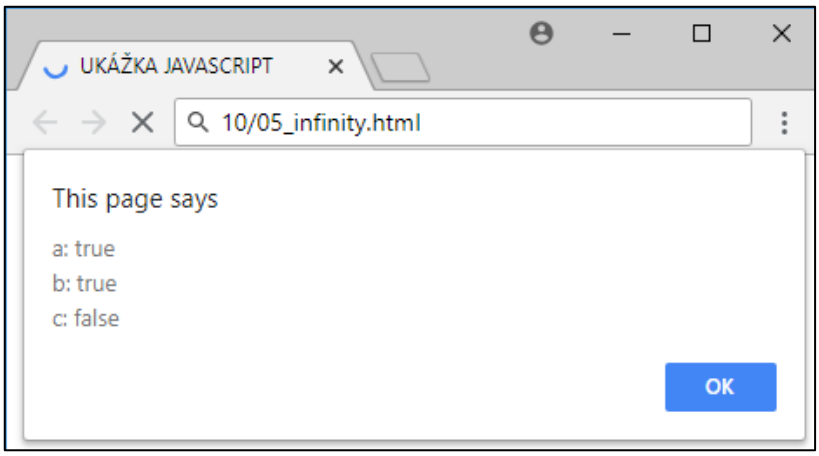
- `maxCislo` *s hodnotou* `Number.MAX_VALUE`,
- `a` s boolean hodnotou, ktorá sa vypočíta podľa toho, či číslo `123` je konečné,
- `b` s hodnotou, ktorá sa vypočíta podľa toho, či číslo `maxCislo` je konečné,
- `c` s hodnotou, ktorá sa vypočíta podľa toho, či číslo `maxCislo` je konečné. Pred vytvorením premennej `c` však zmeňte hodnotu premennej `maxCislo` na hodnotu `maxCislo * maxCislo`.

Pomocou funkcie `alert()` zobrazte v dialógovom okne hodnoty premenných `a,b,c`.

Poznámka: Na vypísanie textu do nového riadku použite znaky `\n`.

```
var maxCislo = Number.MAX_VALUE;

 //použitie funkcie isFinite()
var a = isFinite(123);

var b = isFinite(maxCislo);

maxCislo = maxCislo * maxCislo;

var c = isFinite(maxCislo);

alert("a: " + a + "\nb: " + b + "\n c: " + c);
```

Tento kód zobrazí nasledujúce dialógové okná:



Pri operáciách s číselnými dátovými typmi môže nastať situácia, keď sa namiesto čísla zobrazí hodnota `NaN`. Táto hodnota reprezentuje nečíselnú hodnotu (`"Not a number"`).

POZNÁMKA

V niektorých iných programovacích jazykoch by sa vyvolala chyba.

Ak chceme zistiť, či je nejaká hodnota číselného typu, môžeme nato použiť funkciu `isNaN()`. V prípade, že použijeme ako parameter dátovú hodnotu `String` – funkcia sa najprv pokúsi previesť parameter na číslo (napr. reťazec `"44"`, alebo Boolean hodnoty `true/false` sa dajú previesť na číselnú hodnotu). Ak sa nedá parameter previesť na číslo, tak funkcia vráti `true`.

10/6_nan.html

PRÍKLAD 10.6

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

- `a` s hodnotou `5`,
- `b` s hodnotou `3`,
- `c` s hodnotou `Ahoj`,
- `d` zatiaľ bez hodnoty.

Po vytvorení premenných, priraďte do premennej `d` hodnotu `a/b` a premennú `d` vložte do parametra funkcie `alert()`.

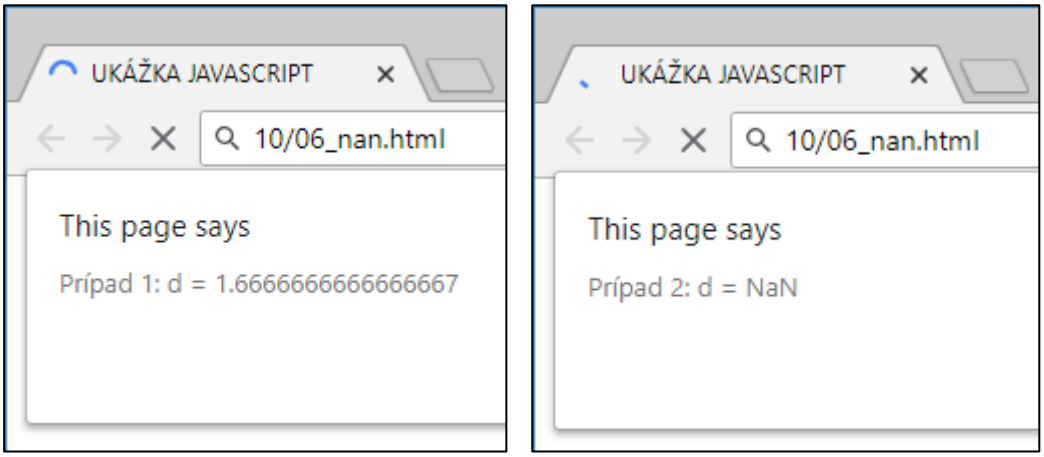
V premennej `d` zmeňte hodnotu na hodnotu `a/c` a premennú `d` s novom hodnotou vložke ako parameter funkcie `alert()`.V ďalšej časti značky `<script>` si vyskúšajte zavolanie funkcie `isNan()`, kde postupne ako parameter vložte:

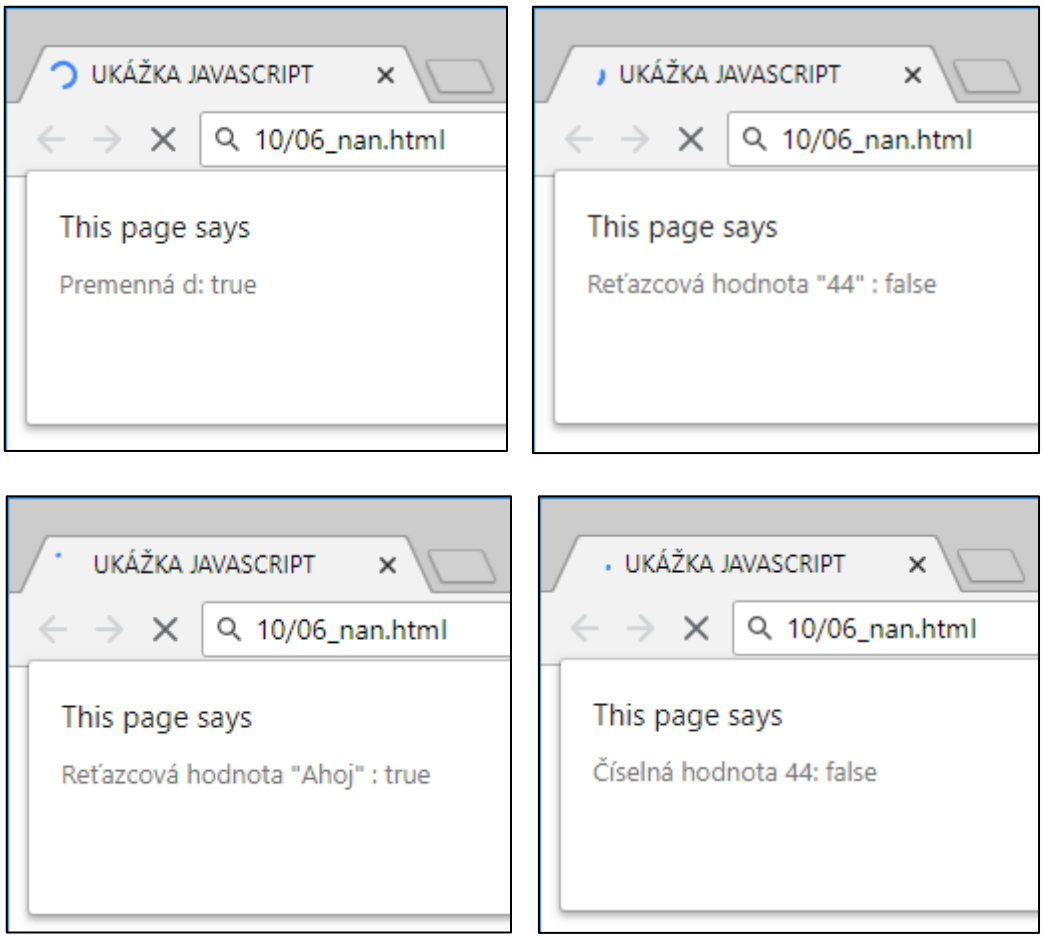
- premennú `d`,
- reťazcovú hodnotu „`44`“ ,
- reťazcovú hodnotu „`Ahoj`“ ,
- číselnú hodnotu `44`.

Poznámka: viac o reťazcových hodnotách sa dozviete v kapitole 10.5.2.

```
//definícia premenných
var a = 5, b = 3;
var c = "Ahoj";
var d;

//výsledok delenia bude číslo
d = a / b;
alert("Prípad 1: d = " + d);

//výsledok delenia nebude číslo
d = a / c;
alert("Prípad 2: d = " + d);

//ukážka použitia funkcie isNaN()
alert("Premenná d: " + isNaN(d));
alert("Reťazcová hodnota \"44\" : " + isNaN("44"));
alert("Reťazcová hodnota \"Ahoj\" : " + isNaN("Ahoj"));
alert("Číselná hodnota 44:" + isNaN(44));
```

Tento kód zobrazí nasledujúce dialógové okná:





Prevod čísiel

JavaScript používa tri funkcie na prevod nečíselných hodnôt na číselné hodnoty:

1) funkciu `Number()`

Pri pretypovaní ľubovoľného dátového typu na číselnú hodnotu:

- boolean hodnota `true = 1, false = 0`,
- dátového typu `null = 0`,
- `undefined = NaN`,
- reťazec `"44" = 44,"044" = 44, " " = 0, "text" = NaN`.

10/07_number.html

PRÍKLAD 10.7

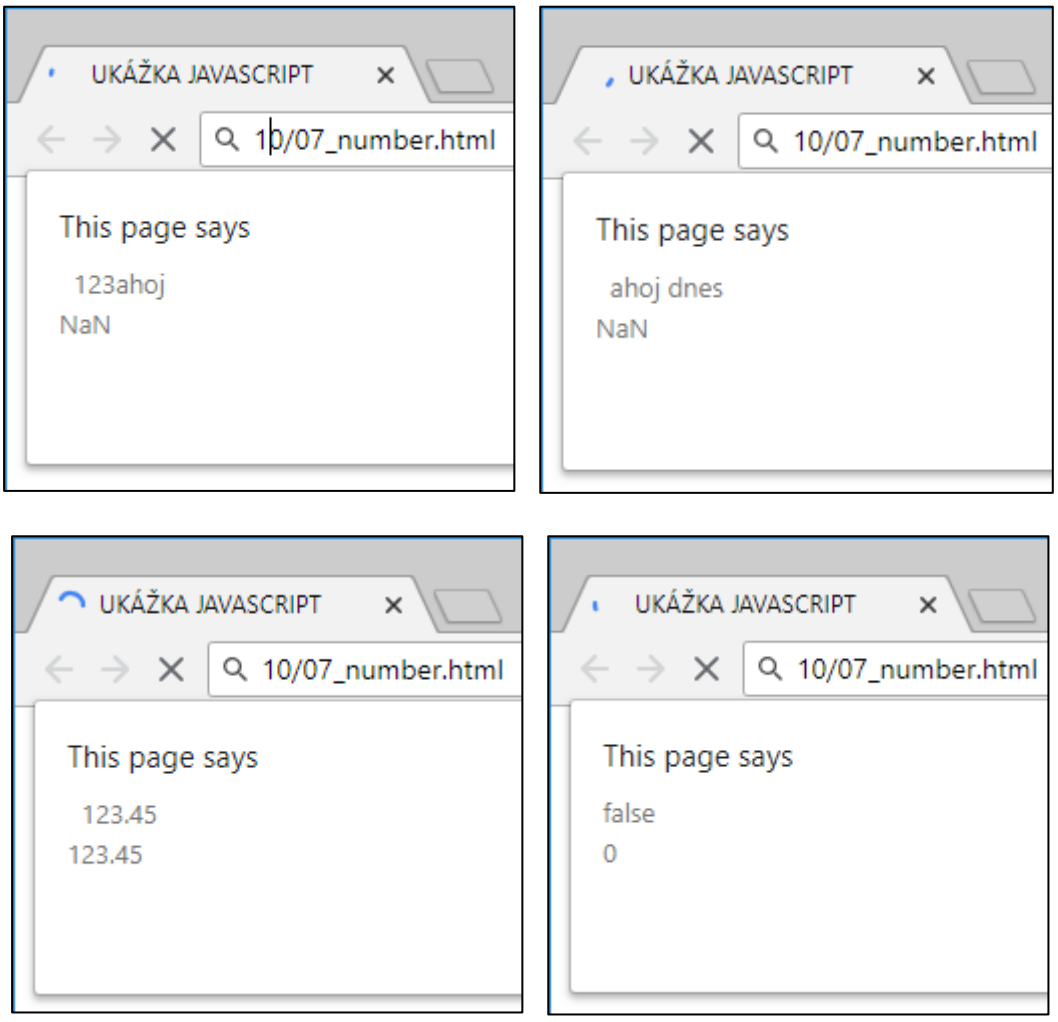
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

- `a` s reťazcovou hodnotou `"  123ahoj"`,
- `b` s reťazcovou hodnotou `" ahoj dnes"`,
- `c` s číselnou hodnotou `123.45`,
- `d` s boolean hodnotou `false`.

Pomocou funkcie `alert()` postupne zobrazte v dialógovom okne hodnotu premennej a na novom riadku novú číselnú hodnotu premennej prevedenú pomocou funkcie `Number()`.

```
var a = "  123ahoj";
var b = "  ahoj dnes";
var c = "  123.45";
var d = false;

alert(a + "\n" + Number(a));
alert(b + "\n" + Number(b));
alert(c + "\n" + Number(c));
alert(d + "\n" + Number(d));
```

Tento kód zobrazí nasledujúce dialógové okná:



2) funkcia `parseInt()`

Prevod reťazcov na celé čísla, pričom ignoruje biele znaky

- boolean hodnota `true = NaN, false = NaN`
- `undefined = NaN`
- reťazec `"44" = 44, "44.55"  = 44, " " = 0, "text" = NaN, "12december" = 12`

ÚLOHA 10.2

Modifikujte súbor *10/07_number.html* a použite namiesto funkcie `Number()` funkciu `parseInt()`. Pozorujte ako sa zmenia výsledky oproti výsledkom z príkladu *07_number.html*.

3) funkcia `parseFloat()`

Prevod reťazcov na reálne čísla, pričom do úvahy berie aj prvú bodku (desatinnú čiarku).

- boolean hodnota `true = NaN, false = NaN`
- `undefined = NaN`
- reťazec `"44" = 44, "44.55" = 44.55, " " = 0, "text" = NaN, "12december" = 12`

ÚLOHA 10.3

Modifikujte súbor *10/07_number.html* a použite namiesto funkcie `Number()` funkciu `parseFloat()`. Pozorujte ako sa zmenia výsledky oproti výsledkom z predchádzajúcich príkladov.

10.5.2 String

String predstavuje textový reťazec, ktorý ohraničujeme v jazyku JavaScript pomocou úvodzoviek alebo apostrofov. Textový reťazec predstavuje množinu za sebou nasledujúcich ľubovoľných znakov (písmená, číslice, interpunkčné znamienka) prípadne môže byť aj prázdny.

```
var text = "Tento text sa zobrazi v alert dialógu";
```

Textový reťazec musí byť zadaný na jednom riadku, nie je možné ho rozdeliť na viac riadkov. V prípade potreby dlhšieho textu je možné spojiť viacero reťazcov pomocou operátora `+`.

```
var text = "ahoj" + " javascript";
```

10/08_string.html

PRÍKLAD 10.8

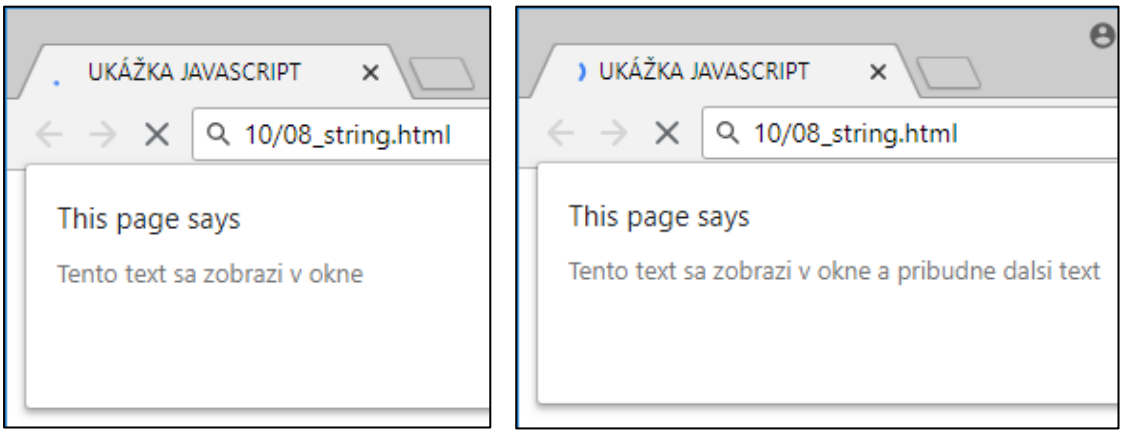
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `text` s reťazcovou hodnotou `Tento text sa zobrazí v okne`.

Pomocou funkcie `alert()` najprv vypíšte hodnotu premennej `text`. Po vypísaní hodnoty v dialógovom okne zmeňte hodnotu premennej `text` na novú hodnotu, ktorá bude obsahovať pôvodnú hodnotu premennej plus ďalší text: `a pribudne dalsi text.`

```
//definícia premnnej
var text = "Tento text sa zobrazi v okne";

//zobrazenie hodnoty premennej
alert(text);

//pripojenie d'alšieho textu k obsahu premennej
text = text + " a pribudne dalsi text";

//zobrazenie hodnoty premennej
alert(text);
```

Tento kód zobrazí nasledujúce dialógové okná:



POZNÁMKA

Prázdny reťazec sa používa na zmazanie obsahu reťazcovej premennej `text = ""`.

Prevod na reťazce

Podobne ako pri číselných hodnotách, aj pri prevádzaní iných hodnôt na reťazce môžeme použiť viacero funkcií:

1) Funkcia `toString()`
   - dostupná pre čísla, Boolean hodnoty, reťazce, objekty,
   - v prípade hodnoty `null` alebo `undefined` sa nemôže použiť .

PRÍKLAD 10.9

10/09_string.html

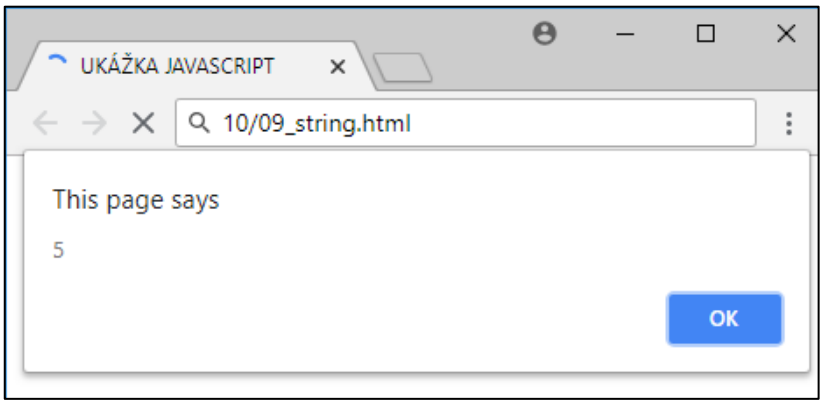
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `znamka` s číselnou hodnotou `5`,
- `znamkaString` zatiaľ bez hodnoty.

Do premennej `znamkaString` uložte hodnotu premennej `znamka` prevedenú na hodnotu `String` pomocou funkcie `toString()`. Premennú `znamkaString` vypíšte v dialógovom okne `alert()`.

```
var znamka = 5;
var znamkaString;

//príklad použitia funkcie toString()
znamkaString = znamka.toString();

//príklad použitia pretypovania
znamkaString = String(znamka);
```

Tento kód zobrazí nasledujúce dialógové okno:



2) Pretypovanie `String()`
   - vráti reťazec bez ohľadu na typ.

ÚLOHA 10.4

Upravte príklad 10.3 (súbor *10/08_string.html*) tak, aby ste namiesto funkcie `toString()` použili funkciu `String()`.

10.5.3 Undefined

Dátový typ `Undefined` sa používa, keď je premenná definovaná bez inicializácie. Môže obsahovať iba jednu jedinú hodnotu = `undefined`.

10.5.4 Null

Typ `null` je prázdna hodnota. Podobne ako `undefined` môže obsahovať iba jednu hodnotu - `null`. Používame ju, ak chceme zistiť, či je premenná naplnená odkazom na objekt:

```
if (objekt == null) {
      ...
}
```

10.5.5 Boolean

Pri vyhodnocovaní logických výrazov je výsledkom hodnota dátového typu `boolean`, ktorý môže nadobúdať iba dva stavy:

- true – pravda
- false – nepravda

```
var x = true;
var y = false;
```

Patrí medzi často používané dátové typy pri programovaní. Používame ho napríklad v podmienkach:

```
if (x == 5) {
      // príkazy sa vykonajú, ak bol výraz vyhodnotený ako true
} else {
      // príkazy sa vykonajú, ak bol výraz vyhodnotený ako false
}
```

PRÍKLAD 10.10

10/10_boolean.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

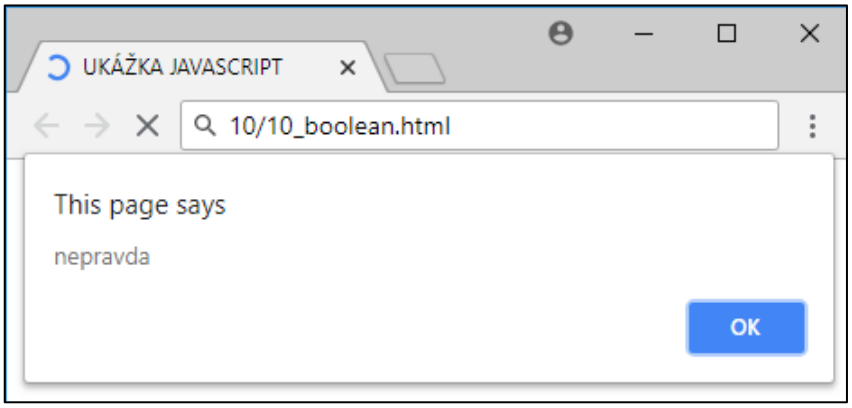
- `x` s logickou premennou `true`,
- `y` s logickou premennou `false`,
- `a` s hodnotou `4`.

Pod definovaním premenných vložte podmienku: ak premenná `a` sa rovná číselnej hodnote `5`, tak zobraz dialógové okno s textom `pravda`. Ak sa nerovná, tak zobraz dialógové okno s textom `nepravda`.

Poznámka: Viac o podmienkach je napísané v kapitole 10.7.1.

```
//definícia logickej premennej
var x = true;
var y = false;

//výsledom výrazu v zátvorke je logická hodnota
var a = 4;
if (a == 5) {
   alert("pravda");
} else {
  alert("nepravda");
}
```

Tento kód zobrazí nasledujúce dialógové okno:



ÚLOHA 10.5

Upravte príklad 10.10 (súbor *10/10_boolean.html*) tak, aby bola splnená podmienka a zobrazil sa text `pravda`.

POZNÁMKA

Upozornenie logická hodnota `true` sa nerovná `True`!!!

10.6 Operátory

S hodnotami uloženými v premenných ako aj s hodnotami v podobe konštánt vieme vykonávať rôzne operácie. Pri číselných hodnotách ide najmä o bežne používané aritmetické operácie. Okrem toho vieme tieto číselné ako aj reťazcové hodnoty navzájom porovnávať, prípadne pomocou logických operátorov skonštruovať zložené podmienky. JavaScript podporuje širokú škálu operátorov:

- aritmetické operátory,
- relačné operátory,

- logické operátory,
- porovnávacie operátory.

POZNÁMKA

V prípade, že niektorí z operandov nie je číslo, tak sa automaticky prevedie na číslo pomocou funkcie `Number()`. Napr. prázdny reťazec je prevedený na `0`, boolean hodnota `false` je prevedená na hodnotu `0, true` na hodnotu `1`, atď.

10.6.1 Aritmetické operátory

Aritmetické operátory sa používajú v matematických výrazoch. Medzi aritmetické operácie zaraďujeme operácie sčitovania (`+`), odčitovania (`-`), násobenia (`*`), delenia (`/`), zvyšok po delení (`%`), increment (`++`) a decrement (`--`).

PRÍKLAD 10.11

10/11_aritmetika.php

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

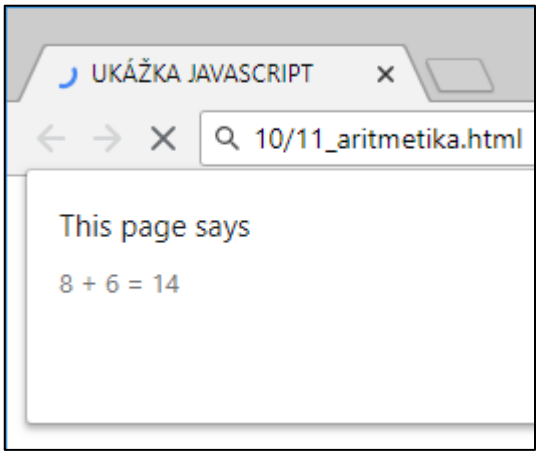
- `premenna1` s číselnou hodnotou `8`,
- `premenna2` s číselnou hodnotou `6`,
- `vysledok` zatiaľ bez hodnoty.

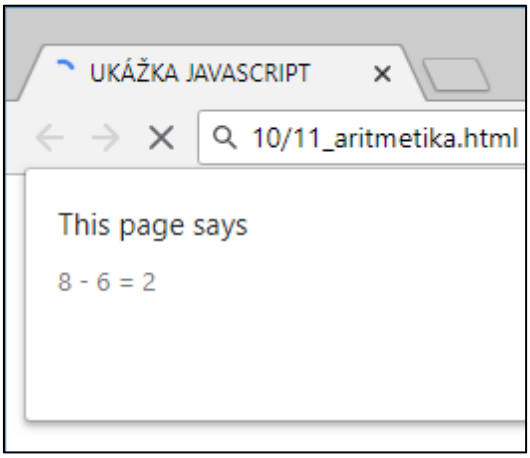
Do premennej `vysledok` postupne uložte hodnoty a súčasne zobrazte výsledok v dialógovom okne:

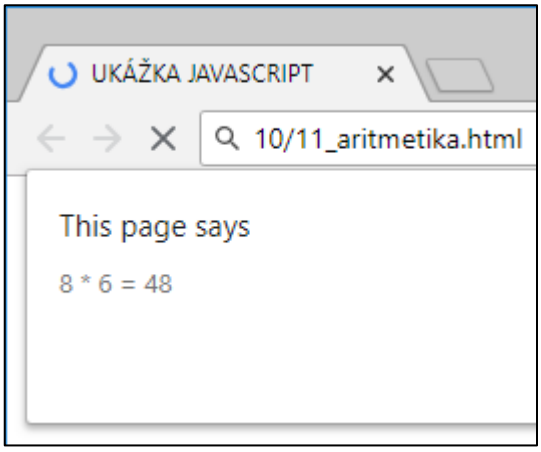
- sčítanie hodnôt `premenna1` a `premenna2`,
- odčítanie hodnoty `premenna2` od hodnoty premennej `premenna1`,
- vynásobenie hodnôt `premenna1` a `premenna2`,
- vydelenie hodnoty `premenna1` hodnotou premennej `premenna2`,

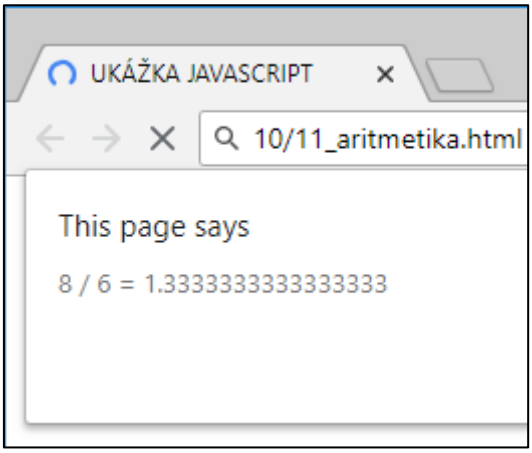
zvyšok po delení hodnôt `premenna1` a `premenna2`.

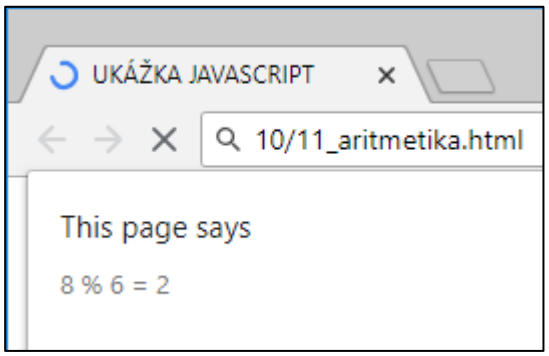
```
var premenna1 = 8;
var premenna2 = 6;

//sčítavanie
var vysledok = premenna1 + premenna2;
alert(vysledok); //vysledok = 14

//odčítanie
vysledok = premenna1 – premenna2;
alert(vysledok); //vysledok = 2

//násobenie
vysledok = premenna1 * premenna2;
alert(vysledok); //vysledok = 48

//delenie
vysledok = premenna1 / premenna2;
alert(vysledok); //vysledok = 1.1428571428571428.

//zvyšok po delení
vysledok = premenna1 % premenna2;
alert(vysledok); //vysledok = 2
```











ÚLOHA 10.6

Upravte predchádzajúci príklad 10.11 (súbor *10/11_aritmentika.html*) tak, že vymeníte hodnoty premenných `premenna1` a `premenna2`.

- `premenna1 = 6`
- `premenna2 = 8`

Pozorujte, ako sa zmenili výsledky.

Operácie inkrement a dekrement pripočítavajú/odpočítavajú k číselnej premennej hodnotu `1`:

- **Inkrement** – pripočítavanie 1,
- **Dekrement** – odpočítavanie 1.

POZNÁMKA

Inkrementácia a dekrementácia nastane až, keď sa vyhodnotí celý riadok.

Tieto operácie ešte delíme podľa umiestnenia pred alebo za premennú na:

- **Prefixové** – pred premennou. Hodnota v premennej `SUMA` je najprv zväčšená/zmenšená o `1` a následne použitá pri vyhodnotení celého riadku.

```
var suma = 20;
var suma2 = --suma + 5;
alert(suma);      // 19
alert(suma2);     // 24
```

- **Postfixkové** – za premennou. Pôvodná hodnota v premennej `SUMA` je použitá pri vyhodnotení celého riadku a až potom je zväčšená/zmenšená o `1`.

10/12_inkrement.html

PRÍKLAD 10.12

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

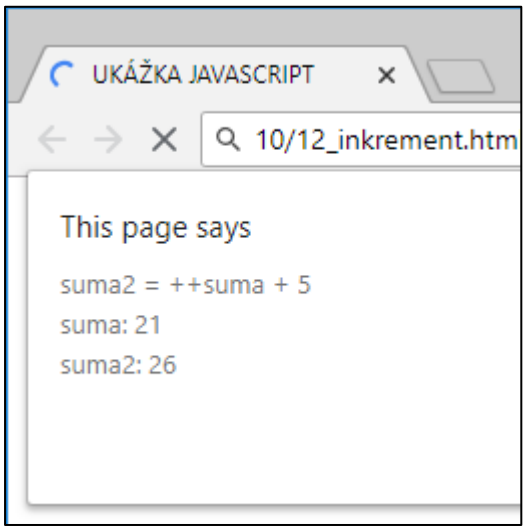
- `suma` zatiaľ bez hodnoty,
- `suma2` zatiaľ bez hodnoty.

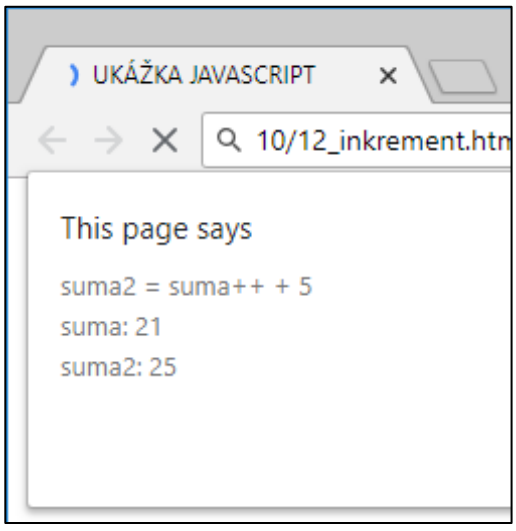
Postupne upravujte hodnoty vytvorených premien:

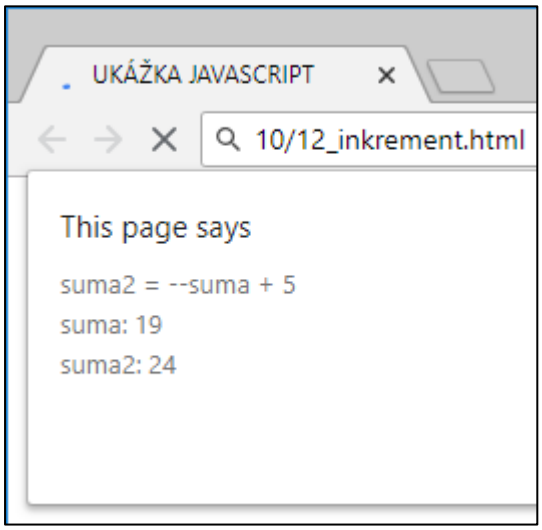
- premennej `suma` prideľte hodnotu `20`,
- premennej `suma2` prideľte hodnotu *prefixový inkrement* premennej `suma + hodnota 5`,
- vypíšte hodnoty premenných `suma` a `suma2`,
- premennej `suma` prideľte hodnotu `20`,
- premennej `suma2` prideľte hodnotu *postfixový inkrement* premennej `suma + hodnota 5`,
- vypíšte hodnoty premenných `suma` a `suma2`,
- premennej `suma` prideľte hodnotu `20`,
- premennej `suma2` prideľte hodnotu *prefixový dekrement* premennej `suma + hodnota 5`,
- vypíšte hodnoty premenných `suma` a `suma2`,
- premennej `suma` prideľte hodnotu `20`,
- premennej `suma2` prideľte hodnotu *postfixový dekrement* premennej `suma + hodnota 5`,

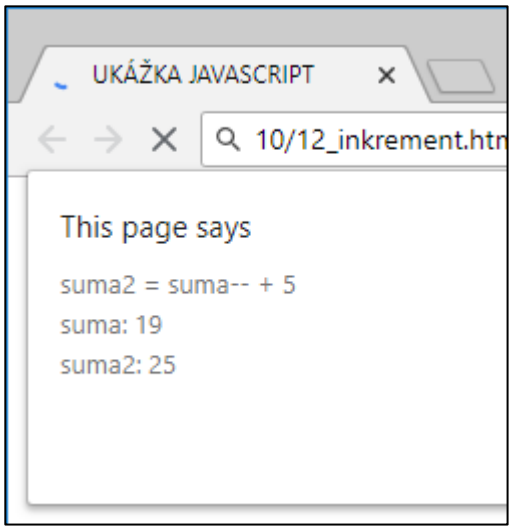
vypíšte hodnoty premenných `suma` a `suma2`.

```
var suma;
var suma2;

//prefix inkrementácia
suma = 20;
suma2 = ++suma + 5;
alert("suma2 = ++suma + 5" + "\n suma: " + suma +  "\nsuma2: " +
suma2);

//postfix inkrementácia
suma = 20;
suma2 = suma++ + 5;
alert("suma2 = suma++ + 5" + "\n suma: " + suma +  "\nsuma2: " +
suma2);

//prefix dekrementácia
suma = 20;
suma2 = --suma + 5;
alert("suma2 = --suma + 5" + "\n suma: " + suma +  "\nsuma2: " +
suma2);

//posfix dekrementácia
suma = 20;
suma2 = suma-- + 5;
alert("suma2 = suma-- + 5" + "\n suma: " + suma +  "\nsuma2: " +
suma2);
```









ÚLOHA 10.7

Porozmýšľajte, aké hodnoty budú uložené v premenných `cislo3` a `cislo4`. Výsledok si zapíšte a vytvorte HTML stránku s párovou značkou `<script>`, do ktorej vložte kód zo zadania nižšie. Výsledok premenných `cislo3` a `cislo4` vypíšte v dialógovom okne.

```
var cislo1 = 19;
var cislo2 = 25;
var cislo3 = --cislo1 + cislo2++;
var cislo4 = cislo1 + cislo2;
```

10.6.2 Relačné operátory

Relačné operátory porovnávajú hodnoty premenných a následne vracajú boolean hodnotu. Použiť môžeme relačné operátory väčšie (`>`), menšie (`<`), väčšie alebo rovné (`>=`) a menšie alebo rovné(`<=`)

POZNÁMKA

Operátory `<` `>` `<=` `>=` `==` `!=` `&&` `||` `!` sa väčšinou používajú pri zapisovaní podmienok pri vetvení (`if`) a cykloch (`while, do..while, for`).

PRÍKLAD 10.13

10/13_relacne.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `x` s číselnou hodnotou `5`,
- `y` s číselnou hodnotou `6`.

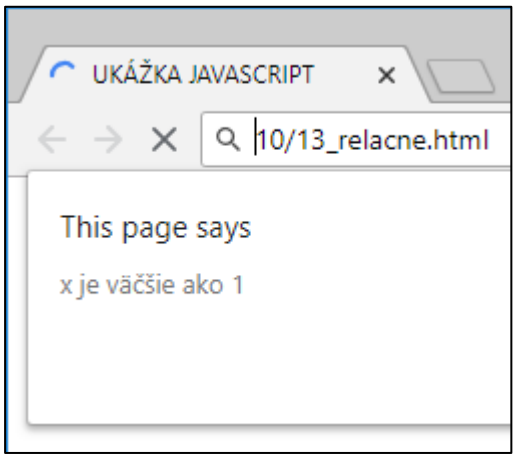
Pod premenné pridajte vetvenia:

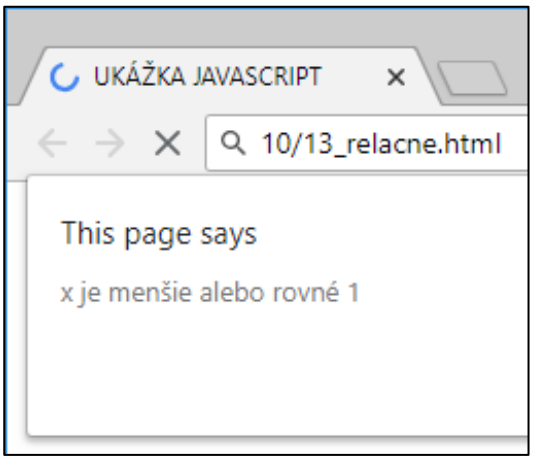
- ak `x` je väčšie ako `1`, tak vypíš `text x je väčšie ako 1`,
- ak `x` je väčšie alebo rovné `1`, tak vypíš text `x je väčšie alebo rovné 5`,
- ak `y` je menšie ako `7`, tak vypíš `y je menšie ako 7`,
- ak `y` je menšie alebo rovné `6`, tak vypíš text *y* `je menšie alebo rovné 6`.

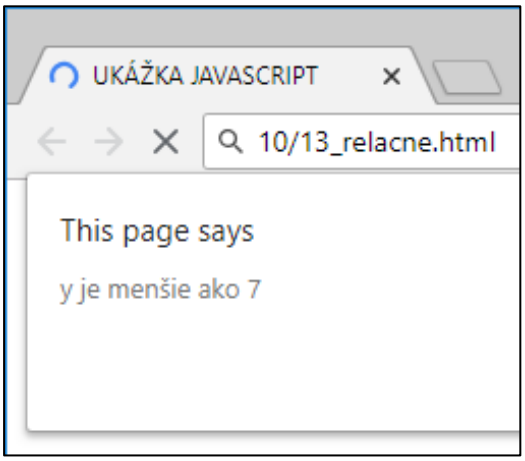
Koľkokrát sa zobrazí dialógové okno?

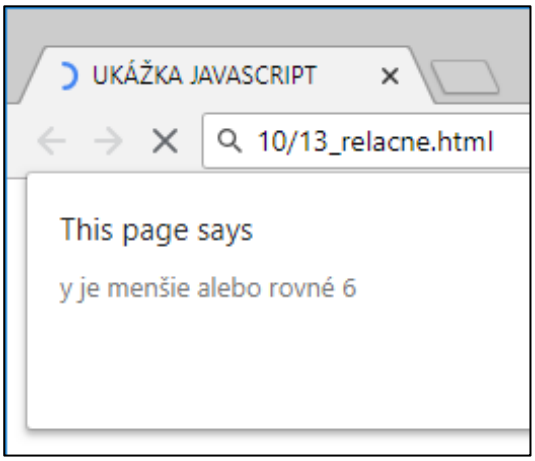
```
var x = 5;
var y = 6;

if (x > 1) {
      alert("x je väčšie ako 1");
}
if (x <= 1) {
      alert("x je menšie alebo rovné ako 1");
}

//menšie ako
if (y < 7)
{
     alert("y je menšie ako 7");
}

//menšie alebo rovné
if (y <= 5)
{
    alert("y je menšie alebo rovné 5");
}
```

Tento kód zobrazí nasledujúce dialógové okná:









ÚLOHA 10.8

Porozmýšľajte, pre akú celú číselnú hodnotu `y` z predchádzajúceho príkladu *10/13_relacne.html* sa nezobrazí dialógové okno?

Vyskúšajte zmeniť hodnoty premenných `x` a `y` a aj podmienky vo vetvení (zmeniť číselné hodnoty a znamienka).

10.6.3 Porovnávacie operátory

Porovnávacie operátory patria medzi najpoužívanejšie operátory pri programovaní. Vracajú hodnotu `true`, ak sú rovné, alebo `false`, ak nie sú rovné. Operátor rovnosti sa znázorňuje pomocou dvoch rovná sa (`==`), operátor nerovnosti sa zobrazuje pomocou výkričníka a znaku rovná sa (`!=`).

10/14_porovnavacie.html

PRÍKLAD 10.14

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `x` s číselnou hodnotou `5`,
- `y` s číselnou hodnotou `6`.

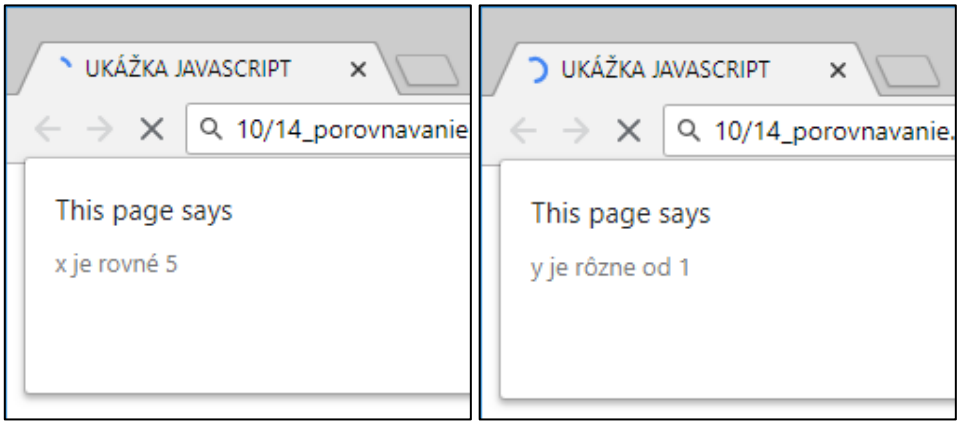
Pod premenné pridajte vetvenia:

- ak `x` sa rovná `5`, tak vypíš `text x je rovné 5`,
- ak `y` sa nerovná `1`, tak vypíš `text y je rôzne od 1`.

Koľkokrát sa zobrazí dialógové okno?

```
var x = 5;
var y = 6;

if (x == 5)
{
 alert("x je rovné 5");
}

if (y != 1)
{
    alert("y je rôzne od 1");
}
```

Tento kód zobrazí nasledujúce dialógové okná:



ÚLOHA 10.9

Porozmýšľajte, pre aké číselné hodnoty `x` a `y` z predchádzajúceho príkladu *10/14_porovnavanie.html* sa nezobrazia dialógové okná?

10.6.4 Logické operátory

JavaScript disponuje tromi logickými operátormi:

- **Negácia (NOT)** – znázorňujeme pomocou znaku výkričník (`!`). Výsledkom negácie je vždy Boolean hodnota, z tohto dôvodu ju môžeme použiť na ľubovoľný typ premennej.
- **Logický súčin (AND)** – znázorňujeme ho pomocou dvojitého ampersanda (`&&`). Použiť ho môžeme v prípade, že ho vložíme medzi dve hodnoty.
- **Logický súčet (OR)** – je reprezentovaný dvojitou zvislou čiarou (`||`). Podobne ako pri logickom súčine, použiť ho môžeme tak, že ho vložíme medzi dve hodnoty.

PRÍKLAD 10.15

10/15_logicke.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `x` s číselnou hodnotou `3`.

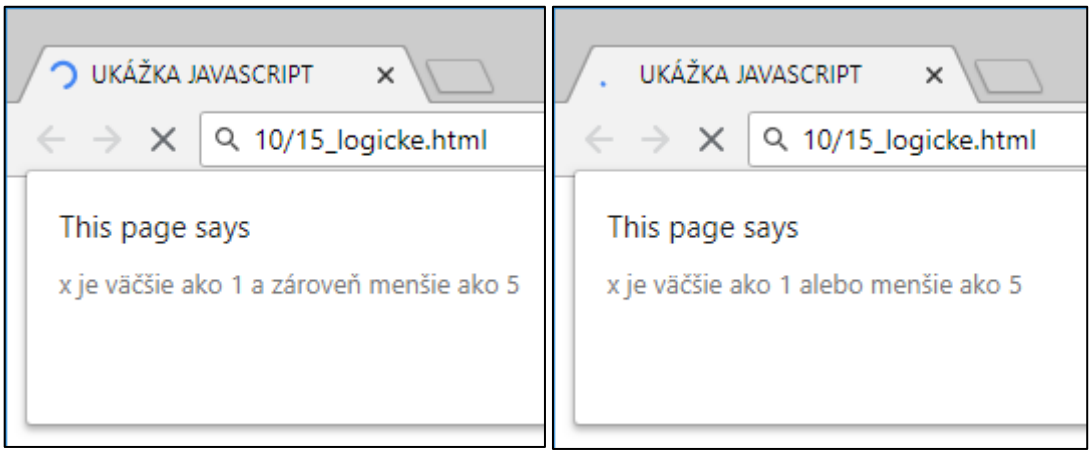
Pod premenné pridajte vetvenia:

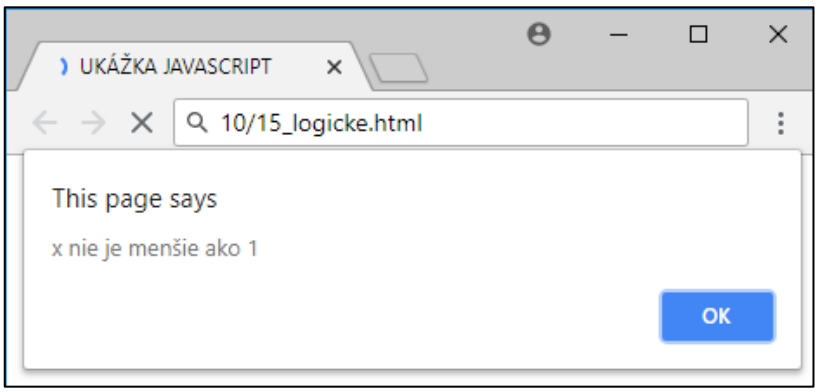
- ak `x` je väčšie ako `1` a súčasne je menšie ako `5`, tak vypíš `text x je väčšie ako 1 a zároveň menšie ako 5`. Ak nie je splnená podmienka tak vypíš text `haha`,

ak `x` je väčšie ako `1` alebo menšie ako `5` tak vypíš text `x je väčšie ako 1 alebo menšie ako 5`.

```
var x = 3;

//logický súčin - AND (musia platiť obe časti podmienky)
if ((x > 1) && (x < 5))
{
  alert("x je väčšie ako 1 a zároveň menšie ako 5");
}
else
{
  alert("haha");
}

//logický súčet - OR (musia platiť aspoň jedna časť podmienky)
if ((x > 1) || (x < 5))
{
  alert("x je väčšie ako 1 alebo menšie ako 5");
}

//logická negácia - NOT
if ( !(x < 1))
{
  alert("x nie je menšie ako 1");
}
```

Tento kód zobrazí nasledujúce dialógové okná:





10.6.5 Priraďovacie operátory

Priraďovacie operátory slúžia na priradenie hodnoty (pravá strana) k premennej (ľavá strana). Priradenie sa vykonáva pomocou znamienka rovná sa (=). Okrem jednoduchého operátora priradenia môžeme ešte použiť ďalšie operátory priradenia:

Operátor	Popis	Príklad	Skrátený zápis
=	Priradenie	x = y	x = y
+=	Sčítanie a priradenie	x = x + y	x += y
–=	Odčítanie a priradenie	x = x – y	x –= y
*=	Násobenie a priradenie	x = x * y	x *= y
/=	Delenie a priradenie	x = x / y	x /= y
%=	Zvyšok po delení a priradenie	x = x % y	x %= y

Tieto príkazové operátory vykonajú príkaz (uložia do premennej novú hodnotu a súčasne ju upravia).

POZNÁMKA

Použitie týchto príkazov nemá vplyv na rýchlosť vykonávania operácii. Jedná sa len o skrátený zápis.

PRÍKLAD 10.16

10/16_priradenie.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `x` bez priradenia hodnoty.

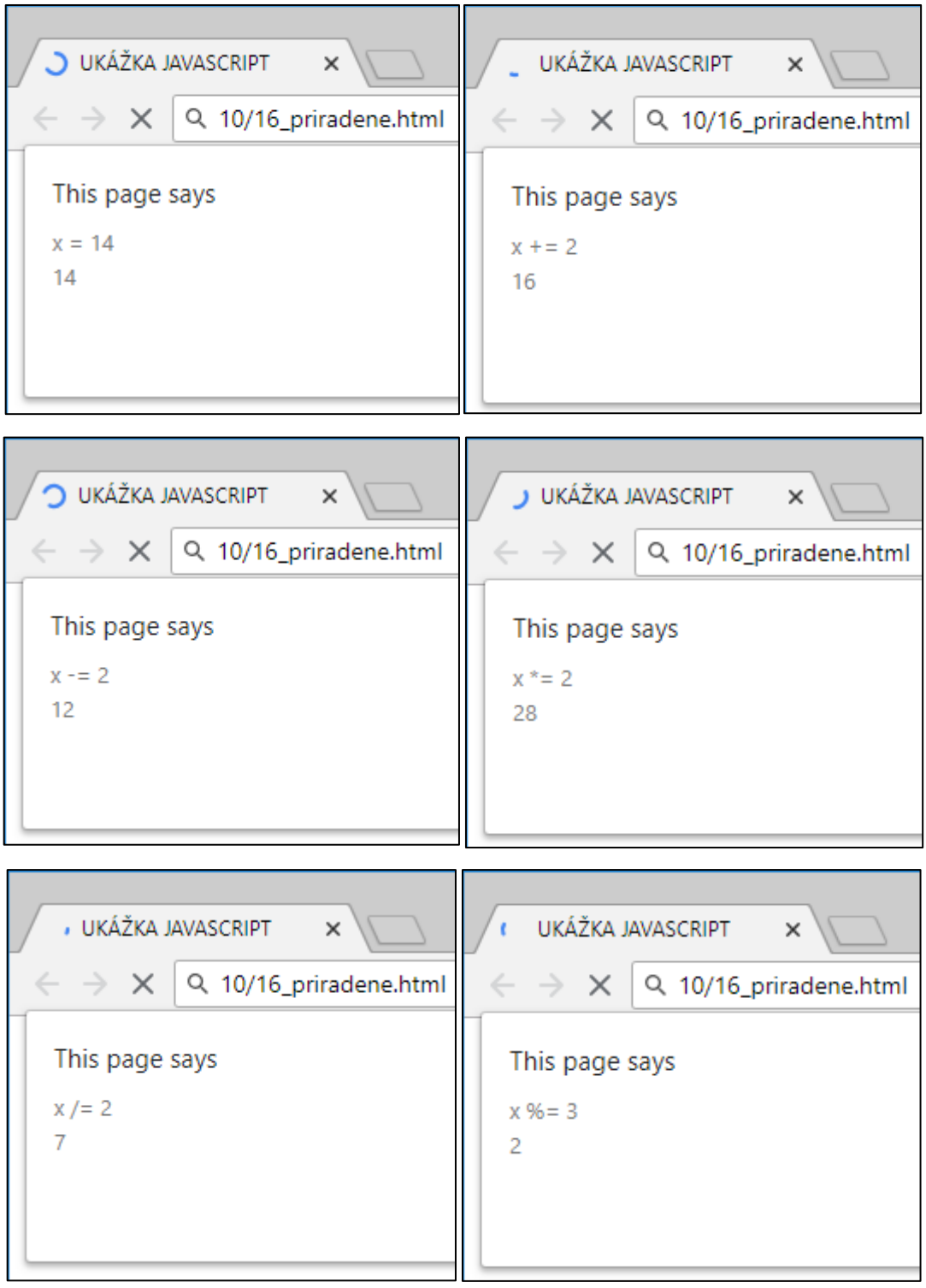
Postupne priraďujte premennej x hodnoty a súčasne ich vypisujte v dialógovom okne:

- premennej `x` priraďte hodnotu `14`,
- premennej `x` sčítajte a priraďte hodnotu `2`,
- premennej `x` priraďte hodnotu `14` a následne pomocou zloženého priradenia sčítajte a priraďte hodnotu `2`,
- premennej x priraďte hodnotu `14` a následne pomocou zloženého priradenia odčítajte a priraďte hodnotu `2`,
- premennej `x` priraďte hodnotu `14` a následne pomocou zloženého priradenia násobte a priraďte hodnotu `2`,
- premennej `x` priraďte hodnotu `14` a následne pomocou zloženého priradenia vydeľte a priraďte hodnotu `2`,

premennej x priraďte hodnotu `14` a následne pomocou zloženého priradenia urobte zvyšok po delení a priraďte hodnotu `3`.

```
var x;

//jednoduché priradenie
x = 15;
alert("x = 15 \n" + x);

//zložené priradenie +=
x = 14;
x += 2;
alert("x += 2 \n" + x);

//zložené priradenie -=
x = 14;
x -= 2;
alert("x -= 2 \n" + x);

//zložené priradenie *=
x = 14;
x *= 2;
alert("x *= 2 \n" + x);

//zložené priradenie /=
x = 14;
x /= 2;
alert("x /= 2 \n" + x);

//zložené priradenie %=
x = 14;
x %= 3;
alert("x %= 3 \n" + x);
```

Tento kód zobrazí nasledujúce dialógové okná:



ÚLOHA 10.10

Z predchádzajúceho príkladu *10/16_priradenie.html* odstráňte všetky jednoduché priradenie `x = 14`. Pred spustením stránky v prehliadači porozmýšľajte, aký bude výsledok po všetkých operáciách. Výsledok si overte spustením stránky v prehliadači.

10.6.6 Priorita vykonávania operátorov

Pri písaní zložitejších výrazov, ktoré obsahujú súčasne niekoľko operátorov, je potrebné poznať prioritu operátorov. Táto určuje, v akom poradí budú jednotlivé operácie vykonané, čo môže mať vplyv na výslednú hodnotu výrazu. V nasledujúcej tabuľke je uvedený prehľad vybraných operátorov počnúc operátormi s najvyššou prioritou.

Operátor	Popis	Asociativita
.	prístup k prvku	zľava doprava
[ ]	prístup k prvku poľa	zľava doprava
new ( )	konštruktor s argumentom	-
( )	volanie funkcie	zľava doprava
++ −−	inkrementácia, dekrementácia	sprava doľava
+ −	unárne +, unárne – (negácia)	sprava doľava
!	logická negácia – NOT	sprava doľava
delete	zrušenie definície	sprava doľava
typeof	vrátenie dátového typu	sprava doľava
void	vrátenie nedefinovaného typu	sprava doľava
* / %	násobenie, delenie, zvyšok po delení	zľava doprava
+ −	sčítanie, odčítanie	zľava doprava
< <=	menšie, menšie alebo rovné	zľava doprava
> >=	väčšie, väčšie alebo rovné	zľava doprava
in	kontrola existencie	zľava doprava
instanceof	preverenie typu objektu	zľava doprava
== !=	rovné, rôzne	zľava doprava
&&	logický súčin – AND	zľava doprava
||	logický súčet – OR	zľava doprava
?:	podmienený výraz	sprava doľava
= += −= *= /=	priradenie	sprava doľava
,	viacnásobné vyhodnotenie	zľava doprava

10.7 Riadiace príkazy

Riadiace príkazy sa používajú v programovaní na zmenu poradia vykonávaných príkazov, vetvenie programu v závislosti na zmenách programu alebo na opakovanie určitých častí programu pomocou cyklu.

10.7.1 Príkaz if

Príkaz `if` sa používa k podmienenému vetveniu programu. Na základe tohto príkazu sa vykoná/nevykoná určitá časť zdrojového kódu. Syntax príkazu `if` je:

```
if (podmienka) {
      //prikaz1
} else {
      //príkaz2
}
```

POZNÁMKA

Dokonca je možné napísať aj príkaz, ktorý nebude obsahovať žiadny riadok, ale táto možnosť sa neodporúča.

Zložené zátvorky v tomto prípade nie sú povinné, ale odporúčajú sa použiť.

Podmienka môže byť ľubovoľný výraz. Ak je podmienka splnená, vykoná sa `príkaz1`, inak sa vykoná `príkaz2`. Namiesto príkazu je možné zadať aj niekoľko viac príkazov za sebou.

PRÍKLAD 10.17

10/17_if1.html

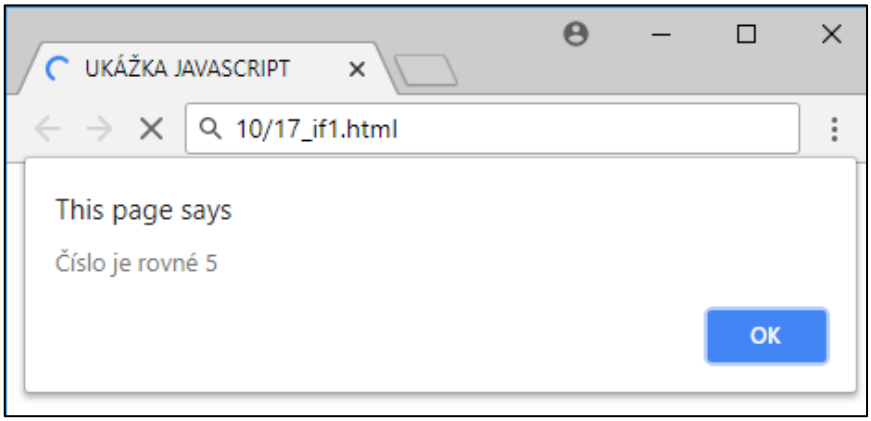
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `m` s číselnou hodnotou `5`.

Pod premennú vložte vetvenie: ak `m` je rovné `5`, tak vypíš text `Číslo je rovné 5`.

```
var m = 5;

//vetvenie programu na zakladenie platnosti podmienky
if (m == 5)
{
   //táto časť sa vykoná iba v prípade, ak podmienka platí
   var text = "Číslo je rovné 5";
   alert(text);
}
```

Tento kód zobrazí nasledujúce dialógové okno:



PRÍKLAD 10.18

10/18_if2.html

Do predchádzajúceho príkladu 10.17 pridajte do podmienky časť `else`, ktorá sa vykoná v prípade, ak podmienka nie je splnená.

```
var m = 5;

//vetvenie programu na zakladenie platnosti podmienky
if (m == 5)
{
   //táto časť sa vykoná iba v prípade, ak podmienka platí
   var text = "Číslo je rovné 5";
   alert(text);
}
else
{
   //táto časť sa vykoná v prípade, ak podmienka neplatí
   var text = "Číslo nie je rovné 5"
   alert(text);
}
```

Príkaz `if` sa nemusí ukončiť hneď v časti `else`, ak časť `if` nie je splnená. Môžeme pridať ešte jednu, alebo viac častí `else if`, kde nastavíme možné stavy, kedy je výraz splnený.

```
if (i > 25) {
      alert("Číslo je väčšie ako 25");
} else if (i < 25) {
      alert("Číslo je menšie ako 25");
} else {
      alert("Číslo je rovné 25");
}
```

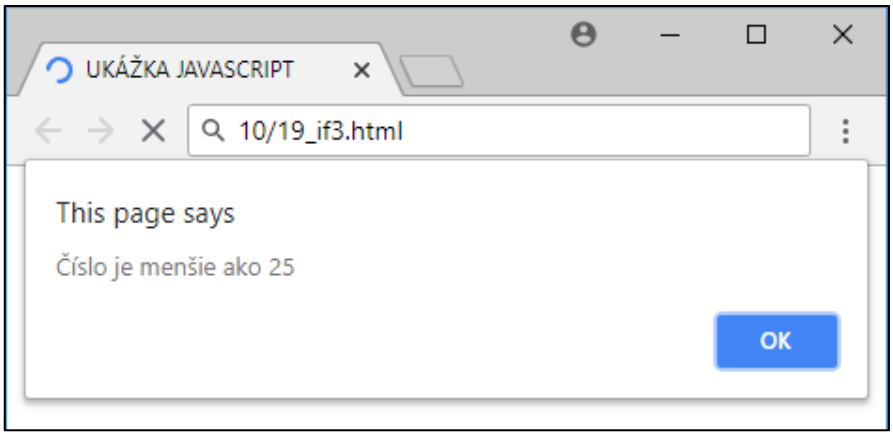
PRÍKLAD 10.19

10/19_if3.html

Do predchádzajúceho príkladu 10.18 pridajte do podmienky časť `else if`, ktorá sa vykoná v prípade, ak prvá podmienka nie je splnená.

```
var m = 5;

//vetvenie programu na zakladenie platnosti podmienky
if (m > 25)
{
   var text = "Číslo je väčšie 25";
   alert(text);
}
else if (m < 25)
{
   //táto časť sa vykoná v prípade, ak podmienka neplatí
   var text = "Číslo je menšie ako 25"
   alert(text);
}
else
{
   //táto časť sa vykoná v prípade, ak podmienka neplatí
   var text = "Číslo nie je väčšie ako 25 ani menšie ako 25"
   alert(text);
}
```

Tento kód zobrazí nasledujúce dialógové okno:



10.7.2 Príkaz while

Príkaz `while` je cyklus s podmienkou na začiatku. Príkazy v tomto cykle sa nemusia nikdy vykonať, pokiaľ nie je splnená podmienka. Podmienka určujúca opakovanie cyklu sa vyhodnotí hneď na začiatku a následne pred každým opakovaním príkazov v cykle.

```
while (vyraz) {
      //prikaz
}
```

10/20_while.html

PRÍKLAD 10.20

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `i` s číselnou hodnotou `0`.

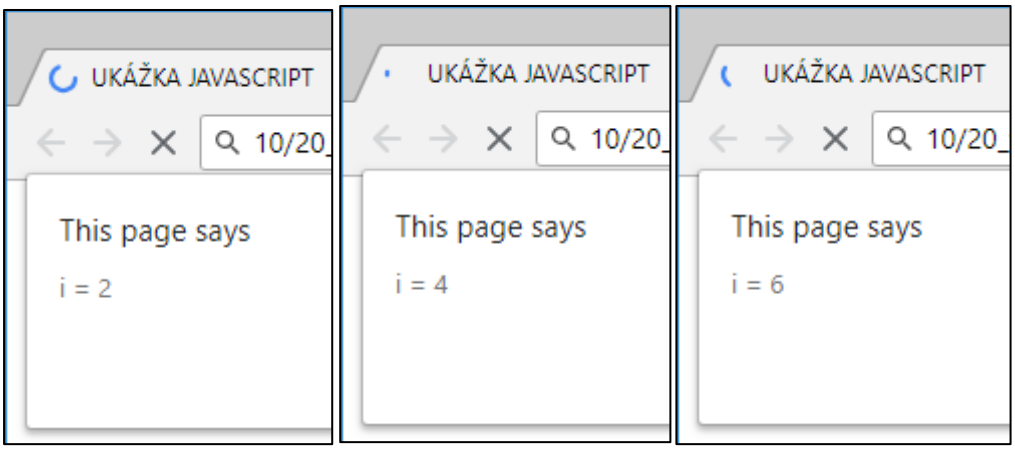
V cykle s podmienkou na začiatku opakujte nasledujúce príkazy, kým `i` je menšie ako `10`:

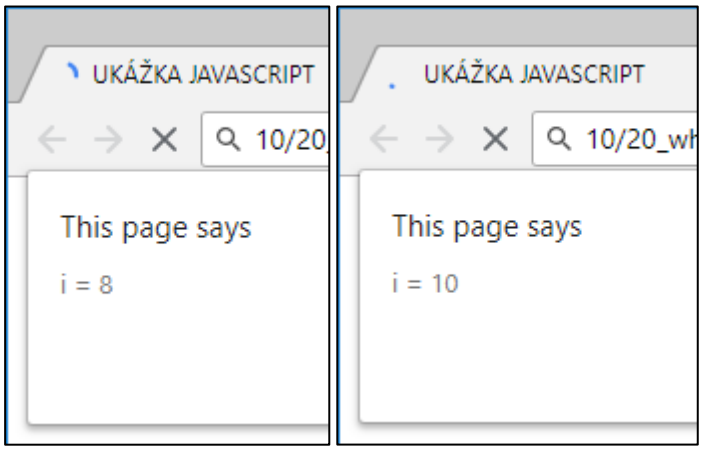
- zväčšite hodnotu `i` o hodnotu `2`,
- vypíšte hodnotu `i` v dialógovom okne.

Zamyslite sa: Koľkokrát sa zobrazí dialógové okno?

```
var i = 0;

//podmienka sa najprv vyhodnotí, ak platí, tak sa vykoná telo
cyklu
while (i < 10)
{
     //telo cyklu sa bude opakovane vykonávať kým bude platiť
     podmienka
     i += 2;
     alert("i = " + i);
}
```

Tento kód zobrazí nasledujúce dialógové okná:





ÚLOHA 10.11

Vytvorte HTML stránku, ktorá po spustení zobrazí dialógové okno, v ktorom budú vypísané všetky nepárne čísla od `1` a menšie ako `20`. Úlohu riešte pomocou cyklu `while`.

10.7.3 Príkaz do-while

Cyklus `do-while` je cyklus s podmienkou na konci. Tento cyklus použijeme vtedy, keď chceme, aby sa príkazy v cykle vykonali aspoň jeden raz. Až po vykonaní príkazov v cykle sa vyhodnotí podmienka, ktorá určuje, či sa bude cyklus ešte opakovať. Syntax:

```
do {
      //prikazy
} while (vyraz);
```

10/21_dowhile.html

PRÍKLAD 10.21

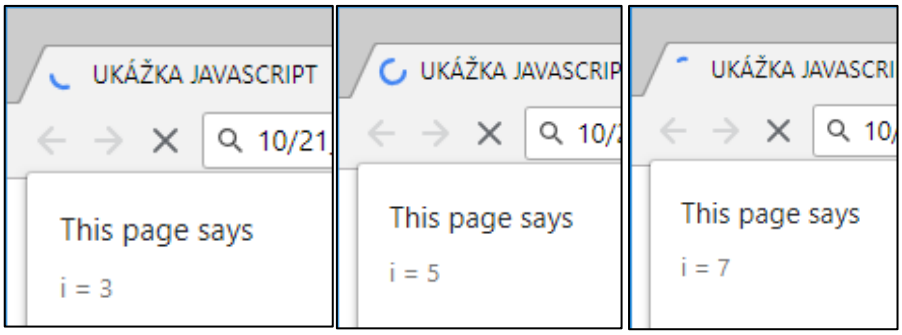
Upravte predchádzajúci príklad 10.20 (súbor 10/20_while.html) tak, aby ste použili namiesto cyklu s podmienkou na začiatku cyklus s podmienku na konci.

Pri vytváraní premennej `i` ju inicializujte na hodnotu `1`.

Zamyslite sa, aké čísla sa zobrazia v dialógovom okne.

```
var i = 1;

//vykoná sa telo cyklu a na koniec sa overí, či podmienka platí a
teda, či sa má cyklus ešte opakovať
do
{
//telo cyklu sa bude opakovane vykonávať kým bude platiť
podmienka
i += 2;
    alert("i = " + i);
} while (i < 10);
```

Tento kód zobrazí nasledujúce dialógové okná:



ÚLOHA 10.12

Vytvorte HTML stránku, ktorá po spustení zobrazí dialógové okno, v ktorom budú vypísané všetky nepárne čísla od `1` a menšie ako `20`. Úlohu riešte pomocou cyklu `do-while`.

10.7.4 Príkaz for

Cyklus `for` sa označuje aj ako cyklus s pevným počtom opakovaní. Používa sa v prípade, keď vieme pomocou počítadla spočítať, koľkokrát sa má cyklus zopakovať. Na rozdiel od predošlých cyklov v ňom priamo inicializujeme premennú – počítadlo počiatočnou hodnotou a definujeme podmienku, po akú hodnotu má počítadlo počítať. Zároveň vieme nastaviť krok, s akým sa má počítadlo inkrementovať alebo dekrementovať.

```
for(inicializacia; test; inkrementácia) {
      //prikaz
}
```

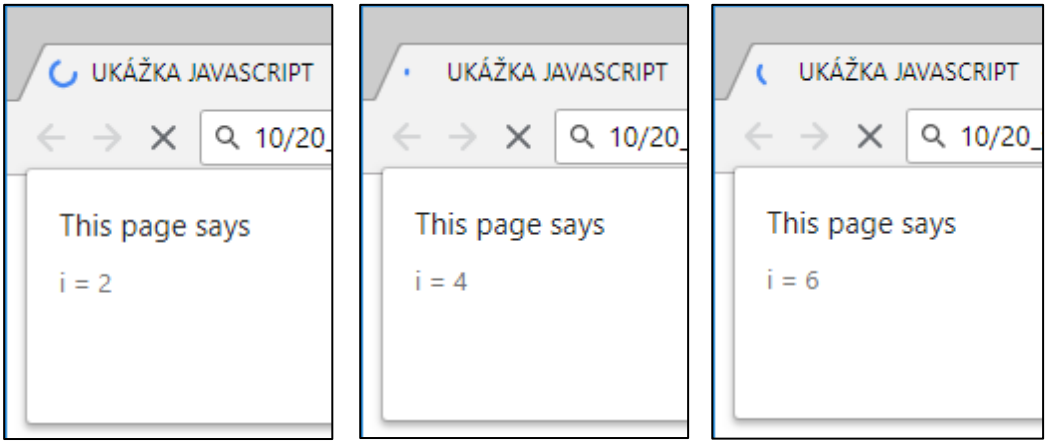
PRÍKLAD 10.22

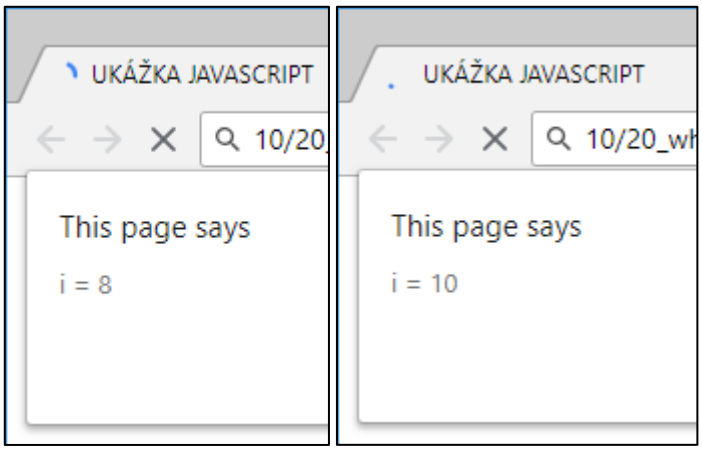
Upravte príklad 10.20 (súbor 10/20_while.html) tak, aby ste použili namiesto cyklu s podmienkou na začiatku cyklus `for`.

Zamyslite sa, aké čísla sa zobrazia v dialógovom okne.

10/22_for.html

```
//počítadlo i bude počítať od 0
//telo cyklu sa bude opakovat kým bude i menšia ako 10
//pri každom opakovaní cyklu sa počítadlo i zväčši o 2
for(var i = 0; i < 10; i += 2)
{
     alert("i = " + i);
}
```





ÚLOHA 10.13

Vytvorte HTML stránku, ktorá po spustení zobrazí dialógové okno, v ktorom budú vypísané všetky nepárne čísla od `1` a menšie ako `20`. Úlohu riešte pomocou cyklu `for`.

10.8 Funkcie

Funkcia je skupina príkazov, ktoré sa nachádzajú v určitom bloku. Funkcie teda umožňujú zapuzdrenie príkazov, ktoré je možné spúšťať kdekoľvek, kedykoľvek a ľubovoľný počet krát. Funkcie začínajú v jazyku JavaScript kľúčovým slovom `function`, za ktorým nasledujú okrúhle zátvorky, v ktorých sa môžu nachádzať parametre. Za zátvorkami sa nachádza telo funkcie v zložených zátvorkách. Každá funkcia má svoj názov, ktorý si definuje programátor sám. Pri písaní názvu funkcie by sme mali dodržiavať určité konvencie – funkcie by mali začínať malým písmenom, v prípade viacslovného pomenovania začínať každé nové slovo veľkým písmenom (Camel case). Funkcie môžu obsahovať písmená, čísla, podčiarkovníky, znak doláru (podobne ako pri premenných, kapitola 10.1).

Syntax:

```
function nazovFunkcie(parameter0, parameter1,..., parameterN) {
      //prikazy
}
```

Funkcie je možné zavolať v kóde viackrát zavolaním názvu funkcie, alebo vyvolaním udalosti (kapitola 12).

```
povedzAhoj("Janko","ako sa máš?");     //Ahoj Janko, ako sa máš?
povedzAhoj("Kubko","a ty sa máš ako?");
      //Ahoj Kubko, a ty sa ako sa máš?
```

Funkcia môže obsahovať ľubovoľný počet argumentov, rôzneho dátového typu. V prípade, že funkcia neobsahuje žiadny parameter uvedieme iba prázdne zátvorky.

PRÍKLAD 10.23

10/23_funkcia.html

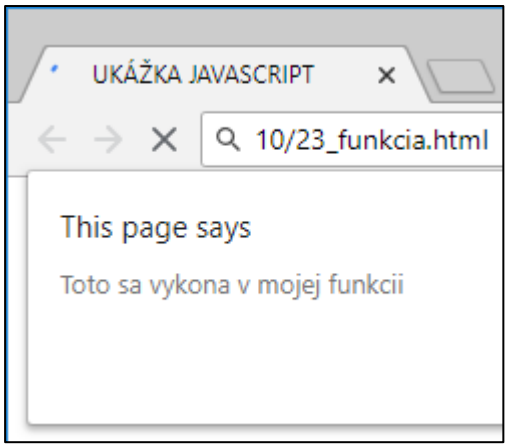
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte funkciu s názvom `mojaFunkcia()`. V tele funkcie vytvorte premennú `text`, ktorej pridajte hodnotu `Toto sa vykona v mojej funkcii`. Hodnotu premennej zobrazte v dialógovom okne.

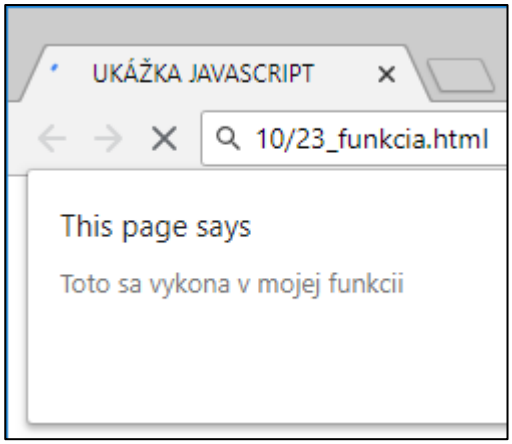
Pod definovaním funkcie zavolajte dvakrát funkciu `mojaFunkcia()`.

Zamyslite sa: Koľkokrát sa zobrazí dialógové okno?

```
//definicia funkcie
function mojaFunkcia()
{
     var text = "Toto sa vykona v mojej funkcii";
     alert(text);
}

//volanie funkcie zabezpečí jej vykonanie
mojaFunkcia();

//funkciu je možné zavolať aj viackrát
mojaFunkcia();
```

Tento kód zobrazí nasledujúce dialógové okná:





PRÍKLAD 10.24

10/24_parametre.html

Upravte príklad 10.23 (súbor 10/23_funkcie.html) tak, aby ste pri definovaní funkcie použili dva parametre:

- `meno`,
- `sprava`.

Hodnoty parametrov zobrazte v tele funkcie pomocou dialógového okna v tvare `Ahoj meno sprava.`

Pod definovaním funkcie zavolajte najskôr funkciu s parametrom:

- *`meno = Janko`*,
- *`sprava = ako sa máš`*.

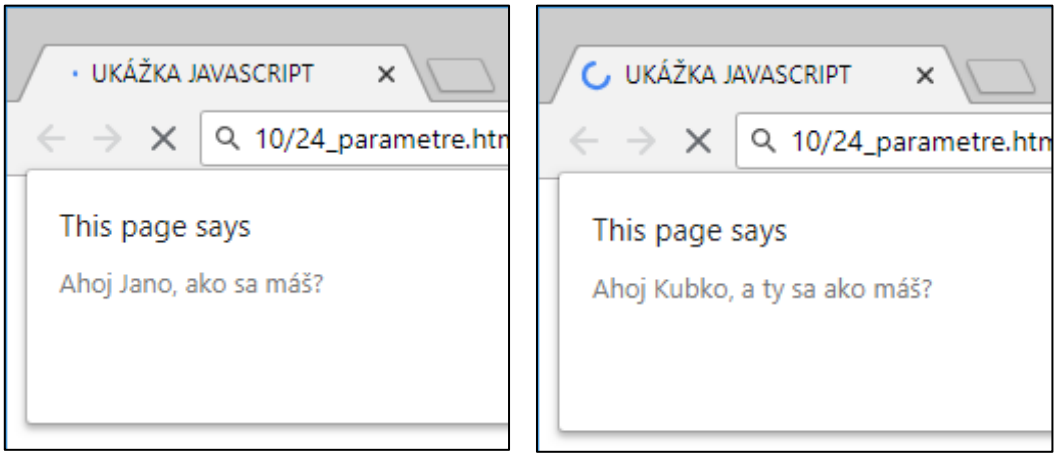
Pod volaním funkcie vytvorte premenné:

- `ja` s hodnotou `Kubko`,
- `pozdrav` s hodnotou `a ty sa ako máš?`

Zavolajte funkciu mojaFunkcia() s parametrami premenných *ja* a *pozdrav*.

```
//definicia funkcie
function mojaFunkcia(meno, sprava)
{
     alert("Ahoj " + meno + ", " + sprava);
}

//volanie funkcie zabezpečí jej vykonanie
mojaFunkcia("Jano", "ako sa máš?");

//pri volaní funkcie je možné použiť ako parametre aj premenné
var ja = "Kubko", pozdrav = "a ty sa ako máš?"

mojaFunkcia(ja, pozdrav);
```

Tento kód zobrazí nasledujúce dialógové okná:



Funkcie môžeme zavolať a následne sa vykoná činnosť, ktorú definujeme v tele funkcie. V niektorých prípadoch budeme chcieť zavolať funkciu, ktorá napríklad niečo vypočíta a následne tento výsledok môžeme niekde použiť. Pri deklarovaní funkcie nemusíme špecifikovať, či funkcia vracia nejakú hodnotu. Každá funkcia môže vrátiť ľubovoľnú hodnotu pomocou príkazu `return`, za ktorým nasleduje hodnota, ktorá sa má vrátiť.

PRÍKLAD 10.25

10/25_return.html

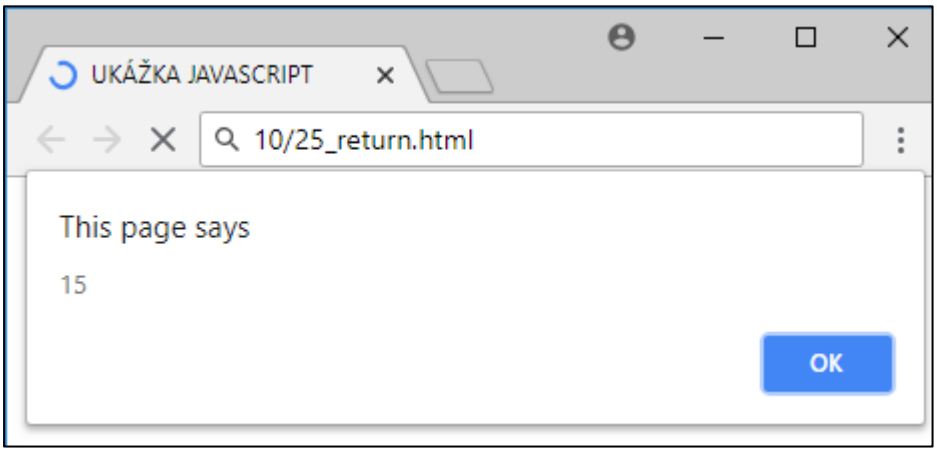
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `vysledok` s hodnotou – zavolaním funkcie `spocitaj(10,5)` .

Vytvorte funkciu `spocitaj()` s dvoma parametrami. V tele funkcie použite príkaz `return`, ktorý po zavolaní funkcie spočíta hodnoty parametrov.

```
function spocitaj(cislo1, cislo2)
{
      return cislo1+cislo2;
}

var vysledok = spocitaj(10,5);
alert(vysledok);        //15
```

Tento kód zobrazí nasledujúce dialógové okno:



Príkaz `return` sa používa k okamžitému ukončeniu funkcie a návrat k miestu, kde bola funkcia zavolaná. Kód, ktorý sa nachádza za týmto príkazom bude ignorovaný.

PRÍKLAD 10.26

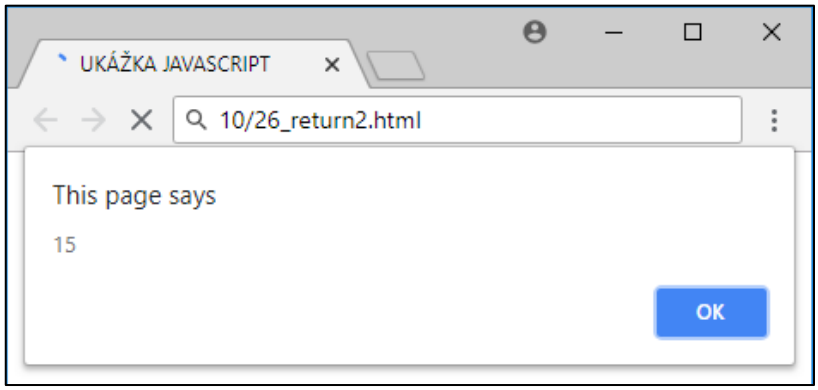
10/26_return2.html

Upravte príklad 10.26 (súbor 10/26_return.html) tak, aby ste v tele funkcie `spocitaj()` zavolali funkciu `alert()` s parametrom hodnoty premennej `vysledok`. Funkciu `alert()` zavolajte za príkazom `return`.

Zamyslite sa, koľkokrát sa zobrazí dialógové okno.

```
//definicia funkcie
function spocitaj(cislo1, cislo2)
{
     return cislo1 + cislo2;
     alert("Časť za príkazom return sa nevykoná!")
}

//volanie funkcie zabezpečí jej vykonanie
var vysledok = spocitaj(10, 5);
alert(vysledok);
```

Tento kód zobrazí nasledujúce dialógové okno:



10/27_vacsie.html

PRÍKLAD 10.27

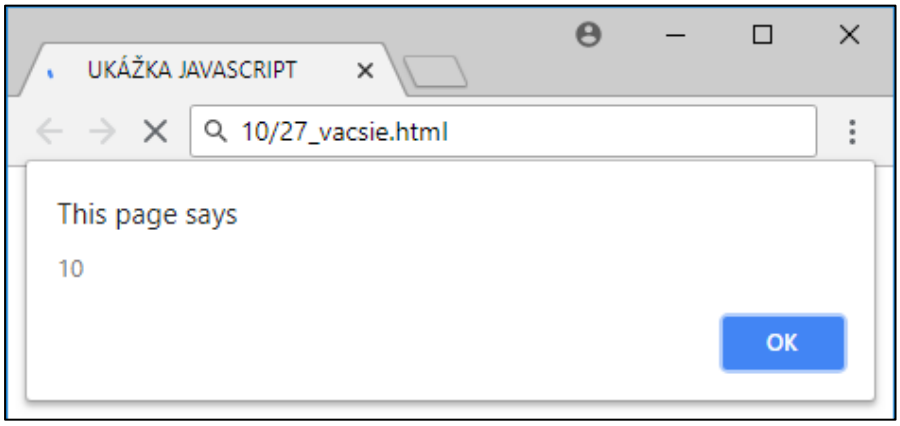
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `vysledok` s hodnotou – zavolaním funkcie `vacsieCislo(10,5)` .

Vytvorte funkciu `vacsieCislo()` s dvoma parametrami. V tele funkcie použite vetvenie `if` ak je hodnota parametra `cislo1` väčšia ako hodnota parametra `cislo2` tak pomocou príkazu `return` vráť hodnotu parametra `cislo1`, ak je menšie tak vráť hodnotu parametra `cislo2`. Výsledok z funkcie `vacsieCislo()` vypíš v dialógovom okne.

```
function vacsieCislo(cislo1, cislo2)
{
     if (cislo1 > cislo2)
     {
          return cislo1;
     }
     else
     {
          return cislo2;
     }
}

//volanie funkcie zabezpečí jej vykonanie
var vysledok = vacsieCislo(10, 5);
alert(vysledok);
```

Tento kód zobrazí nasledujúce dialógové okno:



ÚLOHA 10.14

Upravte príklad 10.27 – vytvorte ďalšie funkcie na odčítanie, násobenie a delenie. V dialógovom okne vypíšte text:

```
Číslo 1: 10
Číslo 2: 5
Výsledok sčítania: (tu bude výsledok vypočítaný pomocou
funkcie spocitaj())
Výsledok odčítania: (tu bude výsledok vypočítaný pomocou
funkcie odpocitaj())
Výsledok násobenia: (tu bude výsledok vypočítaný pomocou
funkcie vynasob())
Výsledok delenia: (tu bude výsledok vypočítaný pomocou
funkcie vydeľ())
```

10.9 Polia

Polia v programovaní reprezentujú usporiadaný zoznam dát, na ktorý sa odkazujem spoločným názvom. Veľkosť poľa je daná dynamicky – závisí od počtu prvkov (ak pridáme ďalší prvok, automaticky sa zväčší aj pole). K jednotlivým prvkom poľa môžeme pristupovať pomocou jeho indexu, pričom prvý prvok má index `0`. V JavaScripte, na rozdiel od ostatných jazykov, môže pole uchovávať v každom prvku iný typ dát (`String, int,`...):

index 0	index 1	index 2	index 3	index 4
`"cervena"`	`"modra"`	`true`	`5`	`"ahoj"`

Polia sa vytvárajú dvomi spôsobmi:

1) Bez kľúčového slova **`new`** a konštruktora **`Array()`**

```
var nazovPola = [item1, item2, …, itemN];
```

POZNÁMKA

Prvý spôsob nie je funkčný v prehliadači Internet Explorer 8.

2) S kľúčovým slovom `new` a konštruktorom `Array()`

```
var nazovPola = new Array(item1, item2, …, itemN);
```

Príklad

```
var farby1 = ["biela","modra","cervena"];
var farby2 = new Array ("zelena","fialova","hneda");
```

Obidva spôsoby vytvorenia poľa sú správne.

K jednotlivým prvkom poľa (pri obidvoch spôsoboch) pristupujeme tak, že za premennú vložíme hranaté zátvorky, do ktorých uvedieme index prvku.

PRÍKLAD 10.28

10/28_pole.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena`.

V dialógovom okne postupne vypíšte:

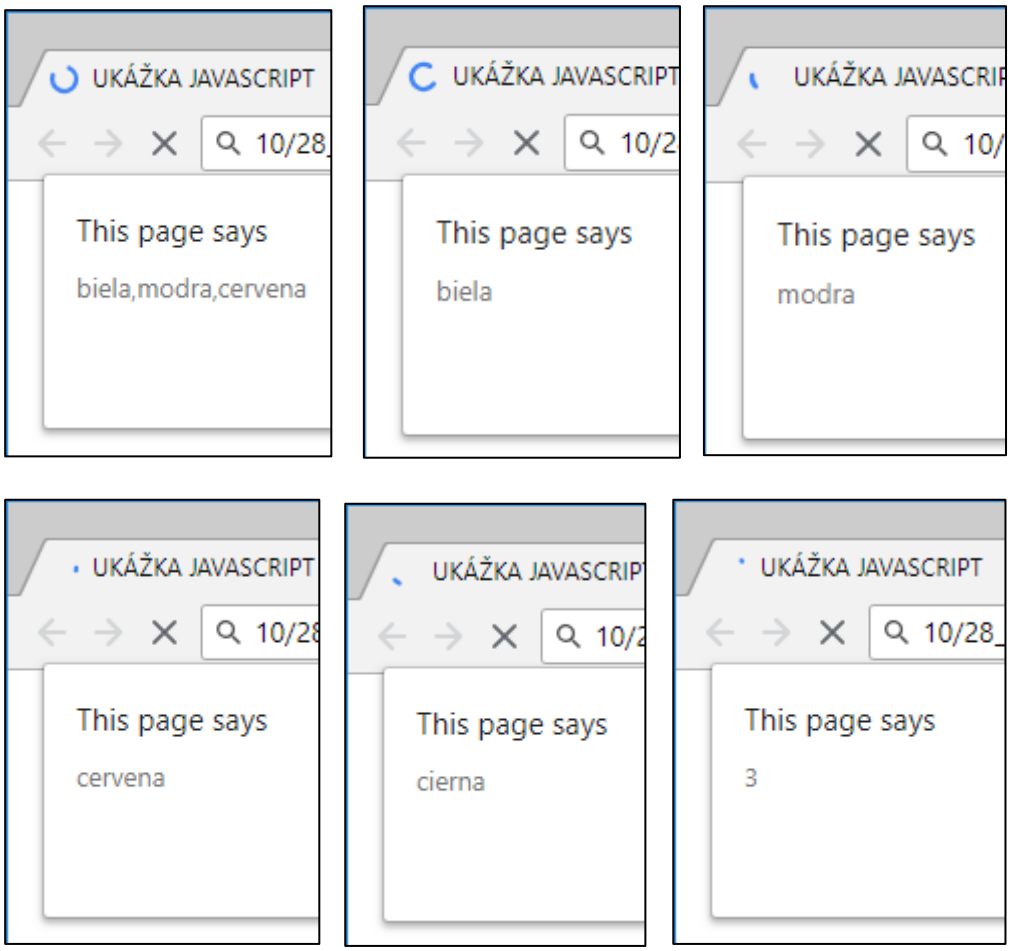
- všetky farby,
- prvú farbu,
- druhú farbu,
- tretiu farbu.

Zmeňte hodnotu prvého poľa na farbu `cierna` a vypíšte túto farbu v dialógovom okne.

Vypíšte v dialógovom okne veľkosť poľa – premennej farby.

```
 //vytvorenie poľa a inicializácia troch prvkov poľa
var farby = new Array("biela", "modra", "cervena");

//vypísanie všetkých prvkoc poľa
alert(farby);

//prístup k prvkom poľa pomocou indexu
alert(farby[0]);
alert(farby[1]);
alert(farby[2]);

//zmena hodnoty prvku poľa
farby[0] = "cierna";
alert(farby[0]);

//zistenie počtu prvkov poľa
alert(farby.length);
```

Tento kód zobrazí nasledujúce dialógové okná:



Hodnoty uložené v poliach môžeme kedykoľvek meniť:

```
farby1[0];  //biela
farby1[1] = "oranzova";      //oranzova farba nahradi modru
```

Na zistenie počtu prvkov v poli je možné použiť vlastnosť `length`, ktorá vždy vráti hodnotu `0` alebo vyššiu.

```
alert(farby1.length); //3
```

Metódy pre prácu s poľom

- **pop()** – odstráni posledný prvok z poľa
- **push()** – pridá na koniec poľa ľubovoľný počet argumentov z **`push()`**
- **shift()** – odstráni prvý prvok z poľa
- **unshift()** – pridá prvok na začiatok poľa

10/29_poleMetody.html

PRÍKLAD 10.29

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

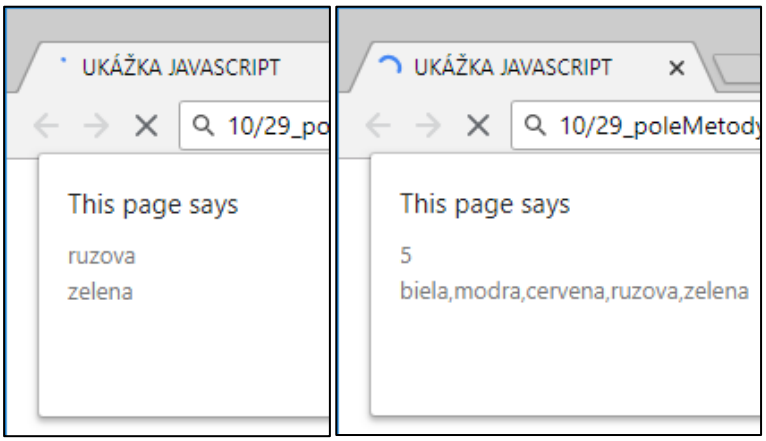
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena`.

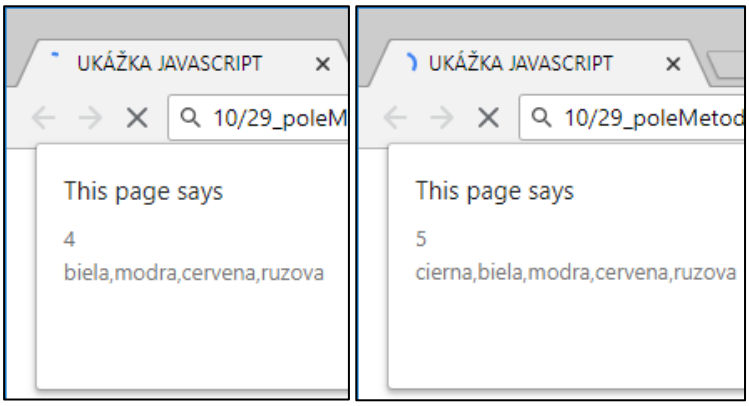
Vykonajte nasledujúce operácie s poľom:

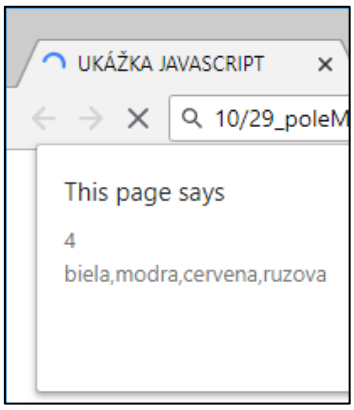
- na koniec poľa pridajte farby `ruzova, zelena`,
- vypíšte pole s indexom `3` a na novom riadku pole s indexom `4`,
- v dialógovom okne postupne vypíšte počet prvkov poľa `farby` a súčasne na novom riadku vypíšte všetky hodnoty poľa,
- odstráňte posledný prvok z poľa,
- v dialógovom okne postupne vypíšte počet prvkov poľa farby a súčasne na novom riadku vypíšte všetky hodnoty poľa,
- pridajte na začiatok poľa `ciernu` farbu,
- v dialógovom okne postupne vypíšte počet prvkov poľa farby a súčasne na novom riadku vypíšte všetky hodnoty poľa,
- odstráňte prvý prvok poľa,
- v dialógovom okne postupne vypíšte počet prvkov poľa farby a súčasne na novom riadku vypíšte všetky hodnoty poľa.

```
//vytvorenie poľa a inicializácia troch prvkov poľa
var farby = new Array("biela", "modra", "cervena");

//pridanie nových prvkov poľa na koniec
farby.push("ruzova", "zelena");
alert(farby[3] + "\n" + farby[4]);
alert(farby.length + "\n" + farby)

//odstránenie posledného prvku z poľa
farby.pop();
alert(farby.length + "\n" + farby)

//pridanie prvku na začiatok poľa
farby.unshift("cierna");
alert(farby.length + "\n" + farby)

//odstránenie prvku zo začiatok poľa
farby.shift();
alert(farby.length + "\n" + farby)
```

Tento kód zobrazí nasledujúce dialógové okná:







- **reverse()** – zmení poradie prvkov poľa
- **sort()** – zoradí prvky od najmenšieho po najväčší prvok (resp. abecedne)

10/30_polePoradie.html

PRÍKLAD 10.30

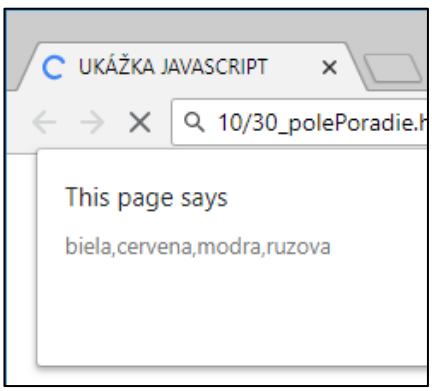
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

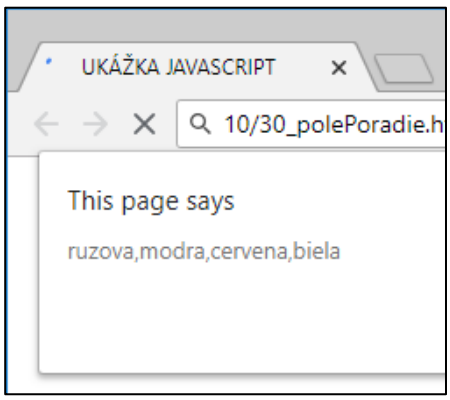
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena, ruzova`.

Vykonajte nasledujúce operácie s poľom:

- zoraďte prvky poľa podľa abecedy a zoradené pole vypíšte v dialógovom okne,
- obráťte poradie prvkov v poli a obrátené pole vypíšte v dialógovom okne.

```
//vytvorenie poľa a inicializácia troch prvkov poľa
var farby = new Array("biela", "modra", "cervena", "ruzova");

//zoradenie prvkov poľa podľa abecedy
farby.sort();
alert(farby);

//obrátenie poradia prvkov v poli
farby.reverse();
alert(farby);
```





- **concat()** – skopíruje pole a umožňuje pridať ďalšie prvky na koniec poľa

PRÍKLAD 10.31

10/31_poleKopirovanie.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

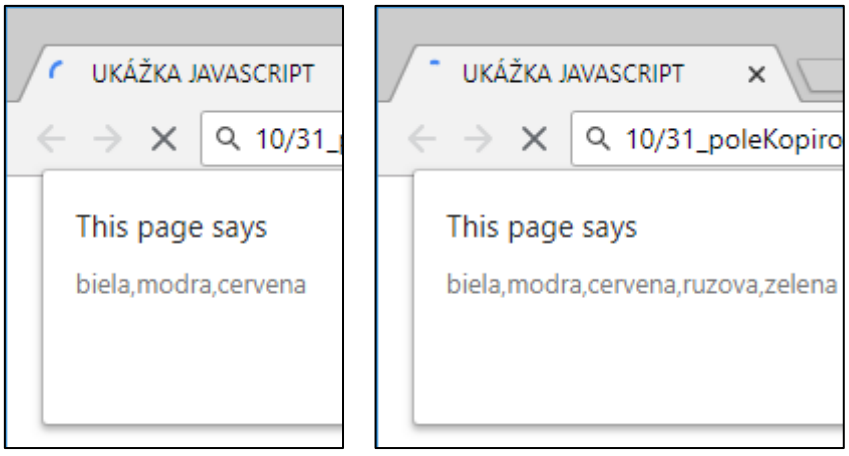
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena`.

Vykonajte nasledujúce operácie s poľom:

- vytvorte nové pole s názvom `farby1` a nakopírujte doňho všetky prvky z poľa farby, pole vypíšte v dialógovom okne,
- do poľa `farby1` a nakopírujte všetky prvky z poľa `farby` a súčasne pridajte nové prvky `ruzova, zelena` , pole vypíšte v dialógovom okne.

```
//vytvorenie poľa a inicializácia troch prvkov poľa
var farby = new Array("biela", "modra", "cervena");

//kopírovanie prvkov poľa do nového poľa
var farby1 = farby.concat();
alert(farby1);

//kopírovanie prvkov poľa do nového poľa a doplnenie nových prvkov
farby1 = farby.concat("ruzova", "zelena");
alert(farby1);
```

Tento kód zobrazí nasledujúce dialógové okná:



- **slice()** – metóda na kopírovanie poľa, pričom prvý argument určuje začiatočnú pozíciu a druhý argument konečnú pozíciu. Ak je v parametri iba jeden argument, tak skopíruje prvky od tejto pozície do konca poľa.

10/32_poleVysek.html

PRÍKLAD 10.32

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

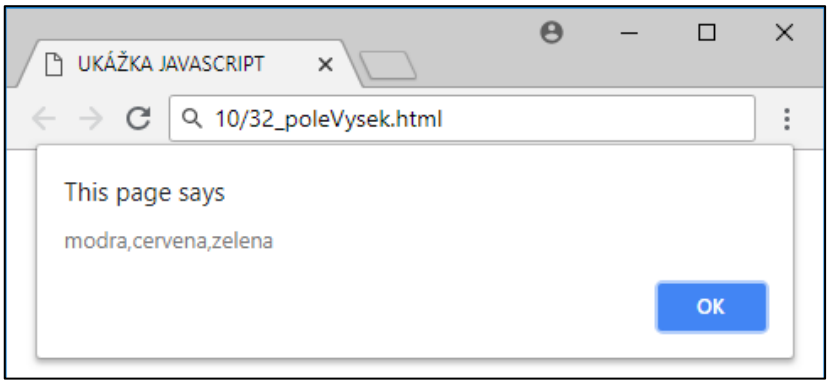
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena, zelena, ruzova`.

Vykonajte nasledujúce operácie s poľom:

- vytvorte nové pole s názvom `farby1` a nakopírujte doňho všetky prvky z poľa `farby` od indexu `1`, pole vypíšte v dialógovom okne,
- do poľa `farby1` nakopírujte hodnoty z poľa `farby` od indexu `1` do indexu `3` (súčasne aj prvok s indexom `3`).

```
//vytvorenie poľa a inicializácia troch prvkov poľa
var farby = new Array("biela", "modra", "cervena", "zelena", "ruzova");

//kopírovanie vybranej časti prvkov poľa do nového poľa počnúc prvkom s indexom 1
var farby1 = farby.slice(1);
alert(farby1);

//kopírovanie vybranej časti prvkov poľa (indexy 1 - 3) do nového poľa
farby1 = farby.slice(1, 4);
alert(farby1);
```

Tento kód zobrazí nasledujúce dialógové okno:



- **splice()** – vkladanie prvkov do stredu poľa. Prvý argument určuje počiatočnú pozíciu, druhý argument koľko prvkov sa má vymazať a tretí argument, čo sa má vložiť (môžeme vložiť aj viacej prvkov).

POZNÁMKA

Funkcia `splice()` sa môže použiť aj na mazanie prvkov z poľa – uvedieme iba prvé dva argumenty.

PRÍKLAD 10.33

10/33_poleVkladanie.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

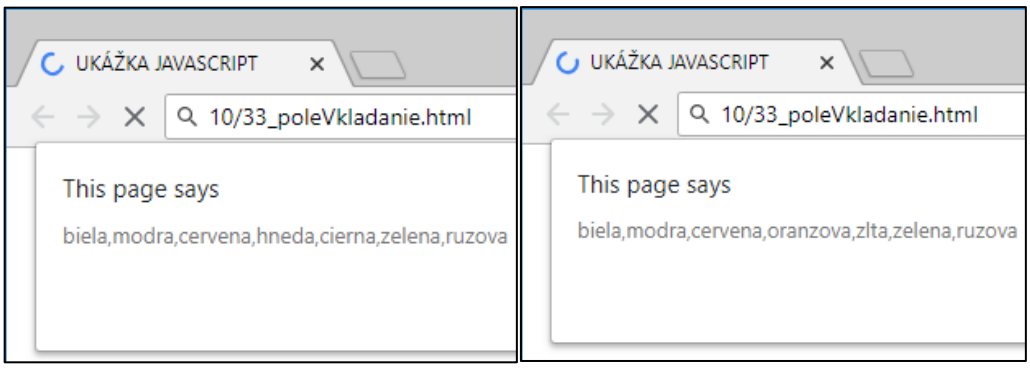
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena, zelena, ruzova`.

Vykonajte nasledujúce operácie s poľom:

- upravte pole farby tak, že vložíte `2` nové prvky do poľa od pozície `3`. Nové pole vypíšte v dialógovom okne,
- upravte pole farby tak, že vymeníte `2` prvky od pozície `3`. Nové pole vypíšte v dialógovom okne,
- upravte pole farby tak, že vymažete `2` nové prvky od pozície `3`. Nové pole vypíšte v dialógovom okne.

```
//vytvorenie poľa a inicializácia troch prvkov poľa
var farby = new Array("biela", "modra", "cervena", "zelena", "ruzova");

//vloženie 2 prvkov od pozície 3
 farby.splice(3, 0, "hneda", "cierna");
alert(farby);

//vymena 2 prvkov od pozície 3
farby.splice(3, 2, "oranzova", "zlta");
alert(farby);

//vymazanie 2 prvkov od pozície 3
farby.splice(3, 2);
alert(farby);
```

Tento kód zobrazí nasledujúce dialógové okná:



10.10 Metodika pre učiteľa

CIEĽ

Cieľom je oboznámiť študenta so základnými údajovými typmi používanými v JavaScripte a bežne používanými operáciami s týmito údajmi. Následne naučiť študenta, ako vykonať vetvenie pomocou a opakovanie časti programu pomocou rôznych druhov cyklov, ako vytvoriť pole, pridávať, odoberať a meniť jeho prvky a ako zadefinovať a zavolať vlastnú funkciu.

MOTIVÁCIA

Študent dokáže vytvoriť jednoduché programy v jazyku JavaScript a následne ich spúšťať a zobraziť si výsledok pomocou dialógové okna vo webovom prehliadači.

VÝKLAD

Študentov treba oboznámiť:

- so základnou syntaxou jazyka JavaScript,
- s premennými,
- s kľúčovými slovami jazyka,
- akým spôsobom je možné vytvoriť komentár a aký je jeho význam,
- s akými údajovými typmi môže pracovať,
- s operátormi (aritmetické, relačné, porovnávacie, logické, priraďovacie),
- ako ovplyvňovať beh programu pomocou vetvenia a cyklov,
- s významom a spôsobom vytvárania vlastných funkcií a ich použitia,
- ako pracovať s poľom údajov.

Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch.

11 JAVASCRIPT – OBJEKTOVÝ MODEL DOKUMENTU

Objektový model dokumentu (skrátene DOM – Document Object Model) sa vytvorí po načítaní internetovej stránky. DOM si môžeme predstaviť ako strom objektov. Pomocou DOM môžeme vytvárať dynamické HTML:

- meniť, pridávať, odstraňovať HTML element, atribúty, udalosti,
- pristupovať k CSS štýlom.

11.1 Objekt Document

V objektovom modely dokumentu reprezentujú HTML elementy objekty. Objekt `document` predstavuje internetovú stránku. Pomocou objektu `document` môžeme pristupovať k jednotlivým HTML elementom. Keď chceme pristúpiť k nejakému HTML element musíme zavolať niektorú z nasledujúcich metód na objekte `document`:

Metóda	Popis
`document.getElementById(id)`	Nájde element s príslušným id.
`document.getElementsByTagName(name)`	Nájde element pomocou tagu name.
`document.getElementsByClassName(name)`	Nájde element pomocou class name.

Samotné metódy však neupravia obsah HTML elementov. Na upravenie elementov musíme ešte zavolať vlastnosti, podľa toho, čo chceme upravovať.

Na pridanie alebo zmenenie textu musíme zavolať vlastnosť `innerHTML`:

```
document.getElementById("nazovId").innerHTML= "text, ktorý sa
zobrazí";
```

Metóda getElementById(id)

Na nájdenie HTML elementu, ktorý obsahuje atribút `id` v DOM môžeme použiť metódu `getElementById(id)`.

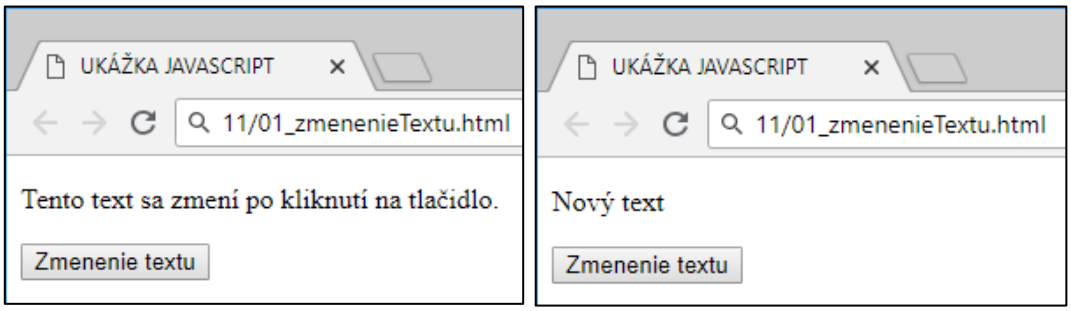
PRÍKLAD 11.1

11/01_zmenenieTextu.html

Vytvorte stránku, ktorá bude obsahovať paragraf s atribútom `id` `text` a tlačidlo `Zmenenie textu.` Po stlačení tlačidla sa zmení text v paragrafe s atribútom `id` `text`.

```
<p id="text">Tento text sa zmení po kliknutí na tlačidlo.</p>

<button type="button"
onclick='document.getElementById("text").innerHTML = "Nový
text"'>Zmenenie textu</button>
```

Tento kód zobrazí stránku:



Pomocou vlastnosti style môžeme zmeniť CSS štýl:

```
document.getElementById("nazovId").style.vlastnost= "nova
hodnota";
```

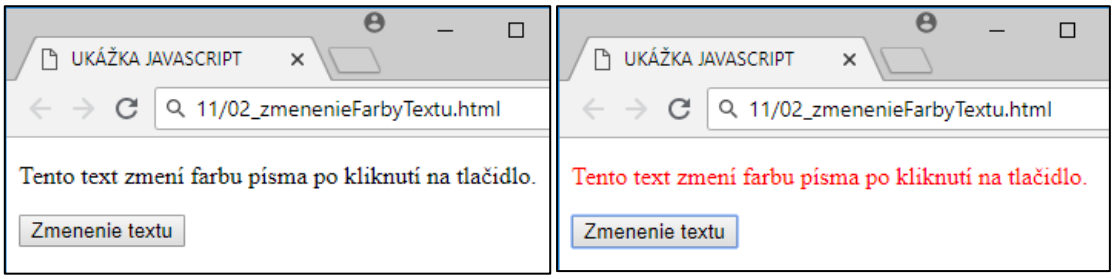
11/02_zmenenieFarbyTextu.html

PRÍKLAD 11.2

Otvorte príklad 11.1 a upravte ho tak, aby sa po stlačení tlačidla `Zmenenie textu` zmenila farba v paragrafe, ktorý má nastavený atribút `id` `text`.

```
<p id="text">Tento text zmení farbu písma po
kliknutí na tlačidlo.</p>

<button type="button"
onclick="document.getElementById('text').style.color='red'">
Zmenenie textu </button>
```

Tento kód zobrazí stránku:



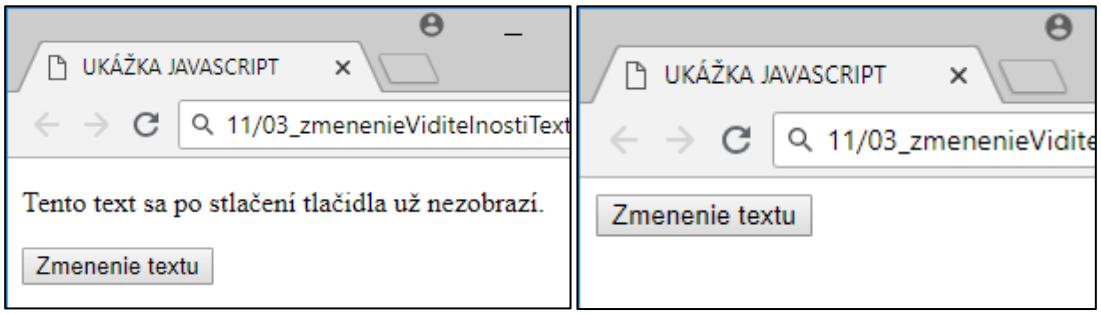
PRÍKLAD 11.3

Otvorte príklad 11.1 a upravte ho tak, aby sa po stlačení tlačidla `Zmenenie textu` nastavil paragraf s atribútom `id` `text` na neviditeľný.

11/03_zmenenieViditelnostiTextu.html

```
<p id="text">Tento text sa po stlačení tlačidla už nezobrazí.</p>

<button type="button"
onclick="document.getElementById('text').style.display='none'">Zmenenie textu</button>
```

Tento kód zobrazí stránku:



Zmenenie hodnoty atribútu:

```
document.getElementById('id').attribute='nova hodnota';
```

Zmeniť obrázok môžeme pomocou vlastnosti `src`:

```
document.getElementById('obrazokId').src='novy_obrazok.jpg';
```

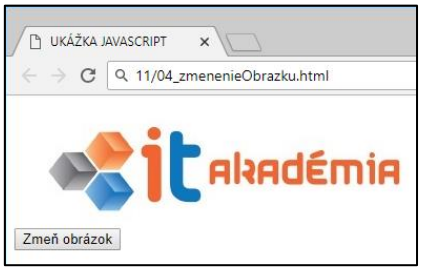
PRÍKLAD 11.4

Vytvorte stránku, ktorá bude obsahovať obrázok s `id` `mojObrazok`, obrázok bude mať nastavený zdroj `obrazok1.jpg`. Pod obrázkom vytvorte tlačidlo `Zmeň obrázok.` Po stlačení tlačidla sa zmení zdroj obrázku na `obrazok2.jpg`.

11/04_zmenenieObrazku.html

```
<div>
<img id="mojObrazok" src="obrazok1.jpg">
</div>

<button
onclick="document.getElementById('mojObrazok').src=obrazok2.jpg'
">Zmeň obrázok</button>
```

Tento kód zobrazí stránku:



Metóda getElementsByTagName(name)

Na nájdenie HTML elementu podľa značky v DOM môžeme použiť metódu `getElementsByTagName(name)`. Nesmieme však zabudnúť nato, že značiek môže byť v HTML stránke ľubovoľný počet. Ak chceme zvoliť konkrétnu značku, musíme za volaním metódy uviesť hranaté zátvorky, do ktorých vložíme index (poradie) značky, s ktorou chceme pracovať.

Ak chceme napríklad pridať, alebo zmeniť text v prvom paragrafe, musíme použiť syntax:

```
document.getElementsByTagName("p")[0].innerHTML = "novy text";
```

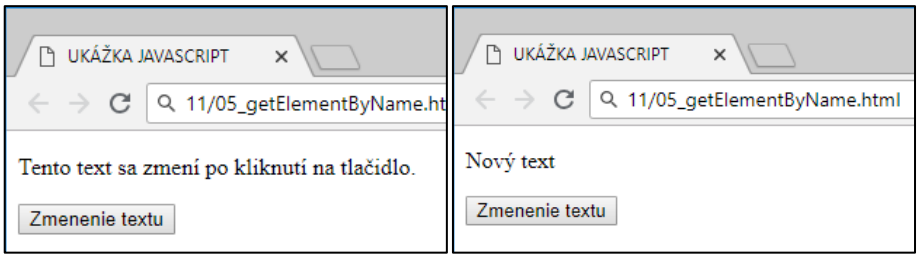
11/05_getElementByTagName.html

PRÍKLAD 11.5

Otvorte príklad 11.1 a upravte ho tak, aby ste namiesto metódy `getElementById()` použili metódu `getElementsByTagName()`.

```
<p>Tento text sa zmení po kliknutí na tlačidlo.</p>

<button type="button"
onclick='document.getElementsByTagName("p")[0].innerHTML = "Nový
text"'>Zmenenie textu</button>
```

Tento kód zobrazí stránku:



Metóda getElementsByClassName(class)

Na nájdenie HTML elementu/elementov s atribútom `class` v DOM môžeme použiť metódu `getElementsByClassName(class)`. Nesmieme však zabudnúť nato, že takýchto značiek môže byť veľa. Ak chceme zvoliť konkrétnu značku s atribútom `class`, musíme za volaním

metódy uviesť hranaté zátvorky, do ktorých vložíme index (poradie) značky, s ktorou chceme pracovať.

Ak chceme napr. pridať, alebo zmeniť text v prvom paragrafe, musíme použiť syntax:

```
document.getElementsByClassName("text1")[0].innerHTML = "novy
text";
```

PRÍKLAD 11.6

11/06_getElementsByClassName.html

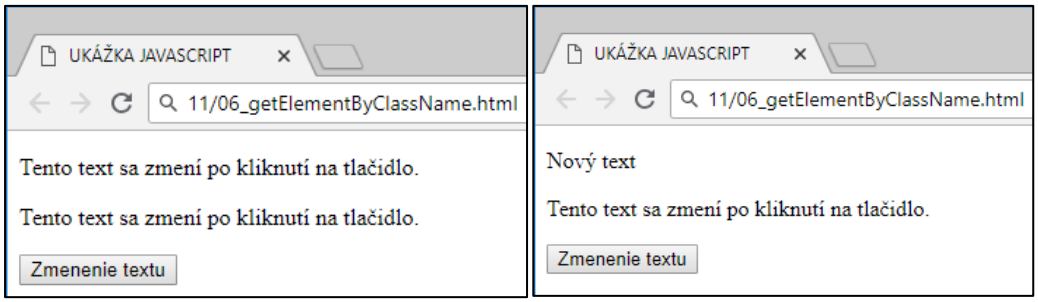
Vytvorte stránku, ktorá bude obsahovať:

- Paragraf s atribútom `class="text"` a textom `Tento text sa zmení po kliknutí na tlačidlo.`
- Paragraf s atribútom `class="text2"` a textom `Tento text sa zmení po kliknutí na tlačidlo.`
- tlačidlo `Zmenenie textu.` Po stlačení tlačidla sa zmení text s atribútom `class="text"` na text: `Nový text`

Poznámka: Na riešenie použite metódu `getElementsByClassName(class).`

```
<p class="text">Tento text sa zmení po kliknutí na
tlačidlo.</p>

<p class="text2">Tento text sa zmení po kliknutí na
tlačidlo.</p>


<button type="button"
onclick='document.getElementsByClassName("text")[0].innerHTML =
"Nový text"'>Zmenenie textu</button>
```

Tento kód zobrazí stránku:



Metodika pre učiteľa

CIEĽ

Cieľom je oboznámiť študenta s objektovým modelom dokumentu. Naučí sa, ako pristupovať k HTML elementom a modifikovať ich vlastnosti, ako aj dynamicky ich pridávať a odoberať z webovej stránky.

MOTIVÁCIA

Študent dokáže vytvoriť jednoduchú webovú stránku a dynamicky upravovať jej obsah. Výsledok si dokáže overiť pomocou webového prehliadača.

VÝKLAD

Študentov treba oboznámiť:

- s objektovým modelom dokumentu,
- ako vyhľadať požadovaný element dokumentu,
- ako modifikovať jeho vlastnosti.

Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch.

12 JAVASCRIPT – UDALOSTI

Interakcia jazyka JavaScript s kódom jazyka HTML prebieha prostredníctvom udalostí (z angl. *events*). Udalosti patria medzi najdôležitejšiu časť jazyka JavaScript, nakoľko umožňujú reagovať na podnety používateľa. Udalosti priraďujeme pomocou atribútu jazyka HTML (napr. atribút `onclick`) s názvom konkrétnej udalosti (špecifikuje programátor, napr. stlačenie tlačidla). Syntax je nasledovná

```
<htmlElement event="JavaScript kód">
```

Najpoužívanejšou udalosťou je udalosť `onclick`, ktorá sa vyvolá po kliknutí na HTML značku, ktorá obsahuje tento atribút.

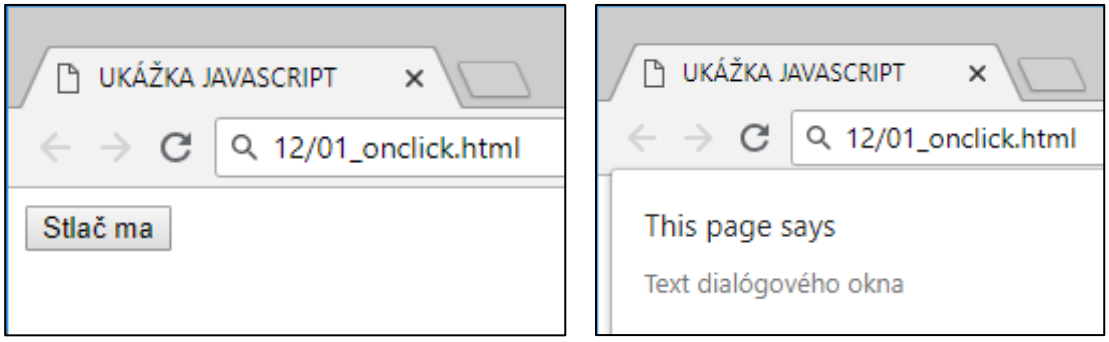
PRÍKLAD 12.01

12/01_onclick.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `body` vytvorte tlačidlo pomocou značky `<button>` , tlačidlu pridajte atribút `type` s hodnotou `button` a atribút `onclick` s hodnotou `alert('Text dialógového okna')`.

```
<button type="button" onclick="zobrazDialogoveOkno()"> Stlač
ma</button>

<script>
function zobrazDialogoveOkno(){
        //kód, ktorý sa má vykonať
        alert("Text dialógového okna");
}
</script>
```

Tento kód zobrazí stránku:



ZAPAMÄTAJTE SI

V hodnote atribútu `onclick` nemôžeme používať syntaktické znaky jazyka HTML (napr. dvojité úvodzovky) bez ich zakódovania. Z tohto dôvodu sme namiesto dvojitých úvodzoviek

použili apostrofy. Alternatívou k apostrofom je použitie HTML entít (v prípade dvojitých úvodzoviek je nutné použiť HTML entitu `"`):

```
<button type="button" onclick="alert(&quot;Text dialógového
okna&quot;)"> Stlač ma</button>
```

12.1 HTML typy udalostí

Udalosti, vyvolané v HTML stránke môžu nastať:

- vyvolaním používateľom napr. stlačenie tlačidla, kliknutím na objekt,
- zmenením hodnoty HTML značky napr. hodnoty elementu `input`,
- po dokončení načítavania HTML stránky.

Najpoužívanejšie udalosti:

Udalosť	**Popis**
onchange	HTML element sa zmení
onclick	Kliknutie na event
onmouseover	Presunutie kurzora na element
onmouseout	Presunutie kurzora preč z elementu
onkeydown	Stlačenie klávesnice
onload	dokončenie načítavania HTML súboru

Niektoré udalosti môžu byť priradené aj statickým HTML značkám, ako sú napríklad značky pre nadpis (`h1...h7`), značky pre zobrazenie textu (`div, p, span`), atď.

```
<h1 onclick="alert('klikol si na nadpis')">Nadpis 1</h1>

<div onclick="alert('klikol si na text)">Text …</div>
```

13/02_prihlaseni.html

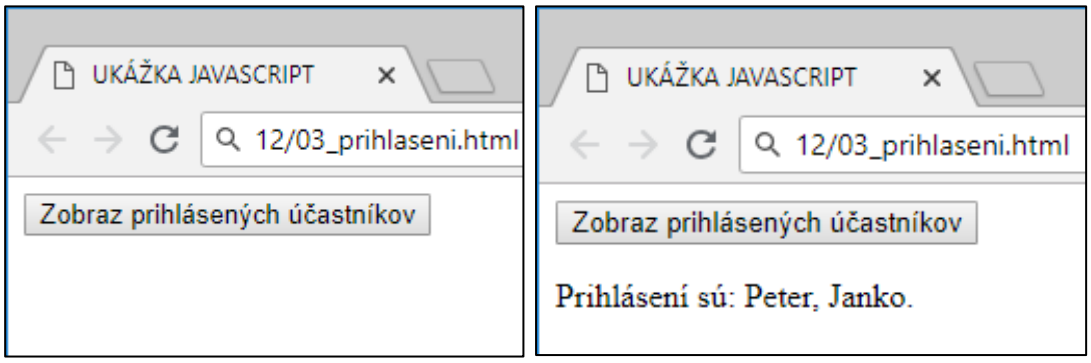
PRÍKLAD 13.02

Vytvorte stránku, v ktorej bude tlačidlo s textom `Vypíš text`. Po stlačení tlačidla vypíše pod tlačidlo text `Prihlásení sú: Peter, Janko`.

Poznámka: text `Prihlásení sú: Peter, Janko`. Nevypisujte do dialógového okna `alert()`, ale priamo do textu. Pomôcka – použite metódu `getElementById()`.

```
<button onclick="zobrazPrihlasenych()">Zobraz prihlásených
účastníkov</button>

<p id="text"></p>

<script>
function zobrazPrihlasenych()
{
      document.getElementById("text").innerHTML = "Prihlásení sú:
      Peter, Janko.";
} </script>
```

Tento kód zobrazí stránku:



ÚLOHA 12.1

Otvorte súbor *12/03_prihlaseni.html*. Upravte ho tak, aby vypisoval mená prihlásených účastníkov pod sebou.

12.1.1 Udalosť onchange

Udalosť `onchange` sa zavolá v prípade, ak sa zmenila hodnota nejakého elementu. V prípade zaškrtávacieho tlačidla, alebo výberu z jednej možnosti sa táto funkcia zavolá, keď sa zmení stav `checked`.

PRÍKLAD 13.03

13/03_onchange.html

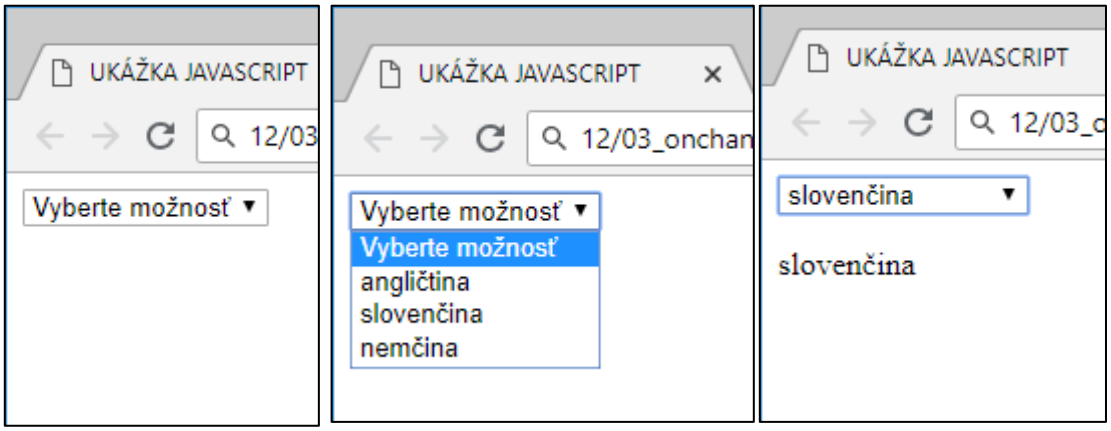
Vytvorte stránku, v ktorej bude vysúvací list `<select>` s hodnotami:

- vyberte možnosť,
- angličtina,
- slovenčina,
- nemčina.

Po vybratí jazyka sa vybraná možnosť uloží do paragrafu s `id` hodnotou `text`.

```
<select id="jazyk" onchange="vypisVybranuHodnotu()">
     <option value="vyberte možnosť">Vyberte možnosť</option>
     <option value="angličtina"> angličtina</option>
     <option value="slovenčina"> slovenčina</option>
     <option value="nemčina"> nemčina</option>
</select>


<p id="text"></p>

<script>
function vypisVybranuHodnotu()
{
     document.getElementById("text").innerHTML =
     document.getElementById("jazyk").value;
}
</script>
```

Tento kód zobrazí stránku:



12.1.2 Udalosť onmouseover a onmouseout

Udalosť `onmouseover` sa vyvolá, keď sa kurzor myši posunie na element, ktorý má nastavený tento atribút. Upozornenie, na vyvolanie tejto udalosti je potrebné, aby sa kurzor myši predtým nenachádzal na tomto elemente.

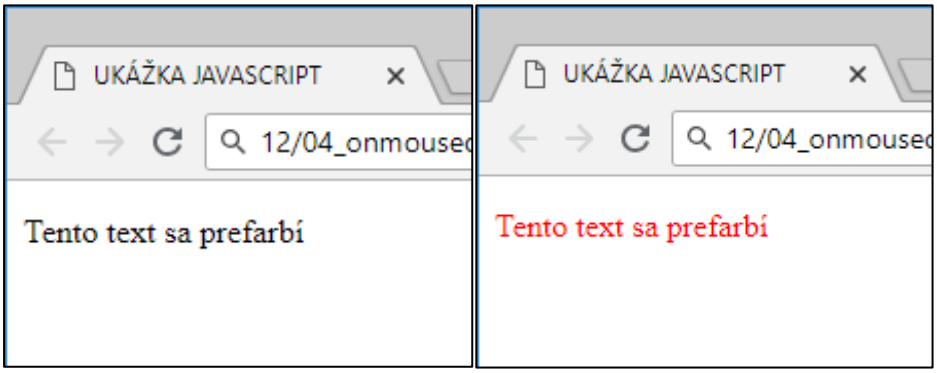
Udalosť `onmouseout` sa spustí, keď sa kurzor myši presunie preč z elementu, ktorý má nastavený tento atribút na iný element.

13/04_onmouseover.html

PRÍKLAD 13.04

Vytvorte stránku, ktorá bude obsahovať jeden paragraf s textom `Tento text sa prefarbí.` Paragrafu pridajte atribúty `onmouseover` a `onmouseout`. Každý z týchto atribútov zavolá vlastnú funkciu, ktorá prefarbí text. Udalosť `onmouseover` prefarbí text paragrafu na červeno. Udalosť `onmouseout` prefarbí text na čierno.

```
<p onmouseover="farba()" onmouseout="farba1()" id="text">Tento
text sa prefarbí</p>

<script>
function farba()
{
     document.getElementById('text').style.color = "red";
}

function farba1()
{
      document.getElementById('text').style.color = "black";
}
</script>
```



12.1.3 Udalosť onkeydown

Udalosť `onkeydown` sa vyvolá po stlačení tlačidla v elemente, ktorý obsahuje tento atribút. Zvyčajne sa táto udalosť pridáva html elementom `textarea` a `input`.

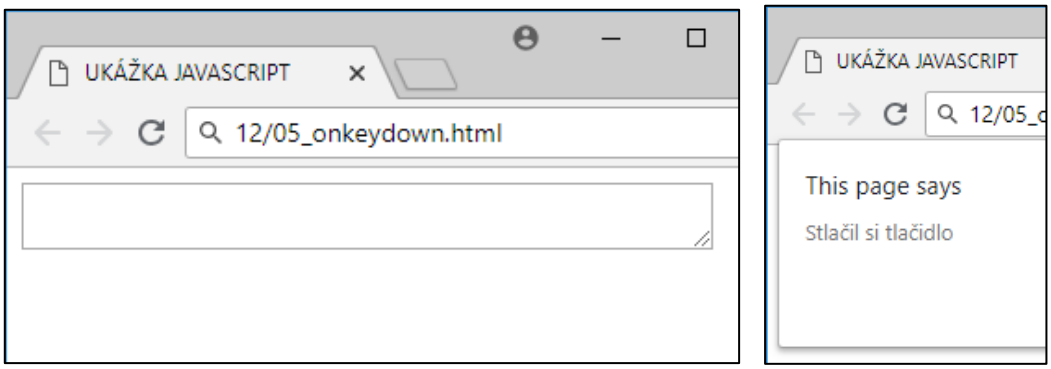
PRÍKLAD 13.05

Vytvorte stránku, ktorá bude obsahovať viacriadkové textové pole, ktorému pridajte atribút `onkeydown` s hodnotou `zobrazDialog()`. V časti `script` vytvorte funkciu `zobrazDialog()`, v ktorej zavolajte dialógové okno `alert()` s textom `Stlačil si tlačidlo`.

13/05_onkeydown.html

```
<textarea rows="2" cols="50" onkeydown="zobrazDialog()">
</textarea>

<script>
function zobrazDialog() {
    alert("Stlačil si tlačidlo");
}
</script>
```

Tento kód zobrazí stránku:



12.1.4 Udalosť onload

Udalosť `onload` sa zavolá po načítaní celej stránky alebo objektu (načítané musia byť všetky externé súbory, obrázky, atď.). Najčastejšie sa táto udalosť používa s HTML elementom `<body>`.

13/06_onload .html

PRÍKLAD 13.06

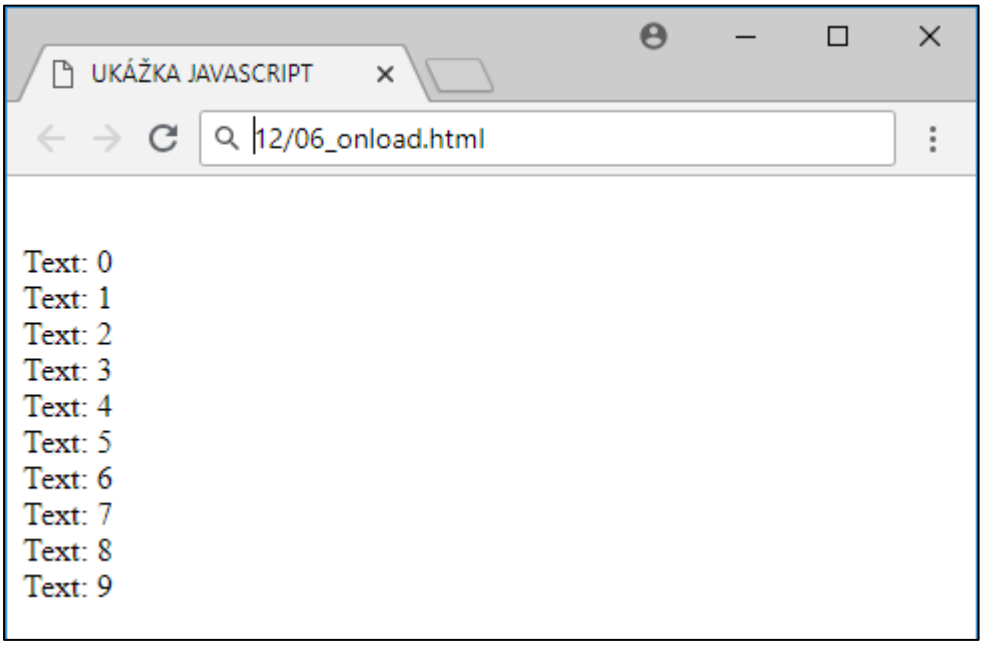
Vytvorte stránku, v ktorej ku značke `<body>` pridajte atribút `onload` s hodnotou zavolaním funkcie `vypisText()`. Do stránky pridajte paragraf s *id text*. V časti `script` vytvorte funkciu `vypisText()`. V tejto funkcii vypíšte 10-krát pod sebou text:

```
Text: 0
Text: 1
Text: 2
Text: 3
Text: 4
Text: 5
Text: 6
Text: 7
Text: 8
Text: 9
```

Poznámka: na riešenie použite cyklus `for`.

```
<body onload="vypisText()">

<p id="text1"></p>

<script>
function vypisText() {
      for(var i = 0; i < 10; i++) {
      document.getElementById("text1").innerHTML += "<br> Text: "
+ i;
    }
}
</script>

</body>
```

Tento kód zobrazí stránku:



12.2 Metodika pre učiteľa

CIEĽ

Cieľom je oboznámiť študenta pojmom udalosť. Naučiť študenta zadefinovať akciu, ktorá sa vykoná pri výskyte udalosti.

MOTIVÁCIA

Študent dokáže vytvoriť jednoduchú webovú stránku a interaktívne dynamicky upravovať jej obsah. Výsledok si dokáže overiť pomocou webového prehliadača.

VÝKLAD

Študentov treba oboznámiť s udalosťami:

- onchange,
- onclick,
- onmouseover,
- onmouseout,
- onkeydown,
- onload.

Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch.

13 JAVASCRIPT – FORMULÁRE

Jedným z dôvodov vytvorenia jazyka JavaScript bolo preniesť spracovávanie formulárov zo servera na prehliadače. Pomocou jazyka JavaScript môžeme napríklad overovať vyplnené dáta, rozhodovať o odoslaní formulára, atď.

Formuláre v jazyku HTML vytvárame pomocou párovej značky `<form>…</form>`. Vo formulároch môžeme použiť ľubovoľný počet komponentov. Medzi najpoužívanejšie komponenty patria elementy:

Názov značky	**Popis**
input	podľa hodnoty atribútu type, môžeme značku input použiť ako textové pole, prepínacie tlačidlá, tlačidlá odoslať, resetuj
select	list s hodnotami, ktorý po kliknutí vysunie všetky možnosti
textarea	viacriadkové textové pole
button	tlačidlo

PRÍKLAD 13.01

13/01_formular1.html

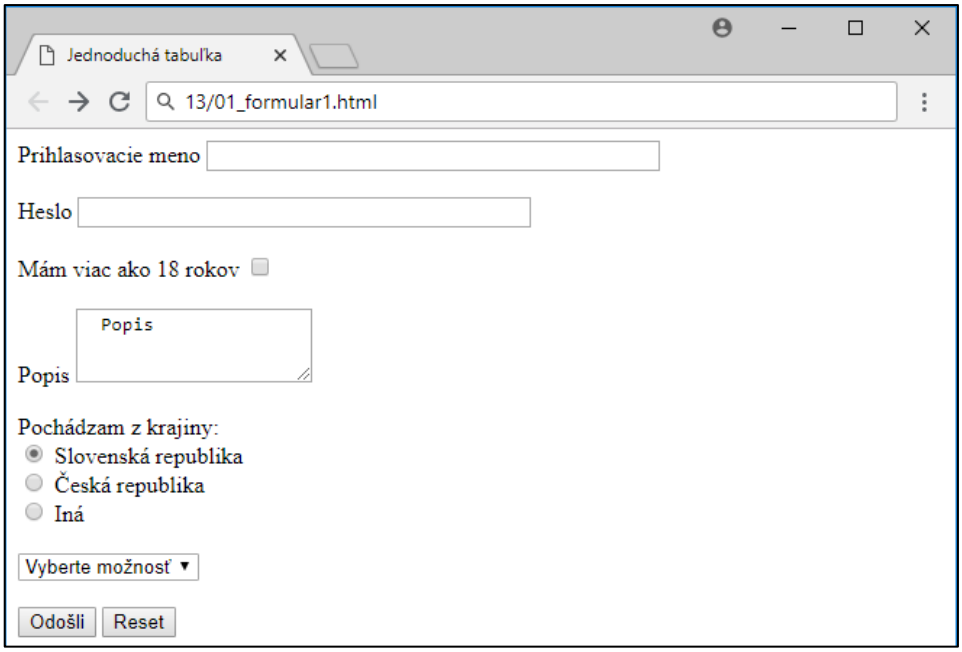
Vytvorte HTML stránku so základnou štruktúrou. V časti `body` vytvorte formulár, ktorý bude obsahovať:

- textové pole pre prihlasovacie meno,
- textové pole pre heslo (znaky, ktoré tvoria heslo sa nesmú zobrazovať),
- zaškrtávacie tlačidlo s textom `Mám viac ako 18 rokov`,
- popis – viacriadkové textové pole,
- možnosť vybrať hodnotu z ponúkaných možností (Radio buttons),
- výber z jednej možnosti (vysúvací list),
- tlačidlo `Odošli`, ktoré zatiaľ nevykoná nič,
- tlačidlo `Reset`, ktoré vynuluje vyplnené hodnoty.

Poznámka: pre jednoduchosť použite na formátovanie HTML značiek značku `<br>`. Správne formátovanie by však malo byť pomocou CSS štýlov.

```
<form name="formular">

Prihlasovacie meno
<input type="text" name="prihlasovacieMeno">
<br><br>

Heslo
<input type="password" name="heslo">
<br><br>

Mám viac ako 18 rokov
<input type="checkbox" name="pohlavie" value="Som muž" >
<br><br>

Popis
<textarea name="popis" rows="3" colls="3">Popis</textarea>

Pochádzam z krajiny: <br>
<input type="radio" name="krajina" value="Slovenská republika"
checked> Slovenská republika<br>
<input type="radio" name="krajina" value="Česká republika">
Česká republika<br>
<input type="radio" name="krajina" value="iná"> Iná
<br><br>

<select name="jazyk">
     <option value="0"> Vyberte možnosť</option>
      <option value="1"> angličtina</option>
      <option value="2"> slovenčina</option>
      <option value="3"> nemčina</option>
</select>
<br><br>

<input type="submit" value="Odošli">
<input type="reset" value="Reset">
</form>
```

Tento kód zobrazí stránku:



13.1 Odosielanie formulárov

Formuláre sa odosielajú po stlačení tlačidla, ktoré sa definuje pomocou značky:

- `input` s atribútom *type* a hodnotou `submit`

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

- `button` s atribútom `type` a hodnotou `submit`

```
<button type="submit">Odošli</button>
```

13.2 Resetovanie formulárov

Formuláre sa resetujú po stlačení tlačidla, ktoré sa definuje pomocou značky:

- `input` s atribútom `type` a hodnotou `reset`

```
<input type="reset" value="Vymaž hodnoty formulára">
```

- `button` s atribútom `type` a hodnotou `reset`

```
<button type="submit">Vymaž hodnoty formulára</button>
```

Pri resetovaní formulára sa odstránia všetky hodnoty, ktoré zadal používateľ do formulárových prvkov. Ak bolo nejaké pole prázdne, tak bude opäť prázdne. V prípade, že malo nejaké pole nastavenú predvolenú hodnotu, ktorú používateľ zmenil, tak táto hodnota sa po stlačení tlačidla s atribútom *type reset* nastaví na predvolenú hodnotu.

13.3 Spracovanie formulára

Stlačenie tlačidla s atribútom type `submit` odošle formulár, ale nesmieme zabudnúť nastaviť akciu, ktorá sa má vykonať po odoslaní formulára. Akcia, ktorá sa má vykonať sa nastavuje prostredníctvom atribútu `action` značky `form`. Ako hodnotu atribútu môžeme nastaviť napríklad súbor php, ktorý spracuje formulárové hodnoty.

```
<form action="spracovanieHodnot.php">
…
</form>
```

13.3.1 Validácia formulárov pomocou jazyka JavaScript

Jednou z možností ako skontrolovať, či používateľ vyplnil formulárové prvky, ktoré potrebujeme mať uložené je prostredníctvom jazyka JavaScript. Pomocou jazyka JavaScript vieme tiež skontrolovať, či používateľ vyplnil správne všetky polia (napr. či zadal telefónne číslo

v správnom tvare, či nezabudol dať zavináč pri emailovej adrese). K vlastnostiam HTML pristupujeme pomocou objektového modelu dokumentu. Na výber máme dve možnosti, ako pristupovať k jednotlivým prvkom:

1) Pomocou atribútov `name` môžeme pristupovať k jednotlivým prvkom

```
document.nazov_formulara.nazov_input_polozky.value
```

Ak chceme pristúpiť k hodnote, ktorú vyplnil používateľ v textovom poli, ktorý má atribút `name="prihlasovacieMeno"` a súčasne formulár má atribút `name="form"`:

```
document.form.prihlasovacieMeno.value
```

13/02_formular2.html

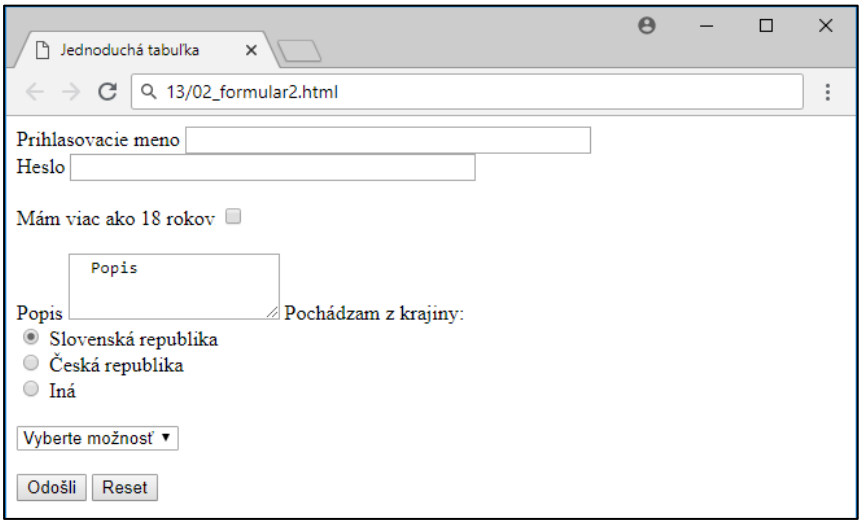
PRÍKLAD 13.02

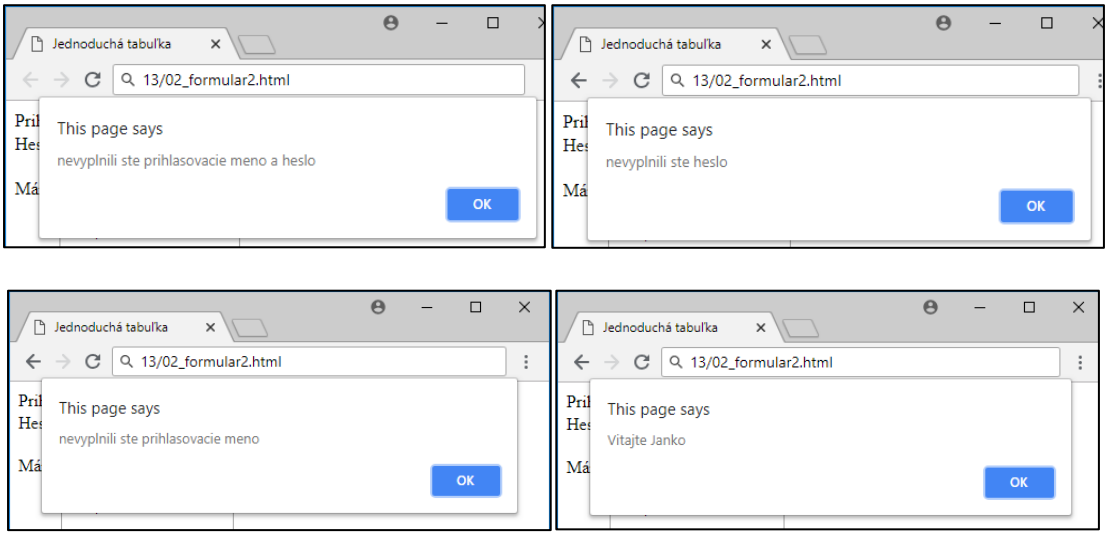
Otvorte si predchádzajúci formulár 13.1 (súbor *13/01_formular1.html*). Do stránky vložte párovú značku `script`, do ktorej pridajte validáciu, či sú zadané hodnoty prihlasovacie meno a heslo:

- Ak nie je vyplnené prihlasovacie meno, tak vypíš v dialógovom okne upozornenie `nevyplnili ste prihlasovacie meno`.
- Ak nie je vyplnené heslo, tak vypíš v dialógovom okne upozornenie `nevyplnili ste heslo`.
- Ak nie je vyplnené prihlasovacie meno a súčasne heslo, tak vypíš v dialógovom okne upozornenie `nevyplnili ste prihlasovacie meno a heslo`.
- Ak je vyplnené prihlasovacie meno a heslo, tak vypíš v dialógovom okne text `Vitajte prihlasovacieMeno`.

```
<script>
function validaciaRegistracie()
{
if (document.formular.prihlasovacieMeno.value == "" &&
document.form.heslo.value == "")
     alert("nevyplnili ste prihlasovacie meno a heslo");
else if (document.formular.prihlasovacieMeno.value == "")
     alert("nevyplnili ste prihlasovacie meno");
else if (document.form.heslo.value == "")
     alert("nevyplnili ste heslo");
else
     alert("Vitajte " + document.form.prihlasovacieMeno.value);
}
</script>

<form>
…
<input type="submit" value="Odošli" onclick="validaciaRegistracie()">
<input type="reset" value="Reset">
</form>
```

Tento kód zobrazí stránku:





2) Ak nezadáme formulárovým prvkom atribúty `name`, tak k jednotlivým prvkom môžeme pristupovať prostredníctvom kolekcie `forms` a `elements`. K jednotlivým poliam kolekcie `elements` pristupujeme prostredníctvom číselného indexu. Začíname od hodnoty `0`.

```
document.forms[0].elements[0].value

document.forms[0].elements[1].value
```

PRÍKLAD 13.03

Otvorte si predchádzajúci formulár 13.2 (súbor *13/02_formular2.html*). Z formuláru odstráňte všetky atribúty `name`. V časti `script` sa odkazujte na formulárové prvky prostredníctvom kolekcie `forms` a `elements`.

13/03_formular3.html

```
function loginForm()
{
if ((document.forms[0].elements[0].value == "") &&
(document.forms[0].elements[1].value == "")) {
     alert("nevyplnili ste prihlasovacie meno a heslo");
}
else if (document.forms[0].elements[0].value == "") {
     alert("nevyplnili ste prihlasovacie meno");
}
else if (document.forms[0].elements[1].value == "") {
     alert("nevyplnili ste heslo");
}
else
     alert("Vitajte " + document.forms[0].elements[0].value);
}
```

POZNÁMKA

Kolekcia `elements` je usporiadaný zoznam odkazov na všetky polia vo formulári. Obsahuje všetky elementy `input, textarea, button`, atď.

Kolekcia `forms` je usporiadaný zoznam odkazov na všetky formuláre. Zvyčajne je však na každej stránke maximálne jeden formulár, takže väčšinou sa na túto kolekciu odkazujeme `forms[0].`

ZAPAMÄTAJTE SI

Validáciu formulára – či používateľ vyplnil všetky povinné prvky môžeme dosiahnuť aj bez jazyka JavaScript. Stačí, ak pridáme atribút `required` k formulárovým prvkom, ktoré chceme, aby boli vyplnené.

```
<input type="text" name="prihlasovacieMeno" required>
```

ÚLOHA 13.1

Vytvorte stránku, do ktorej vložíte formulár, ktorý bude obsahovať:

- textové pole pre prihlasovacie meno,
- textové pole pre heslo,
- textové pole pre opätovné zadanie hesla.

Tlačidlo `Odošli` – ktoré skontroluje, či používateľ vyplnil prihlasovacie meno, heslo a opätovné heslo. Súčasne skontroluje, či sa heslo a opätovné heslo zhodujú. V prípade, že sa nezhodujú, tak vypíše správu v dialógovom okne, že sa heslá nezhodujú.

13.4 Metodika pre učiteľa

CIEĽ

Cieľom je oboznámiť študenta s validáciou formulárov pomocou jazyka JavaScript.

MOTIVÁCIA

Študent dokáže vytvoriť jednoduchú webovú stránku, ktorá bude obsahovať formulár. Študent bude vedieť urobiť validáciu vyplnených položiek pomocou jazyka HTML5 a jazyka JavaScript.

VÝKLAD

Študentom treba pripomenúť:

- ako vytvoriť formulár v HTML,
- HTML značky form, input, select, texarea, button.

Následne študentom treba oboznámiť možnosťami kontroly formulára pred odoslaním pomocou JavaScriptu.

Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch.

14 JAVASCRIPT – TESTOVANIE A LADENIE

Medzi dôležité činnosti pri programovaní patrí aj testovanie. Testovanie môžeme stručne charakterizovať ako proces odhaľovania chýb. Testovanie sa môže robiť buď manuálne, alebo pomocou automatizovaných nástrojov. V tejto kapitole si opíšeme manuálne testovanie.

Bez ohľadu nato, či vyvíjame desktopové aplikácie, mobilné aplikácie alebo webové stránky vždy by sme mali vyvíjaný produkt vyskúšať na rôznych zariadeniach, operačných systémoch a v prípade webových stránok aj na rôznych prehliadačoch.

Medzi najjednoduchšie techniky, ako odstrániť vzniknuté problémy na webových stránkach môžeme postupovať nasledovne:

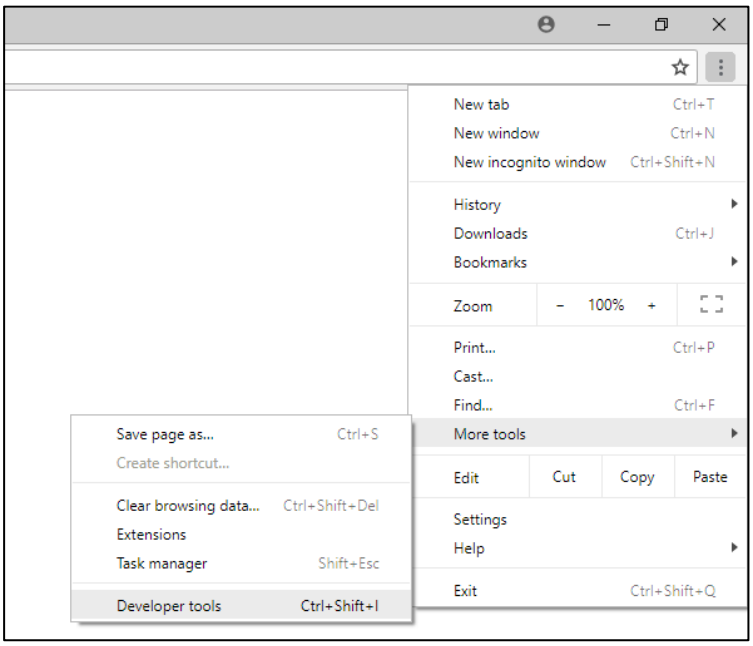
1) Postupovať po častiach – začať kontrolovať jednoduchšie veci a postupne pokračovať kontrolou náročnejších častí. Robiť menšie zmeny v kóde a postupne testovať.
2) Zakomentovať časť kód a postupne odkomentovávať jednotlivé časti, kým sa neobjaví problém.
3) Použiť vývojárske nástroje na preskúmavanie a ladenie zdrojového kódu.

14.1 Vývojárske nástroje pre webové prehliadače

Webové prehliadače majú v súčasnosti implementované vývojárske nástroje. V prípade, že nám tieto nástroje nevyhovujú, môžeme si doinštalovať preferovanejšie nástroje pomocou tzv. doplnkov (pluginov).

Google Chrome

Vývojárske nástroje pre prehliadač Google Chrome sú dostupné v menu prehliadaču → `More tools`“→ `Developer tools`. Pre spustenie nástrojov je tiež možné použiť klávesovú skratku CTRL+SHIFT+I.

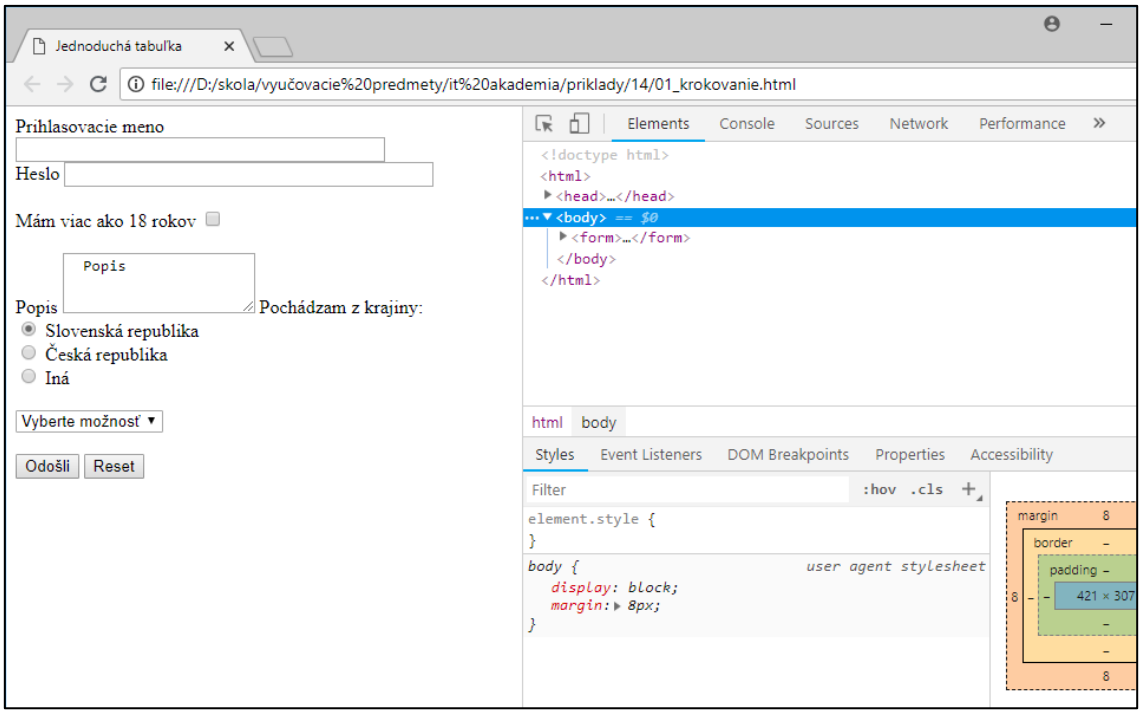


Po spustení vývojárskych nástrojov sa nám z pravej strany vysunie okno, ktoré je vertikálne rozdelené na dve časti (záleží podľa veľkosti obrazovky, môže byť rozdelené aj horizontálne). Prvá časť obsahuje okno so záložkami:

- Elements – pomocou tejto záložky môžeme jednoduchšie identifikovať jednotlivé časti stránky pomocou DOM a CSS,
- Console – slúži na zobrazovanie informácií počas spúšťania stránky,
- Sources – krokovanie kódu JavaScript pomocou tzv. breakpoints,
- Network – informácie o rýchlostiach spúšťania stránky,
- Performance,
- Memory – meranie pamäte,
- Application,
- Security.

Druhá časť obsahuje okno so záložkami:

- CSS štýly,
- JavaScript udalosti,
- DOM body prerušenia,
- Nastavenia,
- Dostupnosť.



Vývojárske nástroje dostupné v prehliadačoch:

Prehliadač	Webový nástroj	Popis
Mozilla Firefox	*Web developer (CTRL + SHIFT + I)*	*https://developer.mozilla.org/son/docs/Tools*
Safari	*Web Inspector (Command + Option + I)*	*https://developer.apple.com/safari/tools/*
Opera	*Vývojár (CTRL+SHIFT+I)*	*https://dev.opera.com/extensions/dev-tools/*
Internet Explorer 11	*Nástroje vývojára (F12)*	*https://msdn.microsoft.com/en-us/library/hh968260(v=vs.85).aspx*
Microsoft Edge	*Vývojárske nástroje (CTRL+SHIFT+I)*	*https://docs.microsoft.com/en-us/microsoft-edge/devtools-guide*

POZNÁMKA

Niektorí programátori preferujú použitie doplnkov – nástrojov pre vývojár od tzv. tretích strán. Napr. v prípade použitia webového prehliadača Firefox sa veľkej popularite teší nástroj Firebug (https://getfirebug.com/).

14.2 Krokovanie kódu

Zdrojový kód môže obsahovať rôzne typy chýb. Nie vždy je jednoduché identifikovať chyby. V niektorých prípadoch sa dokonca nezobrazia ani žiadne chybové hlášky, ktoré by pomohli identifikovať problém. Proces, v ktorom vyhľadávame a opravujeme chyby v zdrojovom kóde nazýva krokovanie (z angl. debugging).

Keď chceme „krokovať“ kód v prehliadačoch, na výber máme viacero možností. Najjednoduchšou možnosťou je volanie funkcie na zobrazovanie dialógového okna. Funkciu `alert()` môžeme vkladať priamo do zdrojového kódu a pomocou parametra tejto funkcie môžeme vypisovať hodnoty premenných, alebo jednoduché správy. V prípade, že sa nám nejaké dialógové okno nezobrazí, vieme, že chybu musíme hľadať ešte pred zavolaním dialógového okna. Takéto riešenie je dosť náročné, pričom nesmieme zabudnúť po identifikovaní chyby odstrániť všetky funkcie `alert()`.

Ďalšou možnosťou je písanie správ do konzoly, ktorú si otvoríme prostredníctvom nástroja vývojára vo webovom prehliadači.

14.2.1 Metóda console.log()

Na vypisovanie hodnôt, správ do konzoly môžeme zavolať metódu `log()` prostredníctvom objektu `console`. Do parametra môžeme vložiť premenné, vlastné správy, kombinovať správy s hodnotami premenných podobne ako pri funkcii `alert()`.

```
var x = 10;
console.log("Hodnota premennej x je:" + x);
```

14/01_krokovanie.html

PRÍKLAD 14.1

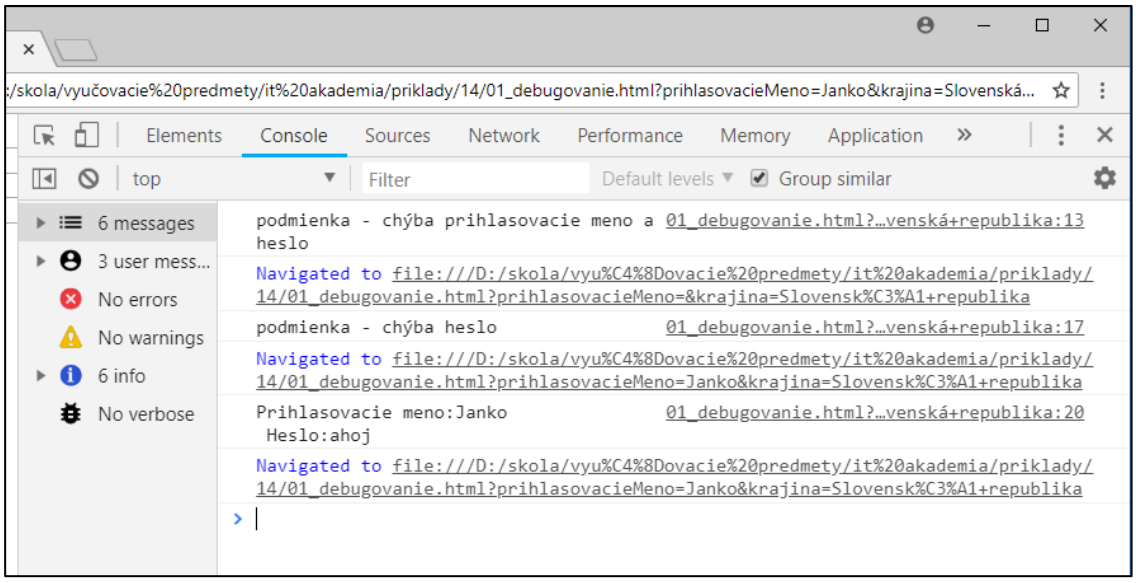
Otvorte si príklad 13.3 (súbor *13/03_formular3.html*). Do formuláru pridajte v časti `<script>` kód, ktorý bude vypisovať do konzoly správy:

- Ak používateľ nezadal meno a heslo, tak vypíše do konzoly text `podmienka - chýba prihlasovacie meno a heslo`.
- Ak používateľ nezadal prihlasovacie meno, tak vypíše do konzoly text `podmienka - chýba prihlasovacie meno`.
- Ak používateľ nezadal prihlasovacie heslo, tak vypíše do konzoly text `podmienka - chýba prihlasovacie heslo`.
- Ak používateľ zadal prihlasovacie meno a heslo, tak vypíše do konzoly text `Prihlasovacie meno: + zadané prihlasovacie meno`.

`Zo zdrojového kódy odstráňte všetky dialógové správy (funkcia alert()).`

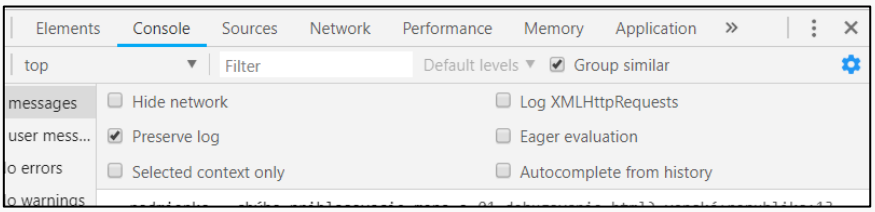
```
<script>
  function loginForm() {
   if ((document.forms[0].elements[0].value == "") &&
     (document.forms[0].elements[1].value == "")) {
      console.log("podmienka - chýba prihlasovacie meno a
      heslo");
   }
   else if (document.forms[0].elements[0].value == "") {
      console.log("podmienka - chýba prihlasovacie meno");
   }
   else if (document.forms[0].elements[1].value == "") {
      console.log("podmienka - chýba heslo");
   }
   else
      console.log("Prihlasovacie meno:" +
      document.forms[0].elements[0].value);
  }
</script>
```

Tento kód zobrazí nasledujúce správy v konzole:



POZNÁMKA

V prípade, že sa text v konzole zobrazí a následne zmizne, tak je potrebné skontrolovať nastavenia pri konzole – zaškrtnúť možnosť `Zachovávať správy` (angl. `preserve logs`.)



Táto možnosť je prednastavená na nezachovávanie správ, nakoľko bežný používateľ internetu nepotrebuje, aby sa mu vypisovali správy do konzoly.

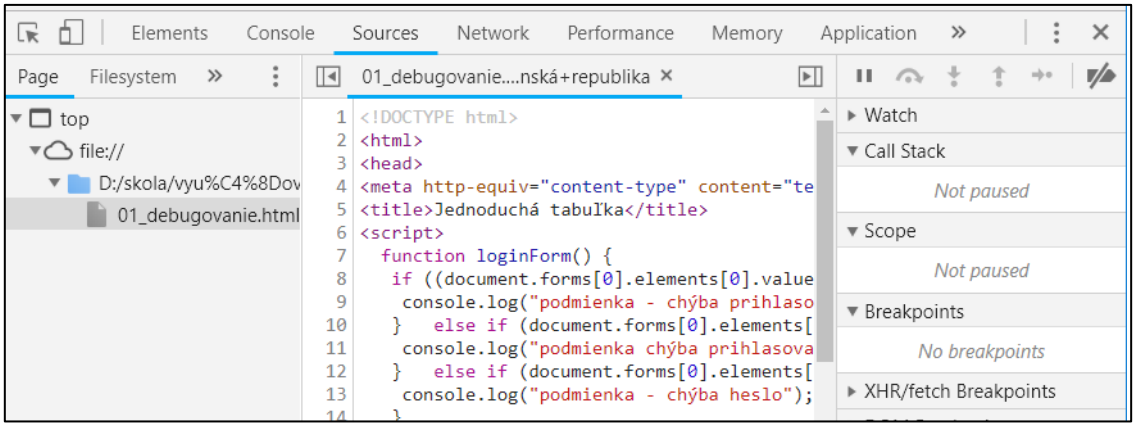
Možnosť vypisovania správ do konzoly je vhodné používať spolu s krokovaním (nastavovaním tzv. bodov prerušenia (breakpoints). V tomto prípade sa správa zobrazí na čas, dokým nestlačíme tlačidlo ukončenie krokovania (prípadne pokračovať na ďalší bod prerušenia).

ÚLOHA 14.1

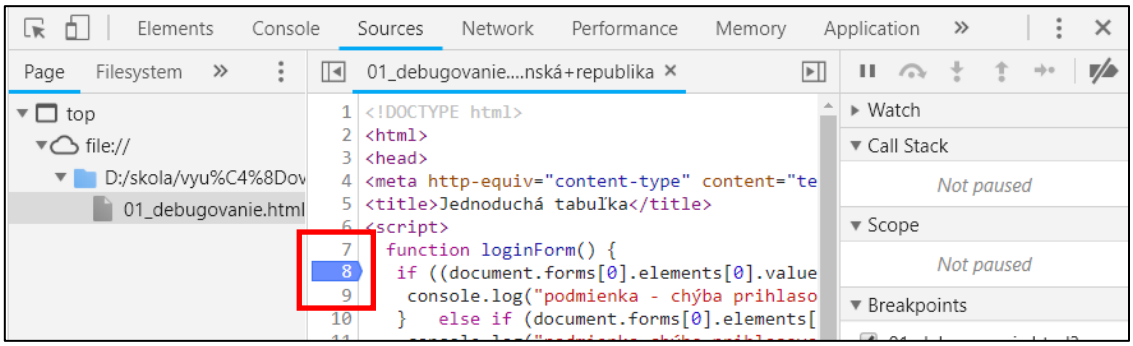
Upravte príklad 13.1 tak, aby ste nahradili všetky funkcie `alert()` výpisom správ do konzoly.

14.2.2 Nastavovanie bodov prerušenia

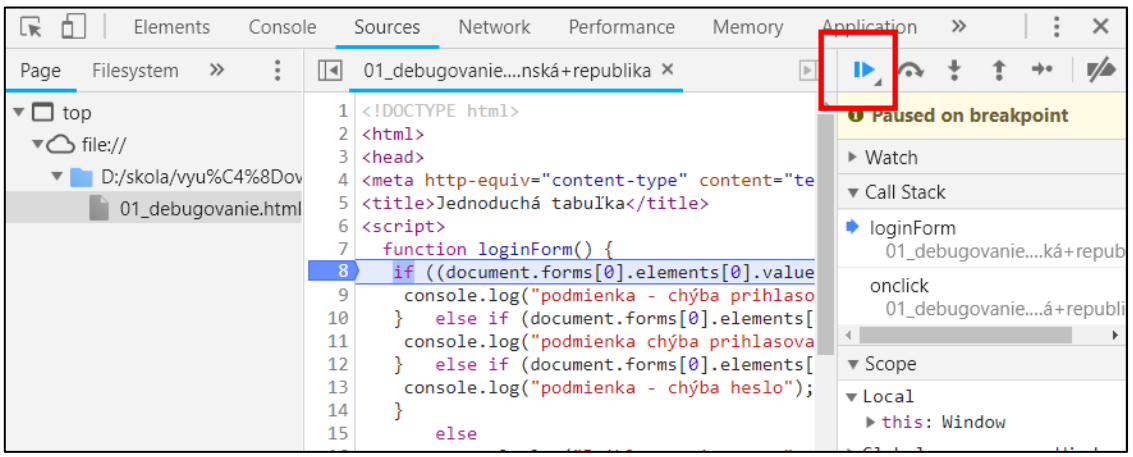
Vypisovanie správ, hodnôt do konzoly môže pomôcť identifikovať chybu. V prípade, že nechceme zasahovať do zdrojového kódu – pridávať a následne odstraňovať metódy na výpis do konzoly, môžeme nastaviť tzv. body prerušenia (angl. breakpoints). Tieto body prerušenia nastavujeme priamo v nástroji vývojára. V prípade použitia webového prehliadača Chrome vyberieme záložku `Sources`.



Bod prerušenia nastavíme tak, že klikneme na číslo riadku, kde chceme, aby sa vykonávanie zdrojového kódu pozastavilo. Napríklad, ak chceme, aby sa vykonávanie zdrojového kódu zastavilo v príklade 14.1 na podmienke `if` (riadok 8), klikneme na riadok s číslom 8.



Keď už máme nastavený bod prerušenia (môže ich byť aj viacej), stlačíme tlačidlo `Odošli` a počkáme kým sa zdrojový kód pozastaví na riadku s bodom prerušenia (riadok sa zvýrazní a v okne napravo sa zobrazí správa `Paused on breakpoint`).



Keď chceme pokračovať vo vykonávaní zdrojového kódu, môžeme stlačiť ikonku (`Resume script execution`), ktorá vyzerá ako ikonka na spustenie prehrávania v prehrávači (ikonka nad textom `Paused on breakpoint`). Po stlačení tohto tlačidla sa vykoná zvyšný zdrojový kód, resp. spustí sa po nasledujúci bod prerušenia (v prípade, že sme ho nastavili).

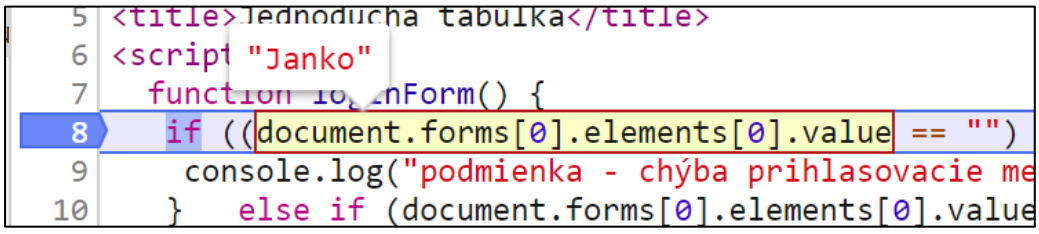
Okrem možnosti pokračovať vo vykonávaní zdrojového kód máme na výber:

- Prejdi cez kód (`Step over`) – ak nás riadok, na ktorom sa nachádzame nezaujíma, tak pokračuj na ďalší riadok.

- Vojsť dnu (`Step into`) – ak sa nachádzame na riadku s volaním funkcie a chceme vojsť do vnútra tejto funkcie, stlačíme toto tlačidlo.

- Odísť preč (`Step out`)– ak sa pri krokovaní nachádzame vo vnútri funkcie, ktorá nie je pre nás dôležitá, pomocou tohto kroku prejdeme na nasledujúci riadok, ktorý sa nachádza za volaním funkcie, kde sa práve nachádzame.

Počas krokovania môžeme sledovať hodnoty jednotlivých objektov, premenných v okne so záložkou `Scope`, alebo ak presunieme kurzor myši na objekt v okne, tak sa zobrazí popis zvoleného objektu. V prípade, že chceme vedieť, čo sa nachádza napr. v príkaze `document.forms[0].elements[0].value` (akú hodnotu zadal používateľ pri prihlasovacom mene) zvýrazníme kurzorom tento príkaz a následne sa nám zobrazí okno s popisom (na obrázku s textom `Janko`).



Kľúčové slovo debugger

Alternatívou k nastaveniu bodov prerušenia v nástroji vývojárov je možnosť vloženia kľúčového slova `debugger` priamo do zdrojového kódu.

```
debugger;
console.log("podmienka - chýba prihlasovacie meno a heslo");
```

14/02_debugger.html

PRÍKLAD 14.2

Otvorte si príklad 14.1 (súbor *14/01_krokovanie.html*). Pridajte bod prerušenia – kľúčové slovo `debugger` nad každý výpis do konzoly.

```
<script>
  function loginForm() {
  if ((document.forms[0].elements[0].value == "") &&
      (document.forms[0].elements[1].value == "")) {
      debugger;
      console.log("podmienka - chýba prihlasovacie meno a
      heslo");
   }
   else if (document.forms[0].elements[0].value == "") {
      debugger;
      console.log("podmienka chýba prihlasovacie meno");
   }
   else if (document.forms[0].elements[1].value == "") {
      debugger;
      console.log("podmienka - chýba heslo");
   }
   else {
      debugger;
      console.log("Prihlasovacie meno:" +
      document.forms[0].elements[0].value);
  }
}
</script>
```

Hlavnou nevýhodou takéhoto spôsobu krokovania je, že musíme pridávať a následne odstraňovať toto kľúčové slovo priamo zo zdrojového kódu. Môže sa teda stať, že ho zabudneme odstrániť.

14.3 Metodika pre učiteľa

CIEĽ

Cieľom je oboznámiť študenta s možnosťami odhaľovania chýb a ladenia programu.

MOTIVÁCIA

Študent dokáže odhaľovať chyby vo vytvorenej webovej stránke pomocou nástrojov pre ladenie.

VÝKLAD

Študentov treba oboznámiť:

- s možnosťou sledovania programu pomocou funkcie `alert()`,
- s možnosťou logovania do konzoly,
- s použitím bodov prerušenia.
- Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch. Žiakom je potrebné zdôrazniť, že ladenie kódu v praxi častokrát predstavuje významnú časť práce programátora pri vytváraní programov.

15 JAVASCRIPT – POKROČILÉ TECHNIKY

Táto kapitola opisuje pokročilejšie techniky pre prácu s jazykom JavaScript. Táto kapitola je nepovinná.

15.1 Knižnice pre JavaScript

Knižnice poskytujú funkcie, ktoré uľahčujú prácu pri tvorbe zdrojového kódu tým, že umožňujú použiť už vytvorený kód aj v iných programoch.

15.1.1 React

React je knižnica pre JavaScript vhodná na vytváranie používateľského rozhrania. O jej vývoj sa stará spoločnosť Facebook spoločne s komunitou vývojárov. Knižnica React je distribuovaná pod MIT licenciou. Pri vytváraní používateľského rozhrania poskytuje možnosť rozdeliť každú stránku na viacero časti, tzv. komponentov. Každý z komponentov obsahuje svoj vlastný zdrojový kód, ktorý zodpovedá príslušnej časti stránky. React používa jazyk JSX, ktorý vyzerá podobne ako jazyk HTML s tým rozdielom, že JSX je možné používať aj v rámci JavaScript zdrojového kódu, čo v podstate umožňuje vkladanie jednoduchých HTML príkazov do sekcie JavaScriptu. Viacero vytvorených komponentov je potom možné vnárať a tak vytvoriť výslednú stránku. Samotné komponenty je možné vytvárať ako funkcionálne alebo triedne. Nevýhodou knižnice je jej veľká pamäťová náročnosť nakoľko pre ukladanie používateľského rozhrania využíva virtuálny DOM, ktorý sa následne synchronizuje so skutočným DOM.

Podrobnejšie informácie, ako aj tutoriál a samotnú knižnicu je možné nájsť na domovskej stránke https://reactjs.org/

15.1.2 jQuery

jQuery je knižnica pre JavaScript, ktorá umožňuje vykonať mnohé bežné úlohy jednoduchým zavolaním metódy namiesto písanie dlhšieho JavaScript kódu. Vývoj knižnice zabezpečuje tím dobrovoľníkov v rámci projektu jQuery. Knižnica je distribuovaná pod MPI licenciou. Medzi základné funkcionality patrí manipulácia s HTML/DOM, manipulácia s CSS, práca s metódami pre HTML udalostí, efekty a animácie, vývoj AJAX aplikácií a rozšíriteľnosť pomocou plug-in modulov. Ďalšou z významných výhod knižnice je jej široká podpora mnohých prehliadačov webových stránok, čo umožňuje jednotný vzhľad a funkcionalitu stránky v rôznych prehliadačoch. Knižnicu nie je potrebné mať nainštalovanú, je možné použiť knižnicu

poskytovanú spoločnosťami Google alebo Microsoft, čo môže prispieť k zrýchleniu prezerania ďalších stránok využívajúcich túto knižnicu vďaka súborom uloženým v pamäti cache.

Podrobnejšie informácie, ako aj tutoriál a samotnú knižnicu je možné nájsť na domovskej stránke https://jquery.org/

15.1.3 AngularJS

AngularJS je pracovný rámec pre front-end webové aplikácie vyvíjaný spoločnosťou Google a komunitou. Jeho distribúcia je možná v rámci MIT licencie. AngularJS je vhodný nástroj pre vývoj SPA (Single Page Application). Poskytuje rozšírenie HTML DOM o ďalšie atribúty a umožňuje jeho jednoduchšie prispôsobenie na akcie používateľa tým, že poskytuje prepojenie dát s HTML. V rámci pracovného rámca je možné používať šablóny pohľadov definované v HTML jazyku. Pozostáva z troch základných častí: ng-app, ktorá vytvára prepojenie medzi AngularJS aplikáciou a HTML, ng-model zabezpečuje prepojenie medzi hodnotou vstupného HTML elementu a dátami aplikácie, ng-bind prepája dáta aplikácie s HTML pohľadom. Vzhľadom k veľkému počtu používateľov je dostupná aj dobrá dokumentácia. Taktiež poskytuje dobré možnosti testovania. Problematické však môže byť zavádzanie do už existujúcich stránok. V prípade použitia veľkého počtu interaktívnych elementov na stránke môže spôsobiť použitie AngularJS spomalenie z dôvodu potreby synchronizácie.

Podrobnejšie informácie, ako aj tutoriál je možné nájsť na domovskej stránke https://angularjs.org/

15.1.4 Ember

Ember predstavuje pracovný rámec založený na MVC (Model-View-Controller) vzore. Je vyvíjaný komunitov programátorov a distribuovaný pod MIT licenciou. Súčasťou pracovného rámca sú aj smerovače a komponenty. Smerovač reprezentuje stav aplikácie v závislosti na konkrétnej URL. Zabezpečuje poskytnutie korektnej šablóny pre používateľské rozhranie. Ku

každému smerovaču sa vzťahuje jeden model, ktorý obsahuje dáta spojené s aktuálnym stavom aplikácie. Na prepojenie modelu a zobrazovacej logiky sa používa kontrolér. Používateľské rozhranie predstavuje samotný pohľad vytvorený na základe šablóny napísanej v jazyku Handlebars, ktorá definuje, ako bude používateľské rozhranie vyzerať. Správanie používateľského rozhranie je kontrolované pomocou komponentov, ktoré pozostávajú z dvoch častí: šablóny napísanej v Handlebars a zdrojového kódu v jazyku JavaScript. Nevýhodou použitia tohto pracovného rámca je použitia jazyka Handlebars na vytváranie šablón a množstvo JavaScript zdrojového kódu v HTML kóde.

Podrobnejšie informácie, ako aj tutoriál je možné nájsť na domovskej stránke https://emberjs.com/

15.1.5 Vue.js

Vue.js je pracovný rámec na vytváranie používateľského rozhrania webových stránok. Je vyvíjaný komunitou vývojárov a distribuovaný v rámci MIT licenčných podmienok. Na rozdiel od iných rámcov je navrhnutý od základov tak, aby ho bolo možné postupne adoptovať. Základ predstavuje knižnica zameraná na zobrazovaciu vrstvu, ktorá je ľahko použiteľná a integrovateľná s inými knižnicami alebo už existujúcimi projektami. V kombinácií s inými nástrojmi a knižnicami je vhodným nástrojom pre sofistikované SPA.

Podrobnejšie informácie, ako aj tutoriál je možné nájsť na domovskej stránke https://vuejs.org/

15.1.6 Polymer

Polymer je knižnica, pomocou ktorej je možné vytvárať vlastné elementy. Je vyvíjaná spoločnosťou Google a ďalšími prispievateľmi je distribuovaná pod BSD licenciou. Knižnica obsahuje funkcionality, ktoré umožňujú jednoduché a rýchle vytváranie vlastných elementov obdobných štandardným DOM elementom. Takýmto spôsobom je možné vytvoriť napríklad vlastný netradičný vstupný element formulára. Zapúzdrenie vlastných elementov poskytuje Shadow DOM. Taktiež umožňuje import distribuovaných komponentov.

Podrobnejšie informácie, ako aj tutoriál je možné nájsť na domovskej stránke https://www.polymer-project.org/

15.1.7 AJAX

AJAX (Asynchronous JavaScript And XML) predstavuje množinu nástrojov pre vývoj webových stránok, ktoré umožňujú meniť obsah stránok bez toho, aby ich bolo potrebné opätovne načítať zo servera. Umožňuje webovej aplikácie asynchrónnu výmenu dát so serverom na pozadí bez toho, aby bolo nejak ovplyvnené zobrazenie alebo správanie stránky. AJAX predstavuje spojenie viacerých prvkov dohromady: HTML a CSS pre značkovanie a štýlovanie informácií pri zobrazení, DOM spojený s JavaScript kódom pre dynamické zobrazenie a interakciu s prezentovanou informáciou, metódy pre výmenu dát medzi prehliadačom a serverom, bez nutnosti obnovovať zobrazovanú stránku (najčastejšie sa používa XMLHttpRequest (XHR) objekt, niekedy je použitý IFrame objekt alebo dynamicky pridaný <script> tag) a formát pre dáta poslané prehliadaču (bežné formáty zahŕňajú XML, predformátované HTML, plain text a JavaScript Object Notation (JSON)), pričom tieto dáta môžu byť dynamicky vytvorené skriptom na strane servera.

Podrobnejšie informácie, ako aj tutoriál je možné nájsť na stránke https://www.w3schools.com/xml/ajax_intro.asp/

15.2 Štandard jazyka JavaScript

JavaScript je skriptovací jazyk, ktorý sa neustále vyvíja. V súčasnosti (rok 2018) sa za najnovšiu verziu považuje verzia ECMAScript 6, ktorá sa zvykne označovať aj ECMAScript 2015, alebo skrátene ES6. Táto verzia bola vydaná v roku 2015 a priniesla viaceré nové funkcie:

- Príkaz `let`,
- Príkaz `const`,
- JavaScript preddefinované parametre,
- Viacriadkové hodnoty String,
- Nové funkcie `find()` a `findIndex()` dostupné pre objekt `Array`,
- Funkcia šípky `=>`,
- Triedy,
- Moduly,
- A mnohé iné.

V tejto kapitole sa budeme venovať príkazom let a const. Zvyšné novinky v jazyku sú opísané na stránke http://es6-features.org.

15.2.1 JavaScript príkaz let

Do verzie ES6 bolo možné vytvárať premenné iba pomocou príkazu `var`. Od verzie ES6 môžeme vytvárať premenné okrem príkazu `var` aj pomocou príkazu `let`. Premenné definované pomocou príkazu `let` majú menší obor platnosti.

Príkaz `let` sa odporúča používať pri podmienkach, cykloch, funkciách, blokoch, keď chceme zabezpečiť, aby sa hodnota premennej nedala použiť mimo blokov.

Syntax:

```
let NÁZOV = hodnota;
```

Príklad:

```
let i = 5;
```

Rozdiel medzi príkazmi var a let

Premenná definovaná pomocou príkazu `var` vo vnútri blokov je dostupná aj mimo blokov. Premenná definovaná pomocou príkazu `let` vo vnútri blokov je dostupná iba v rámci bloku.

15/01_prikaz Var.html

PRÍKLAD 15.1

Vytvorte stránku, do ktorej do časti:

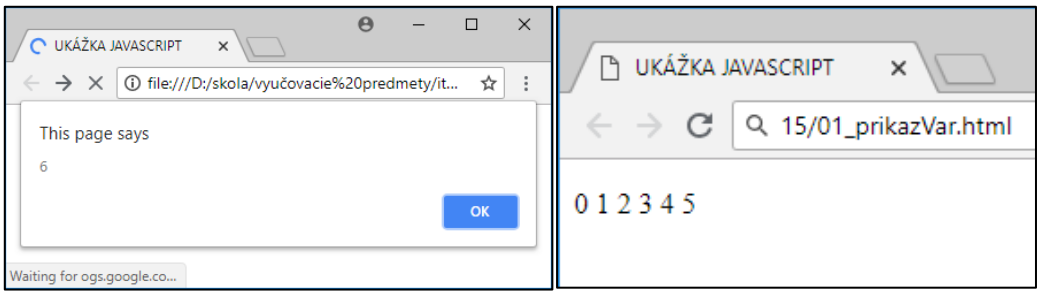
- `body` vložte odstavec s atribútom `id` `cisla`,
- `<script>` vložte cyklus s pevným počtom opakovaní (`for`), ktorý bude postupne vypisovať čísla od `0` po `5`, čísla budú oddelené medzerou,
- za cyklom `for` zavolajte funkciu na zobrazovanie dialógového okna `alert()`, ktorá bude obsahovať parameter hodnotu pomocnej premennej z cyklu.

Poznámka: Na vypisovanie čísiel použite metódu `getElementById()` objektu `document`.

Pri deklarovaní premennej použite príkaz `var`.

```
<p id="cisla"></p>

<script>
      for (var i = 0; i <= 5; i++) {
            document.getElementById("cisla").innerHTML =
            document.getElementById("cisla").innerHTML + " " + i;
      }

      alert(i);

</script>
```

Tento kód zobrazí najskôr dialógové okno s hodnotou `6` a po zavretí dialógového okna vypíše do paragrafu s `id` `cisla` nasledujúce čísla `0 1 2 3 4 5`.



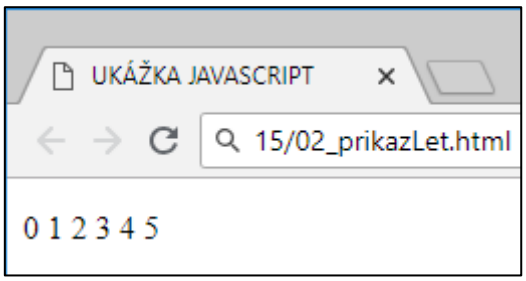
15/02_prikazLet.html

PRÍKLAD 15.2

Upravte príklad 15.1 tak, že nahradíte príkaz `var`, príkazom `let`.

```
<p id="cisla"></p>

<script>
      for (let i = 0; i <= 5; i++) {
            document.getElementById("cisla").innerHTML =
            document.getElementById("cisla").innerHTML + " " + i;
      }

      alert(i);

</script>
```

Tento kód nezobrazí dialógové okno s hodnotou premennej `i`, nakoľko táto premenná nie je dostupná mimo oboru platnosti funkcie. Čísla `0 1 2 3 4 5` sa však vypíšu do paragrafu s `id cisla`.



15.2.2 JavaScript príkaz const

Pomocou príkazu `const` vieme v jazyku JavaScript vytvoriť konštanty. Príkaz `const` má podobné správanie ako príkaz `let` s tým rozdielom, že hodnotu už nemôžeme meniť.

ZAPAMÄTAJTE SI

Konštantou sa pri programovaní označuje premenná, ktorá obsahuje určitú hodnotu, ktorú už nie je možné meniť.

Predstavme si napr. Ludolfovo číslo π , ktoré má hodnotu 3,14 (prvé tri cifry). Táto hodnota sa nikdy nemení, preto ju môžeme pri programovaní označiť ako konštantu a tým zabezpečíme, že sa táto hodnota nikde v programe nezmení.

Pri deklarovaní konštanty musíme nastaviť konštante jej hodnotu. Názov konštanty sa odporúča písať veľkými písmenami.

Syntax:

```
const NÁZOV = hodnota;
```

Príklad:

```
const PI = 3.14;
PI = 4; // konštantu nemôžeme meniť, tento riadok spôsobí chybu
```

15.3 Metodika pre učiteľa

CIEĽ

Cieľom je oboznámiť študenta s pokročilými možnosťami používania jazyka JavaScript pomocou najpoužívanejších JavaScript knižníc.

MOTIVÁCIA

Študent dokáže vytvoriť JavaScript projekt s použitím rôznych knižníc.

VÝKLAD

Výklad je potrebné striedať s praktickými ukážkami na uvedených príkladoch. Žiakom je potrebné zdôrazniť, že používanie knižníc je vhodné na uľahčenie programovania – použiť niečo, čo už niekto vytvoril a nevytvárať nové veci.

BIBLIOGRAFIA

W3schools.com [online]. [cit. 2019-5-23]. URL: <https://www.w3schools.com/js/default.asp>.

Polymer [online]. [cit. 2019-5-23]. URL: <https://www.polymer-project.org>.

Vue.js [online]. [cit. 2019-3-11]. URL: < https://vuejs.org>.

Ember.js [online]. [cit. 2019-3-11]. URL: <https://emberjs.com>.

AngularJS [online]. [cit 2019-3-11]. URL: <https://angularjs.org>.

jQuery [online]. [cit. 2019-3-12]. URL: < https://openjsf.org>.

React [online]. [cit. 2019-3-12]. URL: < https://reactjs.org>.

ŽÁRA, O. 2015. JavaScript (Programátorské techniky a webové technologie). Computer Press, 2015. 184 s. ISBN 978-80-251-4573-9.

FRIESBIE, M. 2017. Professional JavaScript for Web Developers. John Wiley & Sons, 2017. 900 s. ISBN 978-1-119-36644-7.

ZAKAS, N. 2009. JavaScript pro webové vývojáře. Computer Press, 2009. 832 s. ISBN 978-80-251-2509-0.