PROGRAMOVANIE WEBOVÝCH STRÁNOK

MENO PRIEZVISKO

OBSAH

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).

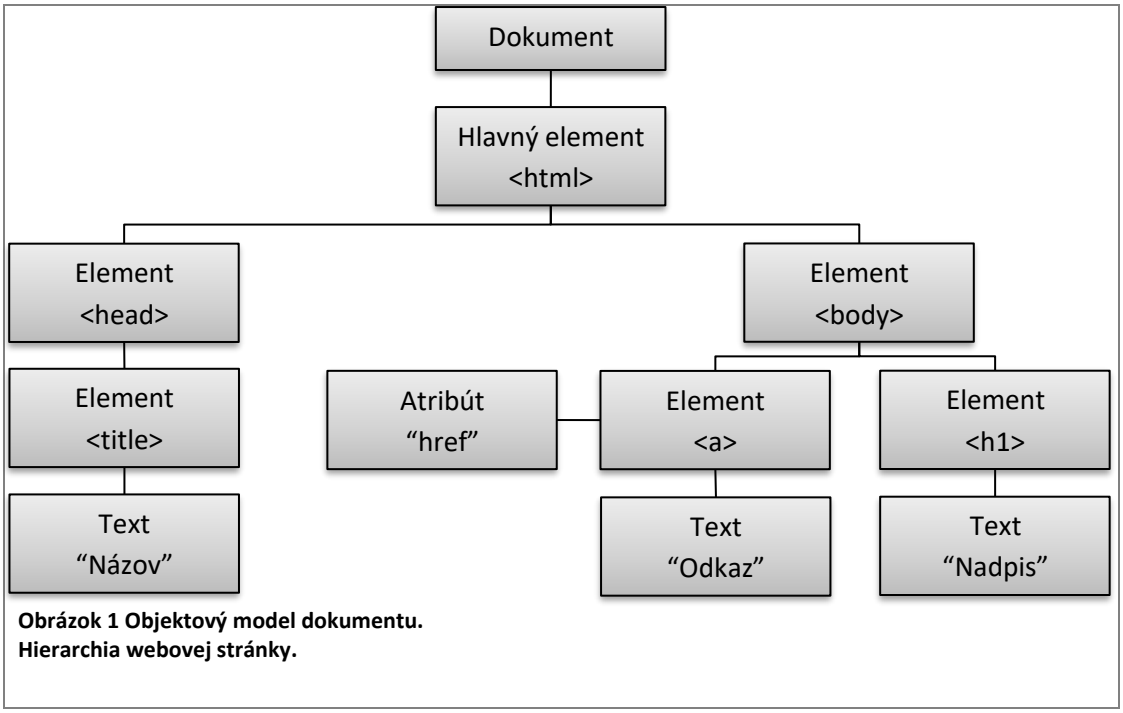
ECMAScript

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()`.

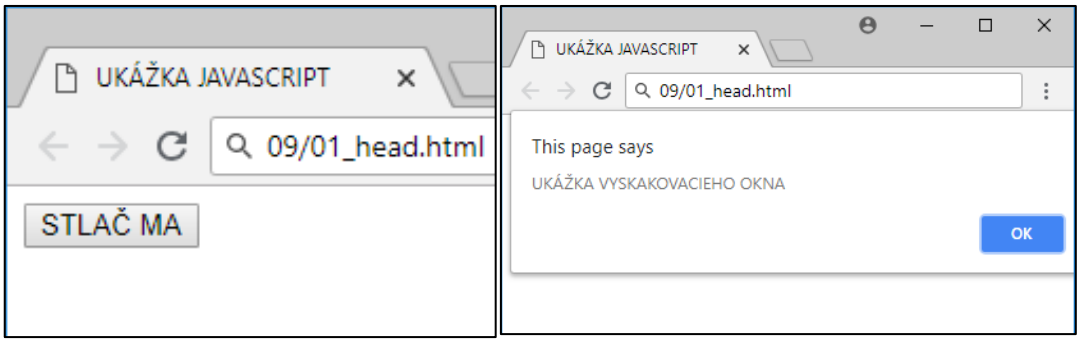
Do časti head vložte párovú značku `<script>`, ktorá bude obsahovať funkciu `mojaFunkcia()`. V tele funkcie bude zavolaná funkcia `alert()`, ktorá zobrazí dialógové okno s textom „`Ukážka vyskakovacieho okna`“.

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>`:

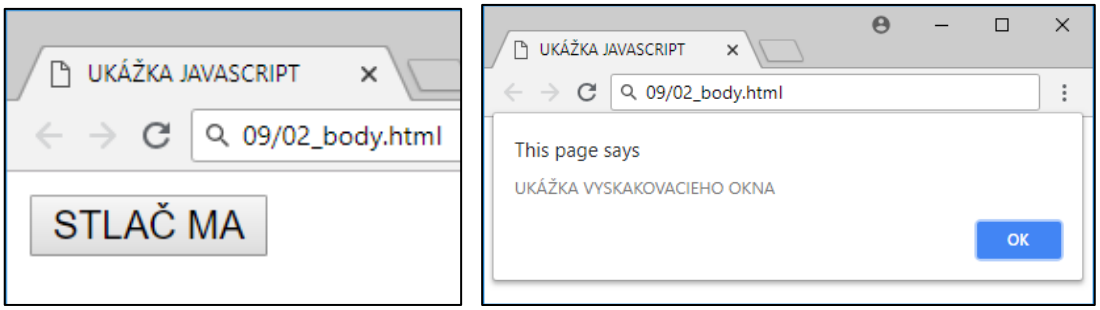
09/02_body.html

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 príklade do časti `<head>` alebo do časti `<body>`.

ZAPAMÄTAJTE SI

Do súboru s príponou `.js` už nedávame párovú značku `<script>`. Obsah tohto súboru predstavuje všetko, čo sa nachádza v párovej značke `<script>`.

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

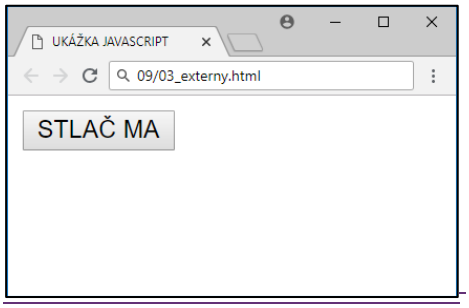
Obsah súboru `03_externy.html:`

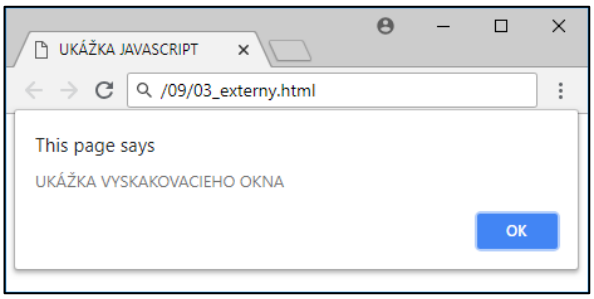
```
<!DOCTYPE html>
<html>

<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 1 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 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, podčiarnik (`_`) alebo znak dolára (`$`). Ostatné znaky už môžu byť čísla, písmená, znak dolára, podčiarnik, 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 podčiarnik.

```
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).

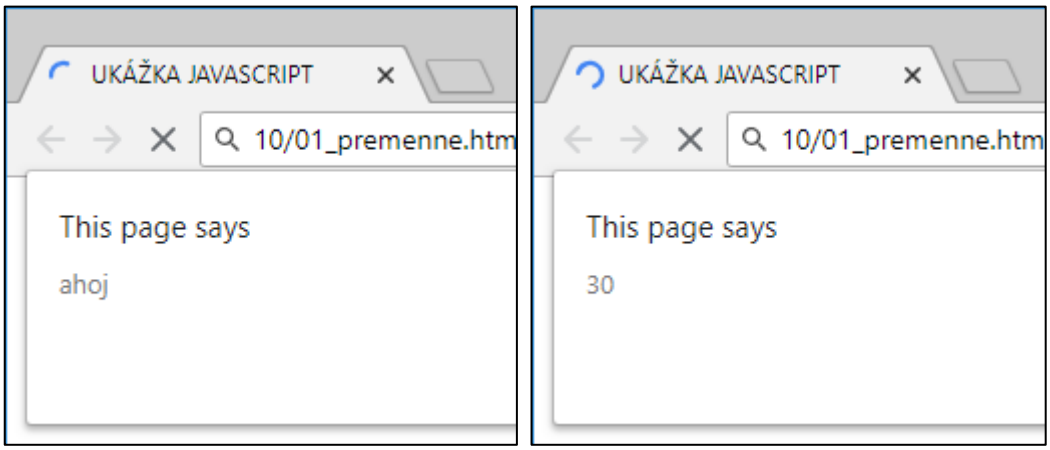
```
var nazovPremennej;
var nazovPremennej2 = hodnota_ktoru_uchovava_premenna;
nazovPremennej2 = nova_hodnota_ktoru_uchovava_premenna;
```

10/01_premenne.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

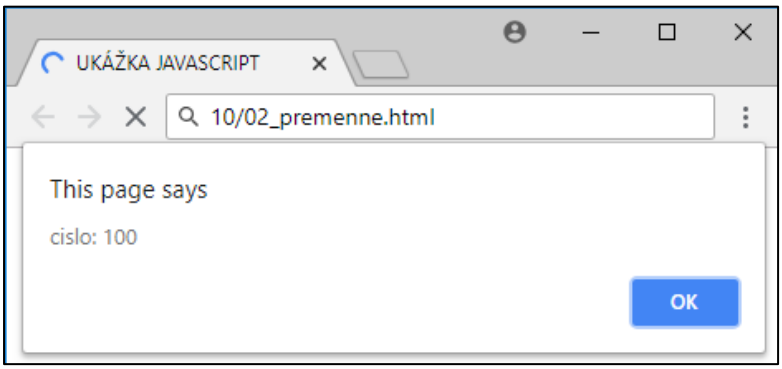
10/02_premenne.html

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 premenných hovoríme kľúčové, resp. vyhradené slová. V tabuľke 2 sa nachádzajú niektoré najčastejšie používané slová, ktoré nemôžeme použiť ako identifikátori. 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 2 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.

PRÍKLAD 10.4

10/04_number.html

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`.

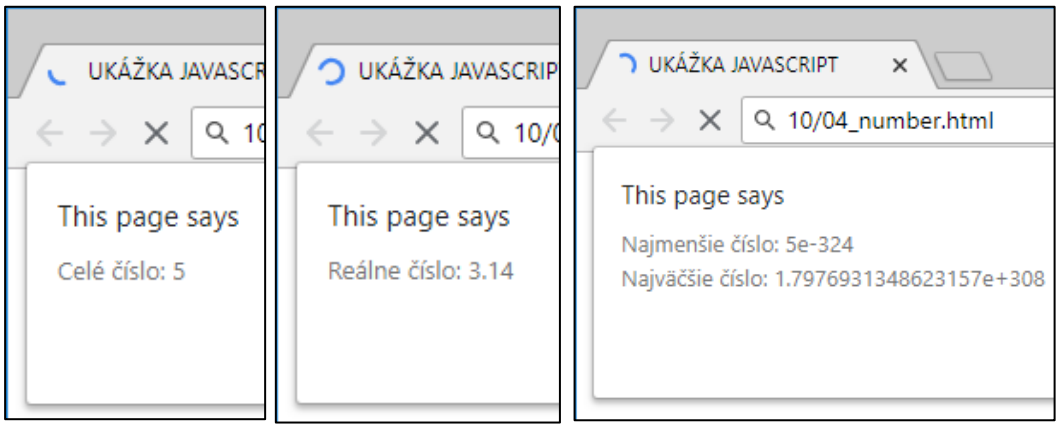
Posledný príkaz, ktorý vložte do značky `<script>` je `alert()` s hodnotou parametra `Najmenšie číslo: Number.MIN_VALUE + Najväčšie číslo: Number.MAX_VALUE.`

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:

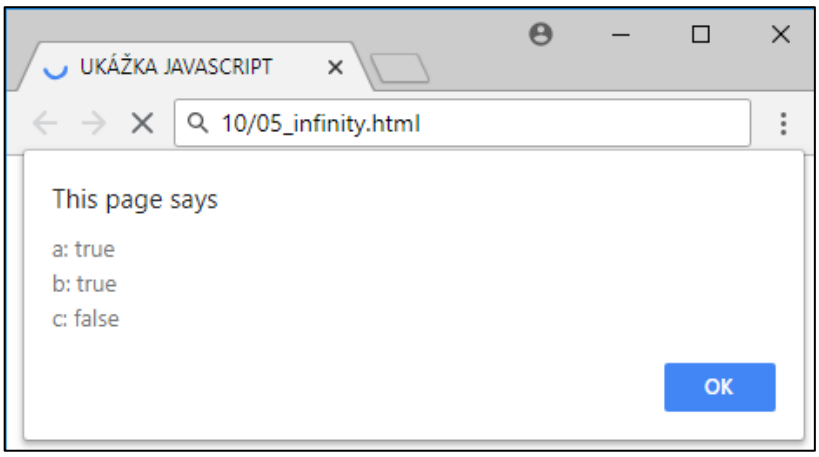
- `maxCislo` s hodnotou `Number.MAX_VALUE`,
- `a` s boolean hodnotou, ktorá sa vypočíta podľa toho, či číslo `123` je konečné,
- `b` s hodnotou, ktorá sa vypočíta podľa toho, či číslo `maxCislo` je konečné,
- `c` s hodnotou, ktorá sa vypočíta podľa toho, či číslo `maxCislo` je konečné. Pred vytvorením 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`.

PRÍKLAD 10.6

10/6_nan.html

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()`.

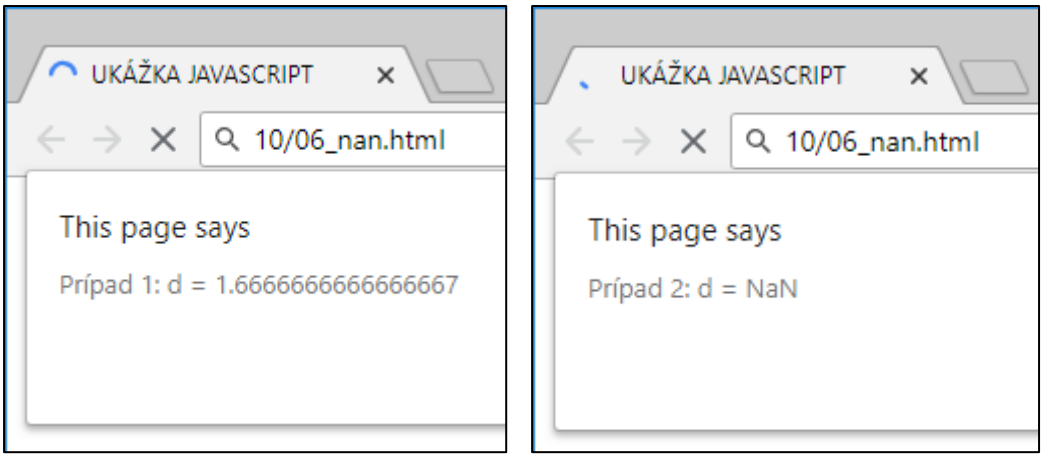
V premennej `d` zmeňte hodnotu na hodnotu `a/c` a premennú `d` s novom hodnotou vložke ako parameter funkcie `alert()`. V ďalšej časti značky `<script>` si vyskúšajte zavolanie funkcie `isNan()`, kde postupne ako parameter vložte:

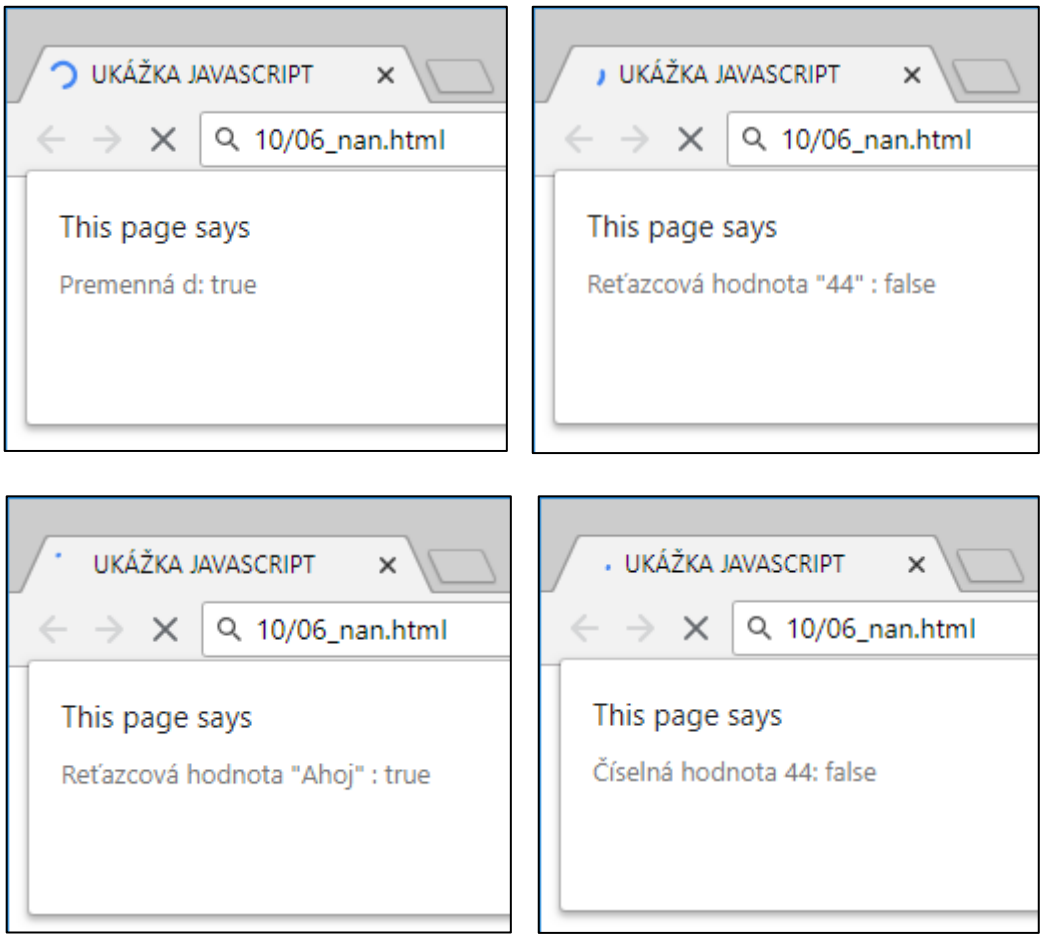
- premennú `d`,
- reťazcovú hodnotu „`44`“,
- reťazcovú hodnotu „`Ahoj`“,
- číselnú hodnotu `44`.

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

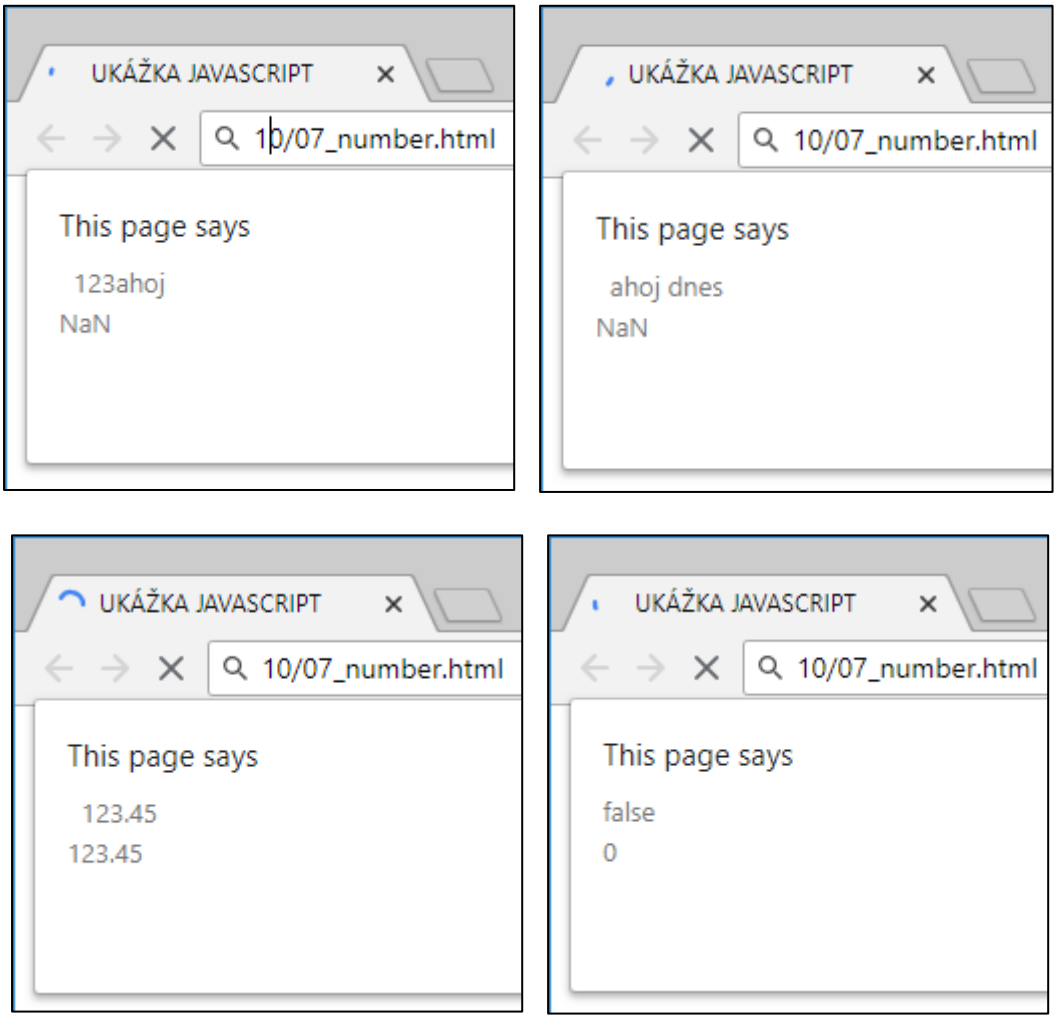
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premenné:

- `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";
```

PRÍKLAD 10.8

10/08_string.html

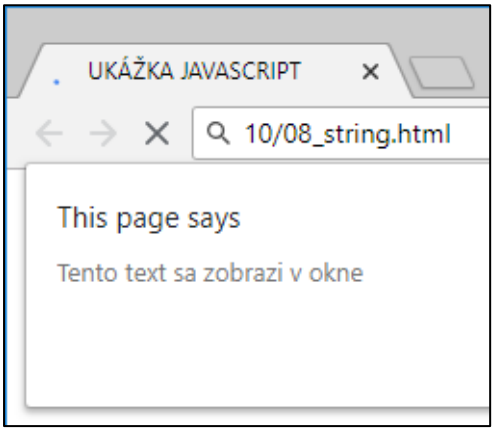
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

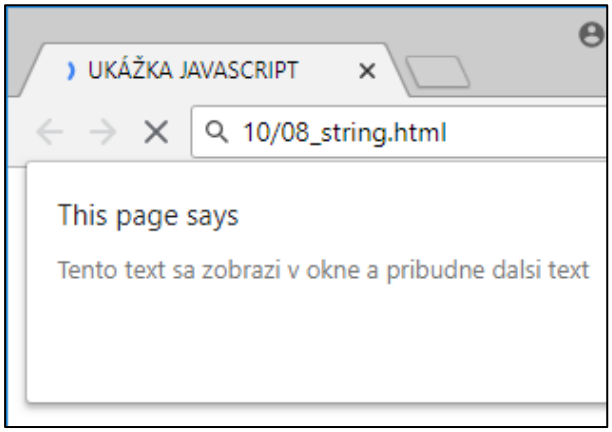
- `text` s reťazcovou hodnotou `Tento text sa zobrazí v okne`.

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 ďalšieho 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

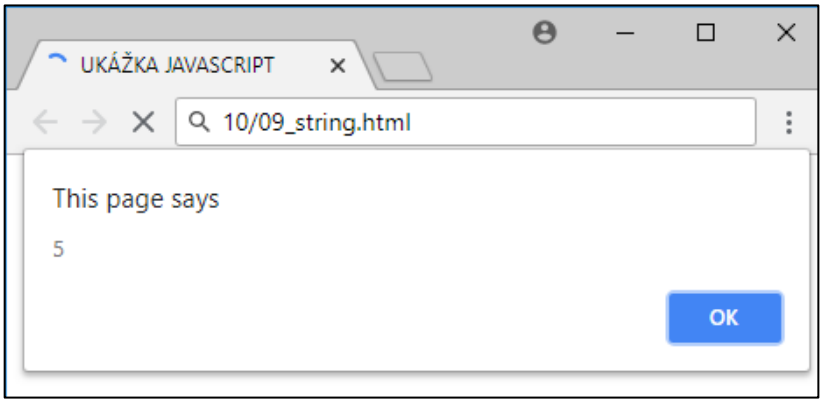
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `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:

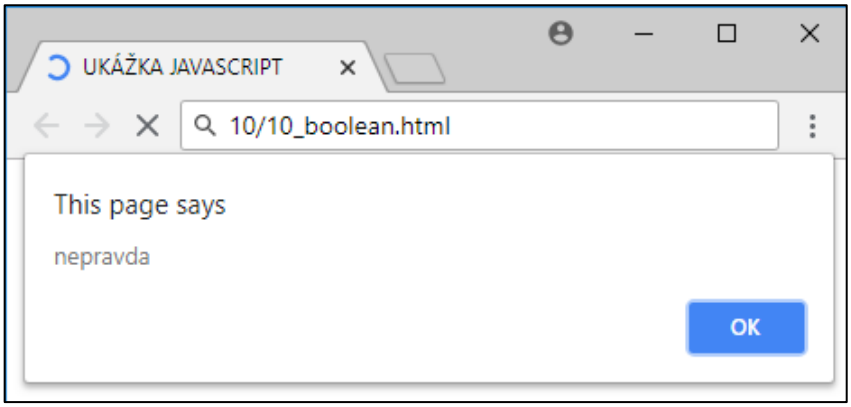
- `x` s logickou premennou `true`,
- `y` s logickou premennou `false`,
- `a` s hodnotou `4`.

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:

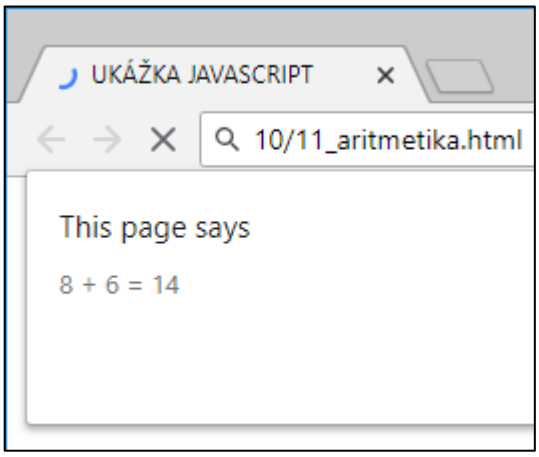
- `premenna1` s číselnou hodnotou `8`,
- `premenna2` s číselnou hodnotou `6`,
- `vysledok` zatiaľ bez hodnoty.

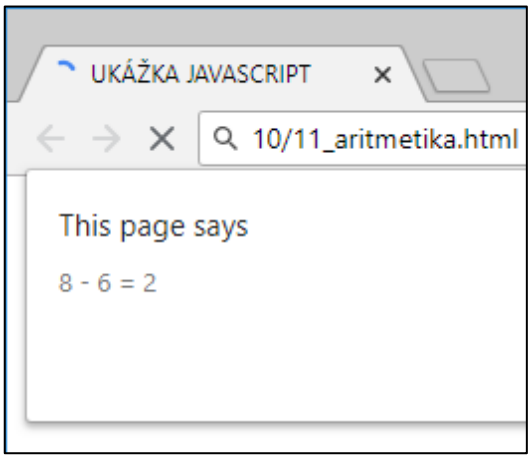
Do premennej `vysledok` postupne uložte hodnoty a súčasne zobrazte výsledok v dialógovom okne:

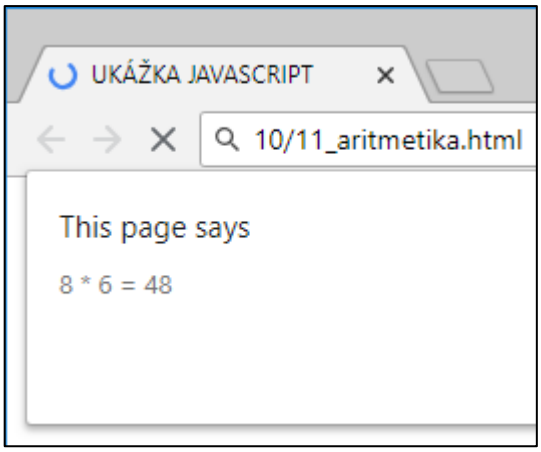
- sčítanie hodnôt `premenna1` a `premenna2`,
- odčítanie hodnoty `premenna2` od hodnoty premennej `premenna1`,
- vynásobenie hodnôt `premenna1` a `premenna2`,
- vydelenie hodnoty `premenna1` hodnotou premennej `premenna2`,

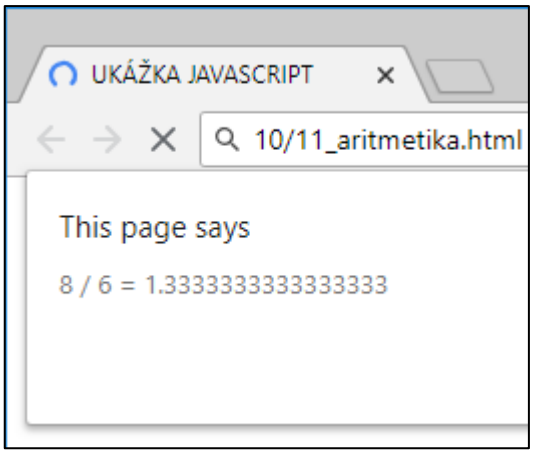
zvyšok po delení hodnôt `premenna1` a `premenna2`.

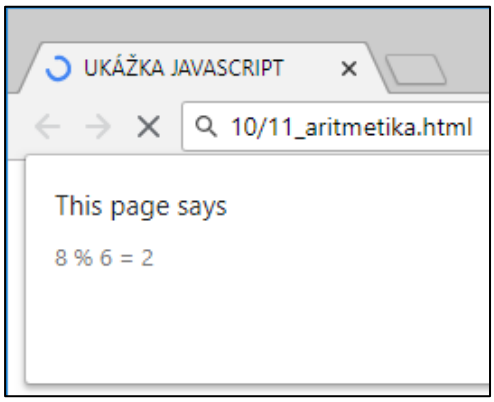
```
var premenna1 = 8;
var premenna2 = 6;

//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:

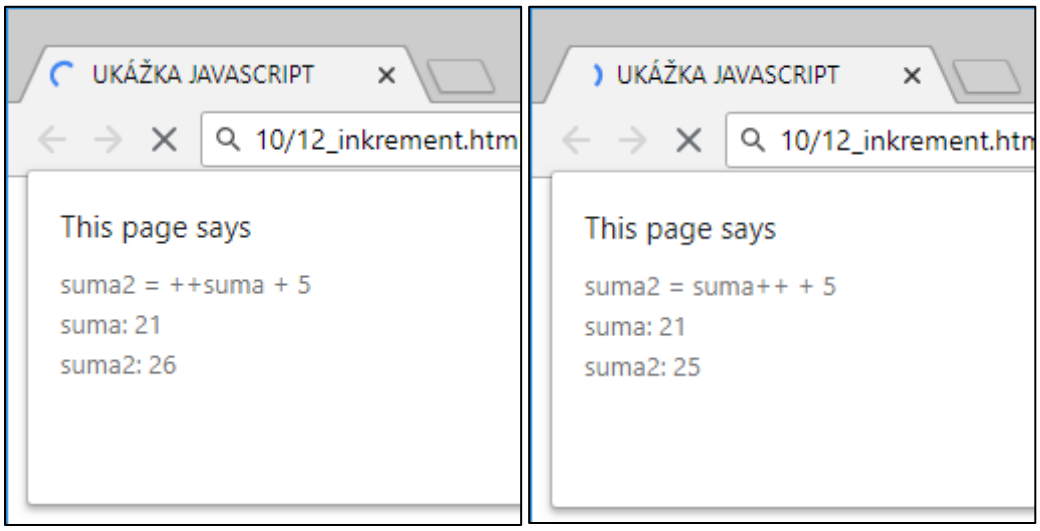
- `suma` zatiaľ bez hodnoty,
- `suma2` zatiaľ bez hodnoty.

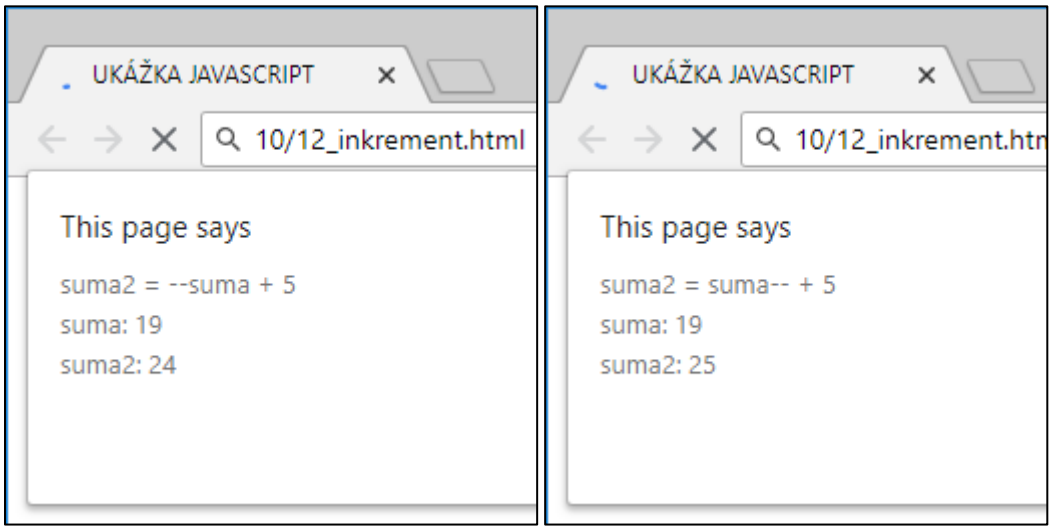
Postupne upravujte hodnoty vytvorených premien:

- 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(`<=`)

ZAPAMÄTAJTE SI

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`.

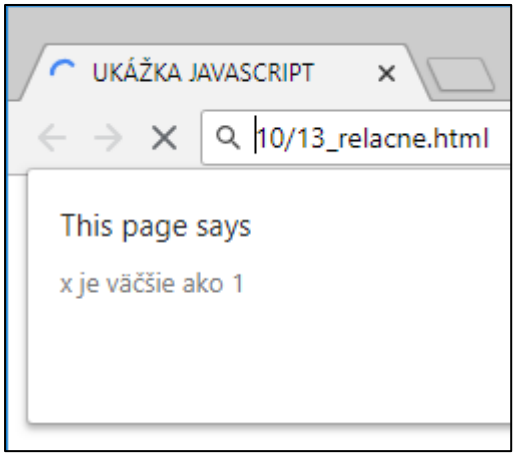
Pod premenné pridajte vetvenia:

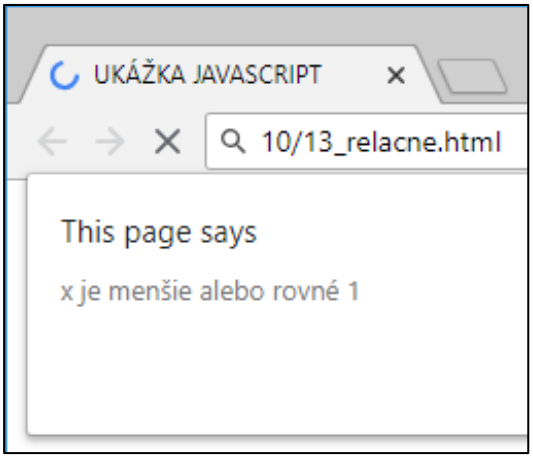
- ak `x` je väčšie ako `1`, tak vypíš `text x je väčšie ako 1`,
- ak `x` je väčšie alebo rovné `1`, tak vypíš text `x je väčšie alebo rovné 5`,
- ak `y` je menšie ako `7`, tak vypíš `y je menšie ako 7`,
- ak `y` je menšie alebo rovné `6`, tak vypíš text *y* `je menšie alebo rovné 6`.

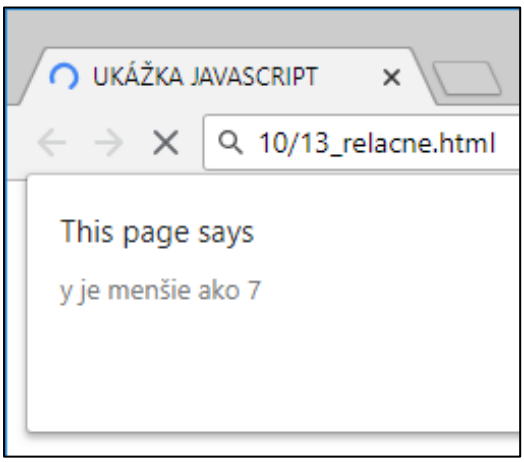
Koľkokrát sa zobrazí dialógové okno?

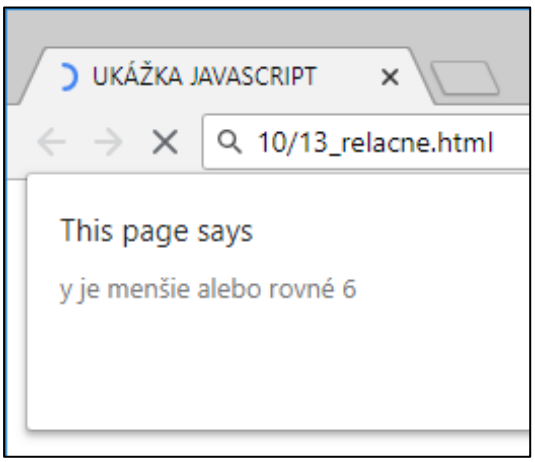
```
var x = 5;
var y = 6;

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 (`!=`).

PRÍKLAD 10.14

10/14_porovnavacie.html

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `x` s číselnou hodnotou `5`,
- `y` s číselnou hodnotou `6`.

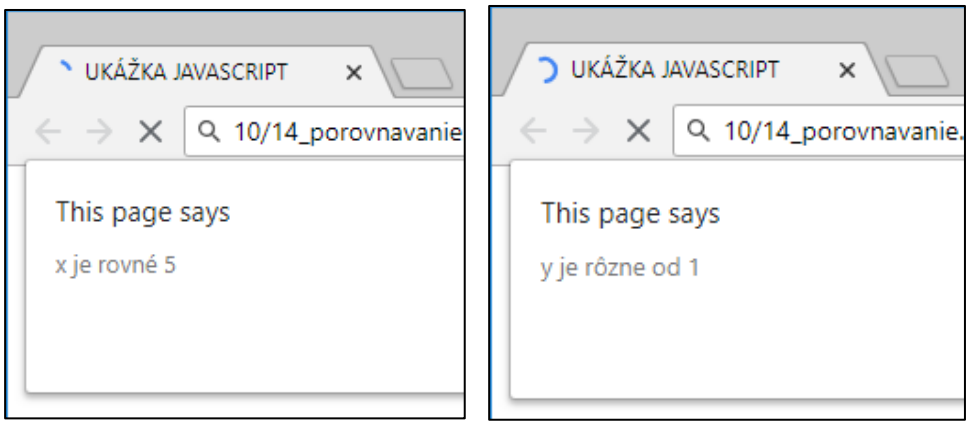
Pod premenné pridajte vetvenia:

- ak `x` 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:

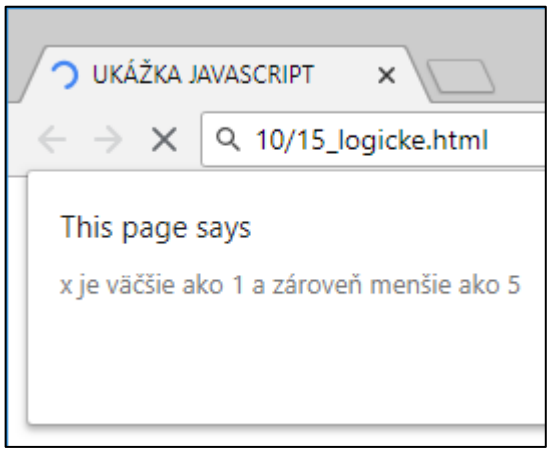
- `x` s číselnou hodnotou `3`.

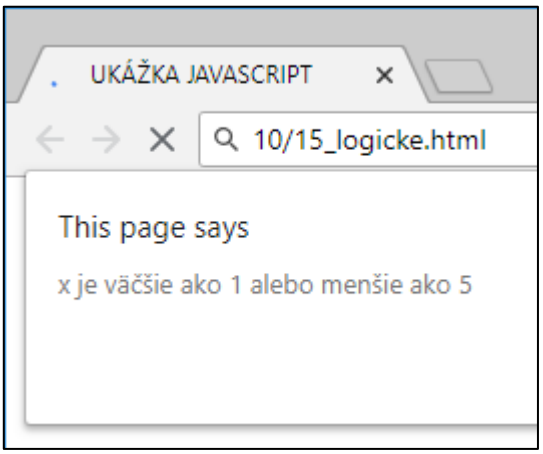
Pod premenné pridajte vetvenia:

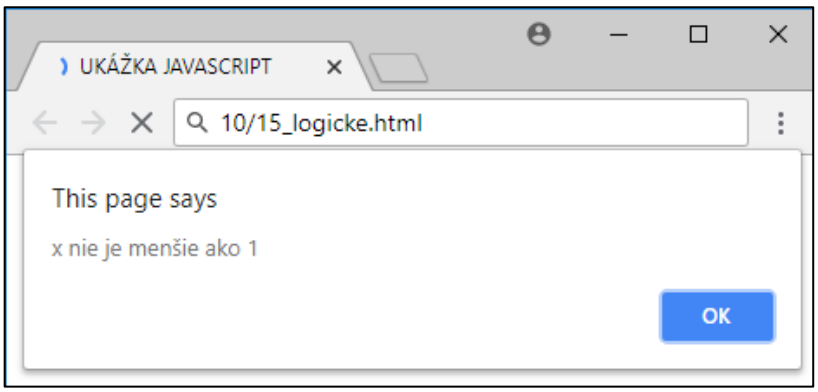
- ak `x` je väčšie ako `1` a súčasne je menšie ako `5`, tak vypíš `text x je väčšie ako 1 a zároveň menšie ako 5`. Ak nie je splnená podmienka tak vypíš text `haha`,
- ak `x` je väčšie ako `1` alebo menšie ako `5` tak vypíš text `x je väčšie ako 1 alebo menšie ako 5`.

```
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.

10/16_priradenie.html

PRÍKLAD 10.16

Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

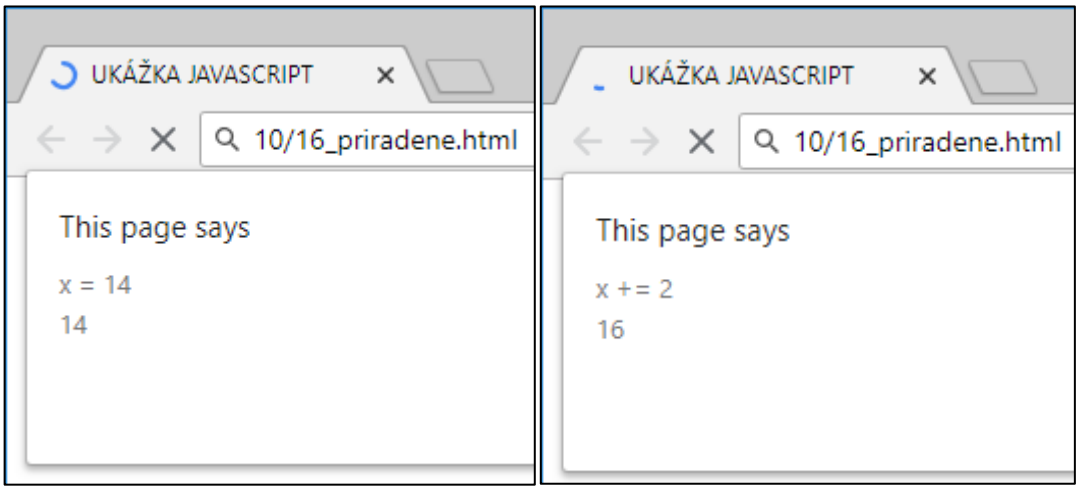
- `x` bez priradenia hodnoty.

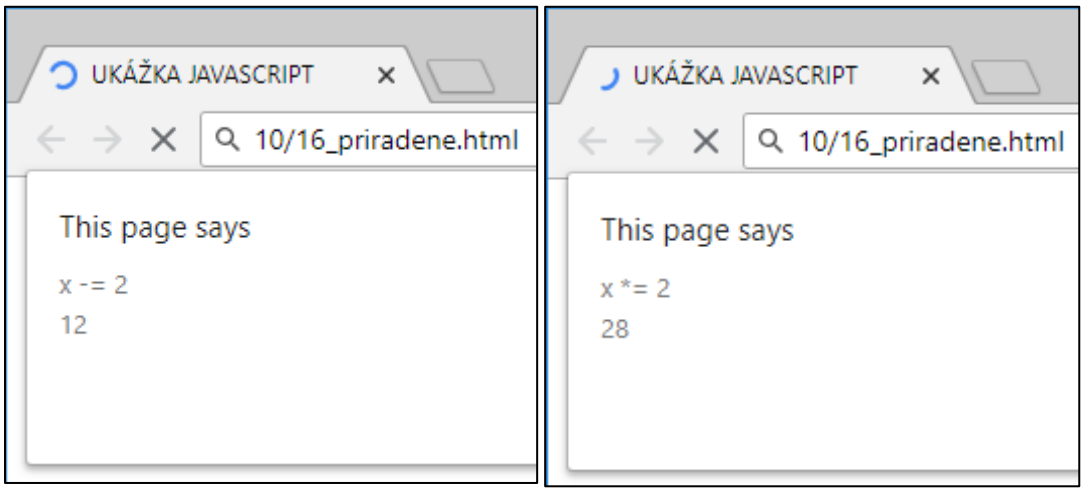
Postupne priraďujte premennej x hodnoty a súčasne ich vypisujte v dialógovom okne:

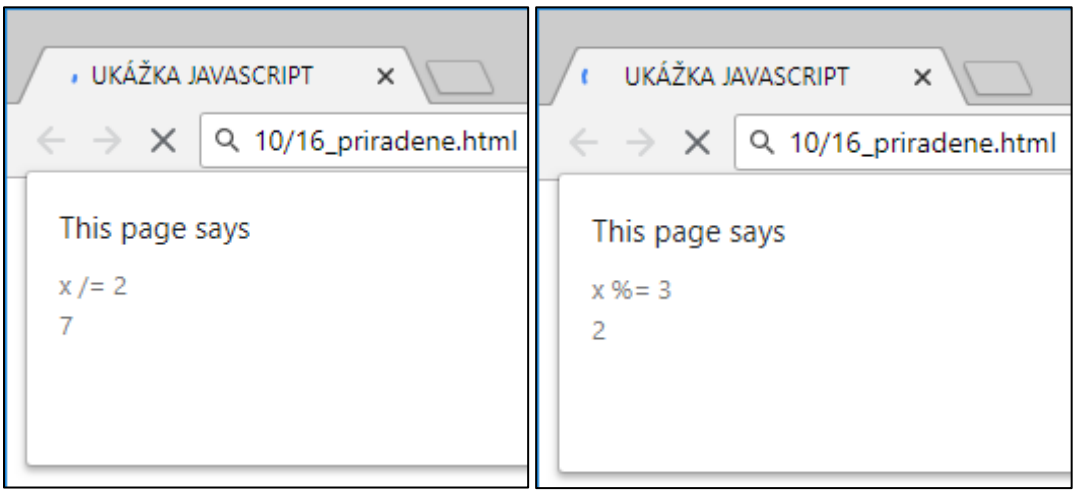
- premennej `x` priraďte hodnotu `14`,
- premennej `x` sčítajte a priraďte hodnotu `2`,
- premennej `x` priraďte hodnotu `14` a následne pomocou zloženého priradenia sčítajte a priraďte hodnotu `2`,
- premennej x priraďte hodnotu `14` a následne pomocou zloženého priradenia odčítajte a priraďte hodnotu `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

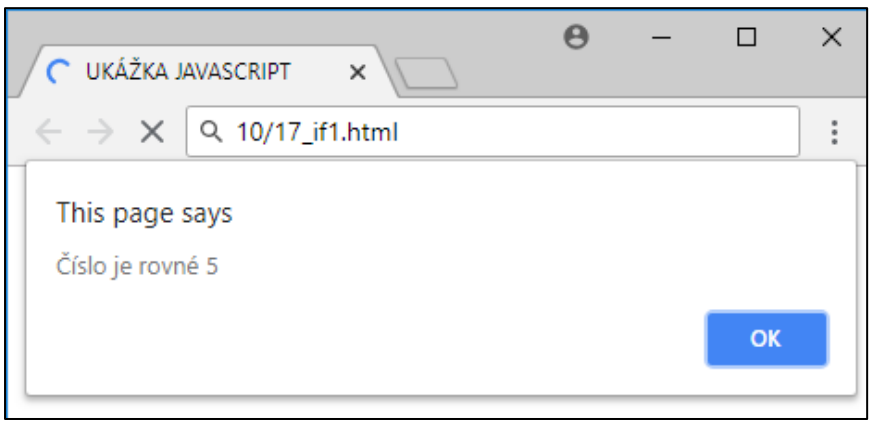
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `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");
}
```

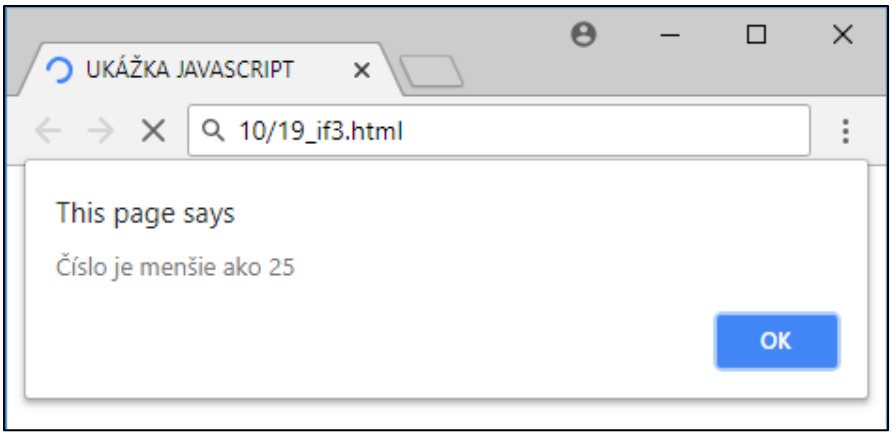
PRÍKLAD 10.19

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`.

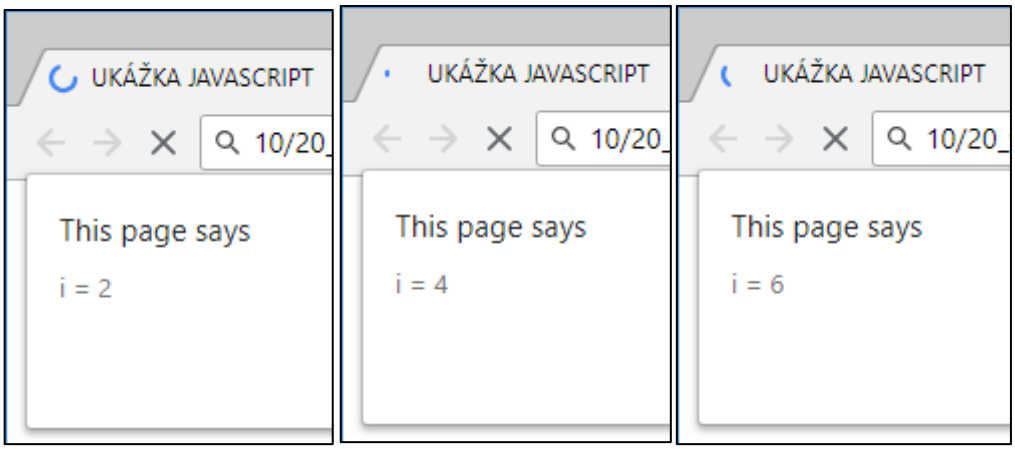
V cykle s podmienkou na začiatku opakujte nasledujúce príkazy, kým `i` je menšie ako `10`:

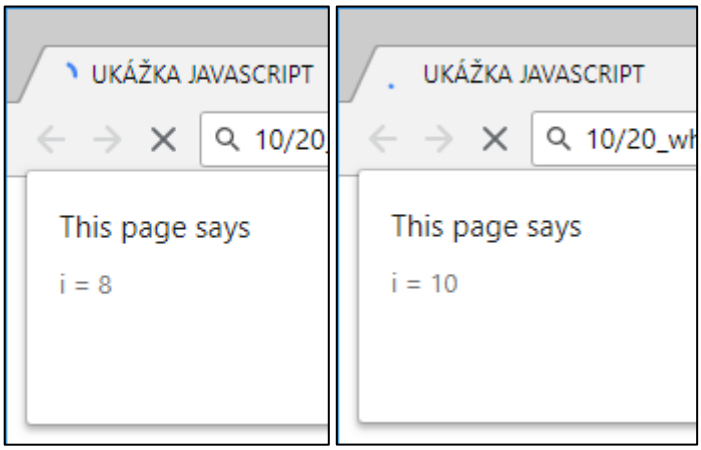
- zväčšite hodnotu `i` o hodnotu `2`,
- vypíšte hodnotu `i` v dialógovom okne.

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

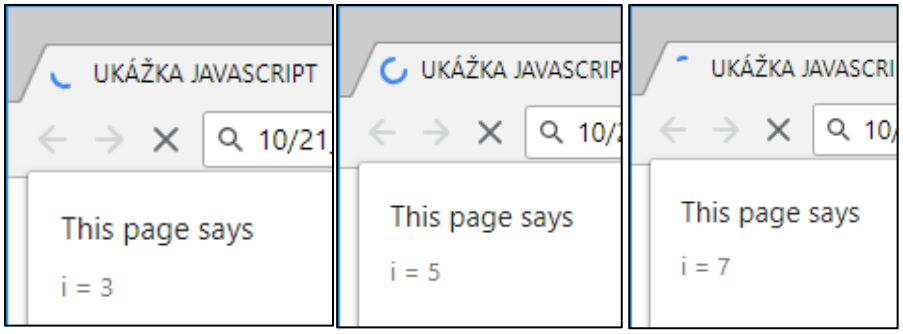
Upravte predchádzajúci príklad 10.20 (súbor 10/20_while.html) tak, aby ste použili namiesto cyklu s podmienkou na začiatku cyklus s podmienku na konci.

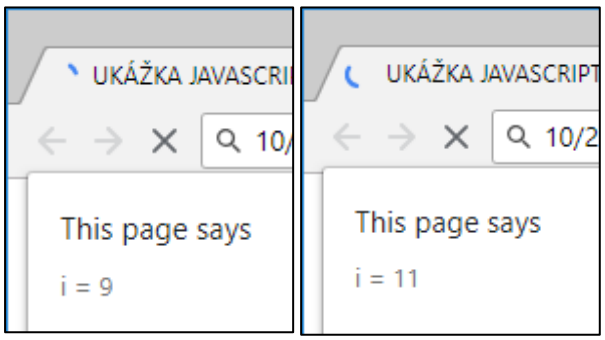
Pri vytváraní premennej `i` ju inicializujte na hodnotu `1`.

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
}
```

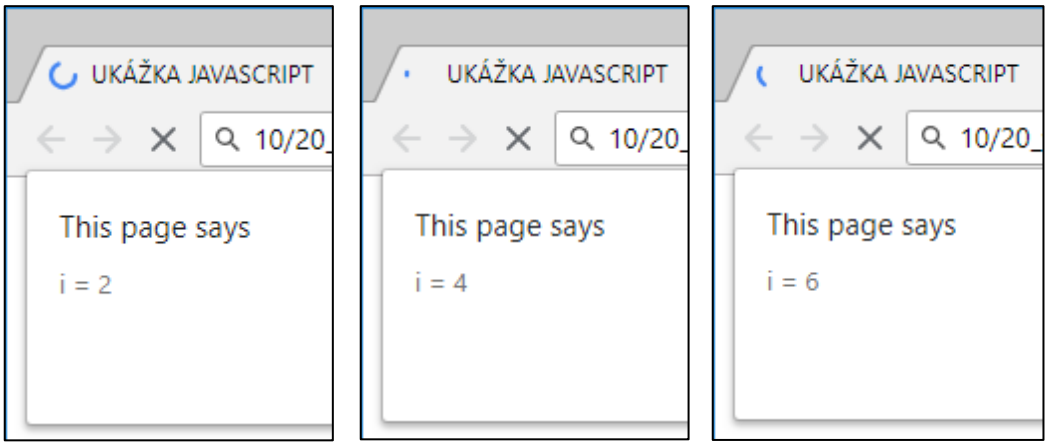
PRÍKLAD 10.22

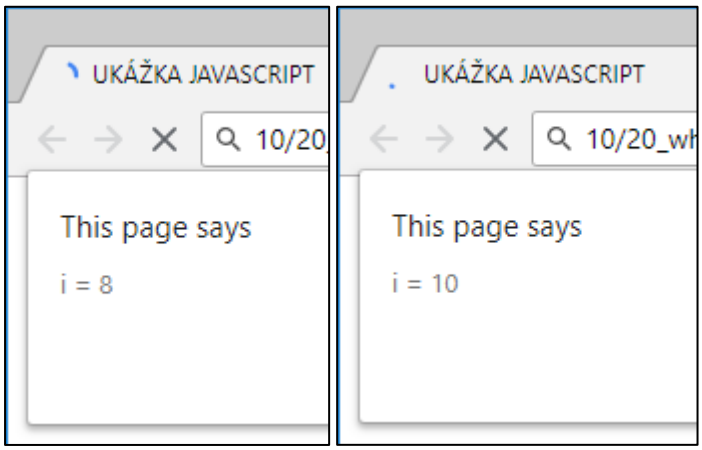
Upravte príklad 10.20 (súbor 10/20_while.html) tak, aby ste použili namiesto cyklu s podmienkou na začiatku cyklus `for`.

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, podčiarniky, 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

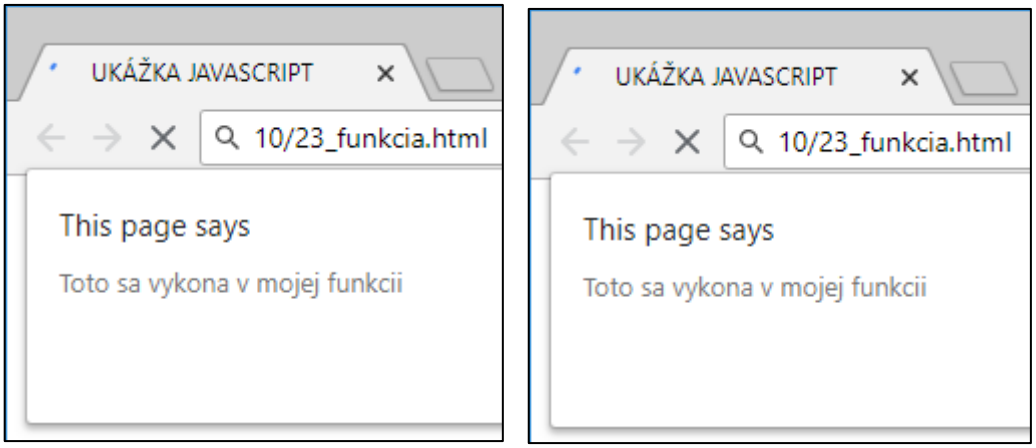
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte funkciu s názvom `mojaFunkcia()`. V tele funkcie vytvorte premennú `text`, ktorej pridajte hodnotu `Toto sa vykona v mojej funkcii`. Hodnotu premennej zobrazte v dialógovom okne.

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`*.

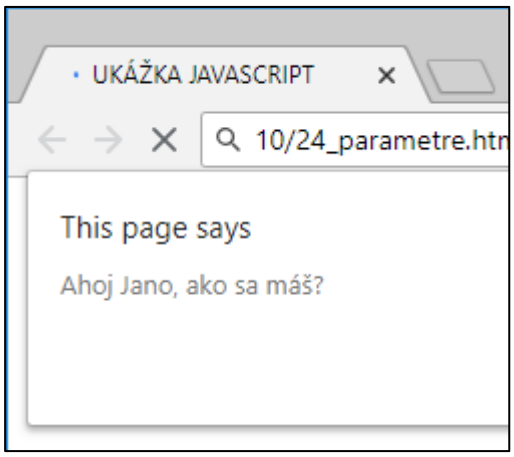
Pod volaním funkcie vytvorte premenné:

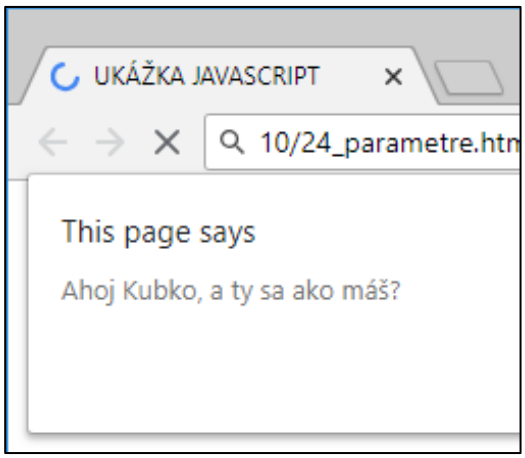
- `ja` s hodnotou `Kubko`,
- `pozdrav` s hodnotou `a ty sa ako máš?`

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

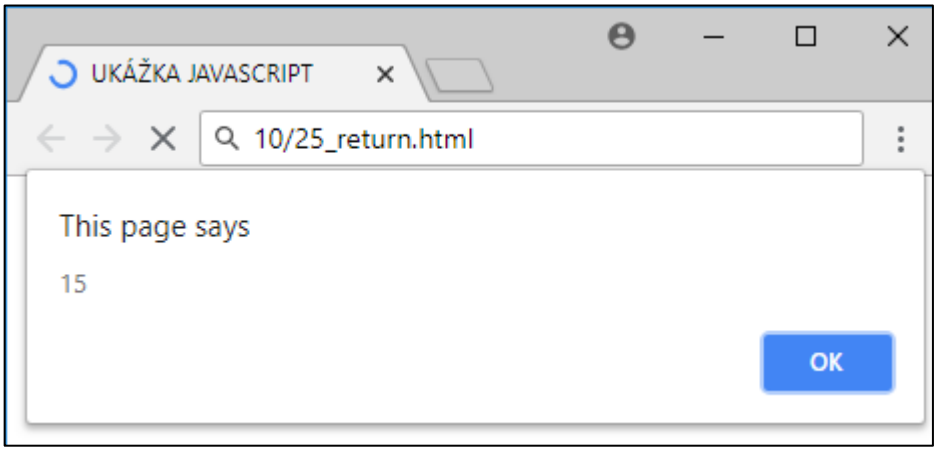
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

- `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

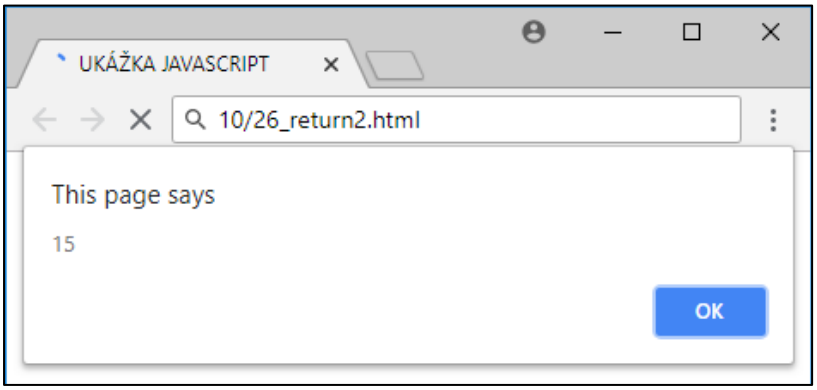
10/26_return2.html

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:

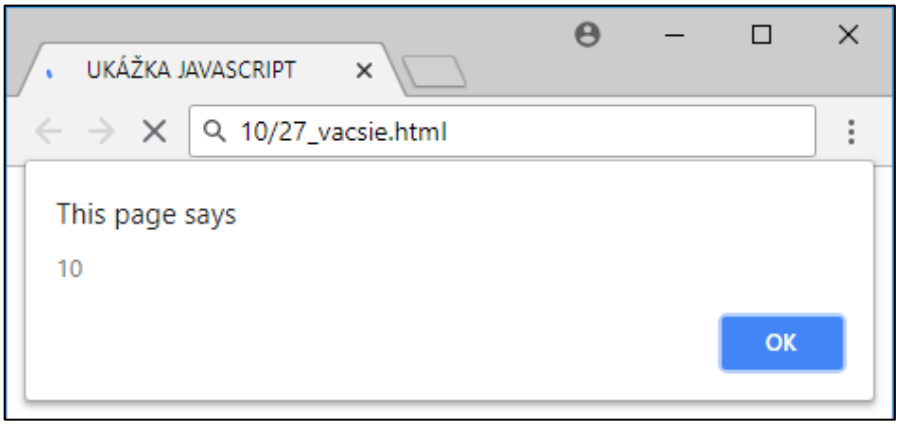
- `vysledok` s hodnotou – zavolaním funkcie `vacsieCislo(10,5)` .

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 slovo `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:

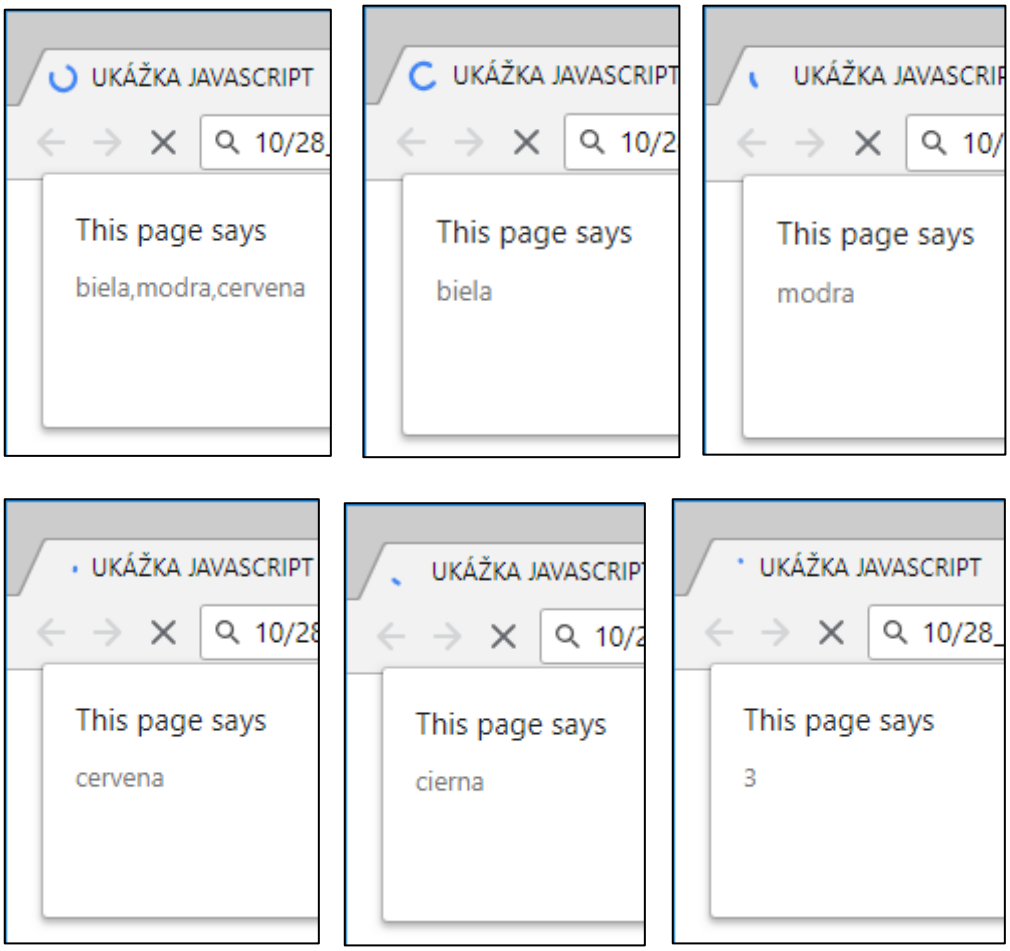
- všetky farby,
- prvú farbu,
- druhú farbu,
- tretiu farbu.

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:

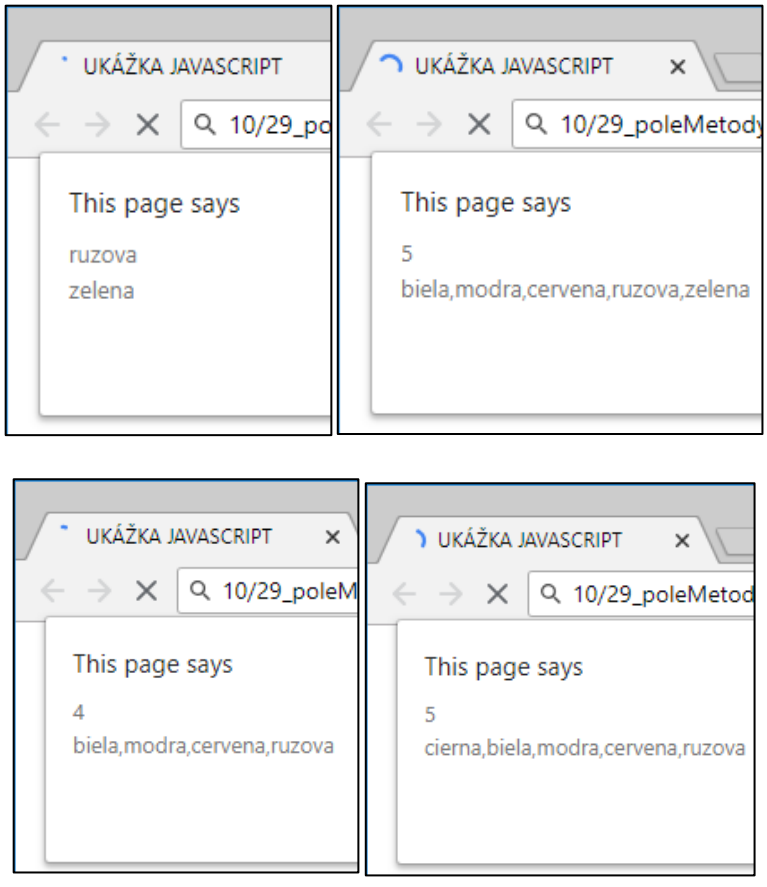
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena`.

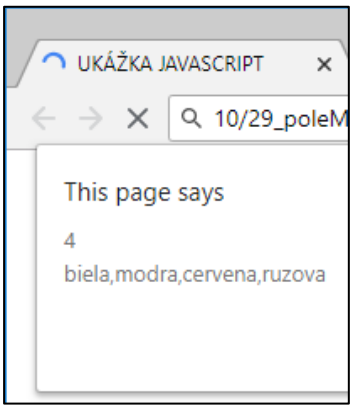
Vykonajte nasledujúce operácie s poľom:

- 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

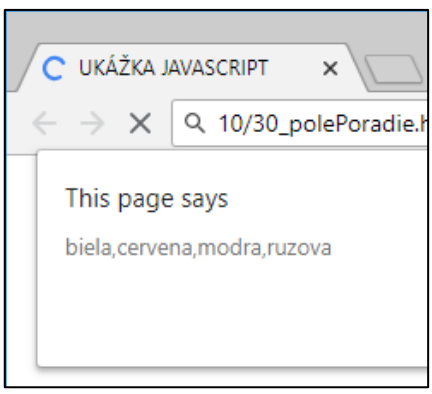
Vytvorte HTML stránku so základnou štruktúrou. V časti `<script>` vytvorte premennú:

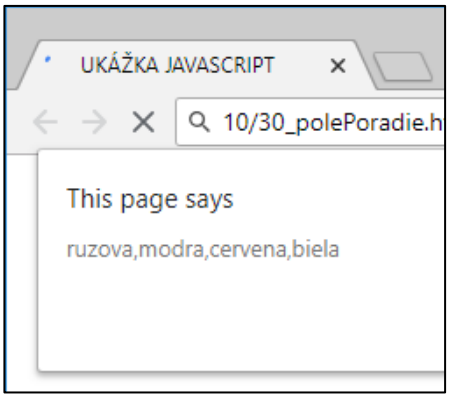
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena, ruzova`.

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:

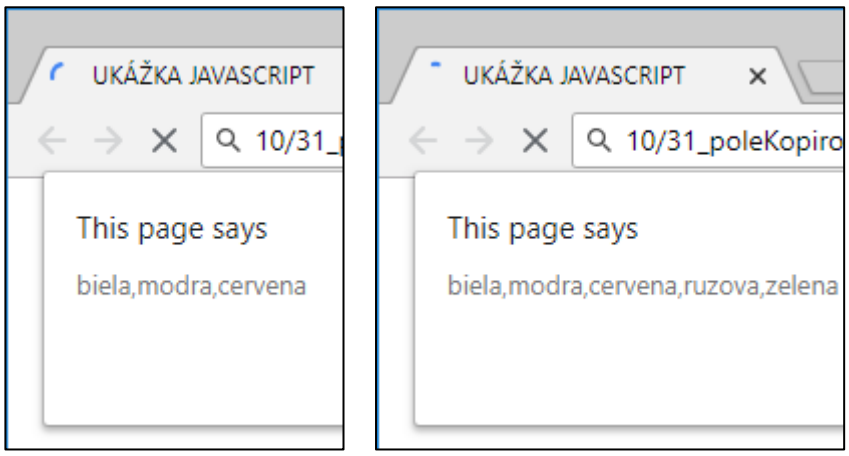
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena`.

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:

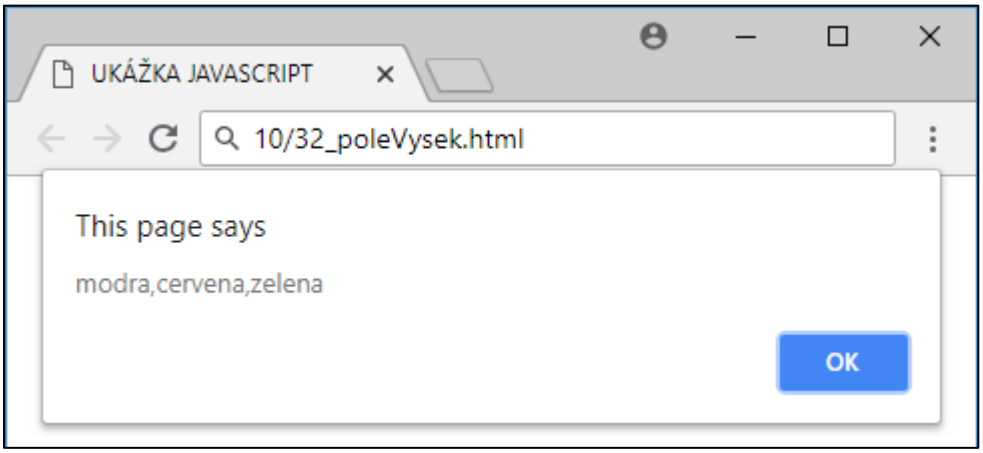
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena, zelena, ruzova`.

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:

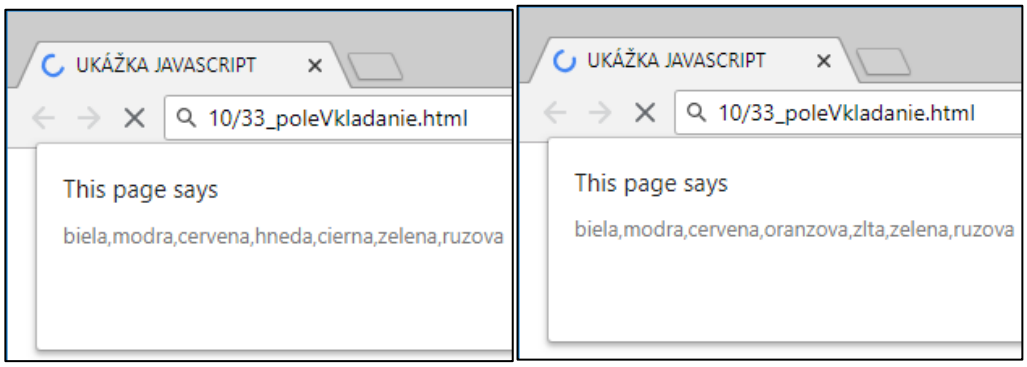
- `farby`, ktorá bude obsahovať ako hodnotu pole. Na začiatku inicializujte hodnotu poľa na tri farby: `biela, modra, cervena, zelena, ruzova`.

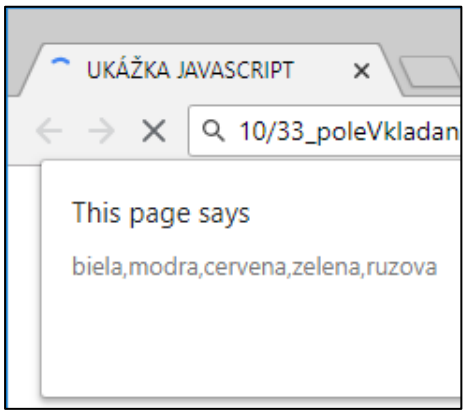
Vykonajte nasledujúce operácie s poľom:

- 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)`.

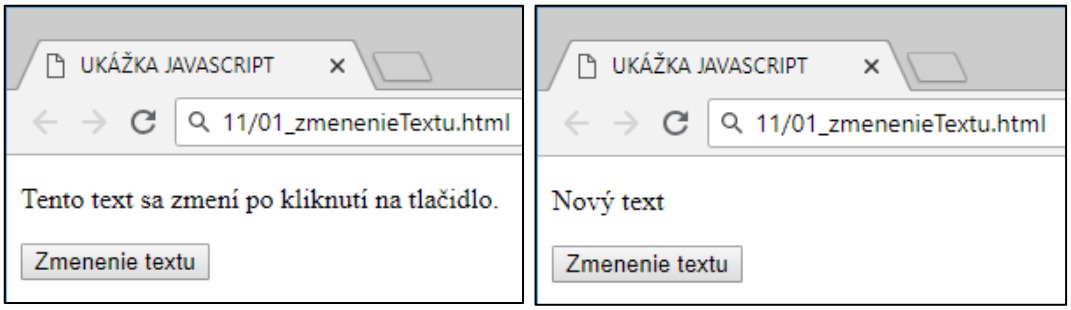
PRÍKLAD 11.1

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`.

11/01_zmenenieTextu.html

```
<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";
```

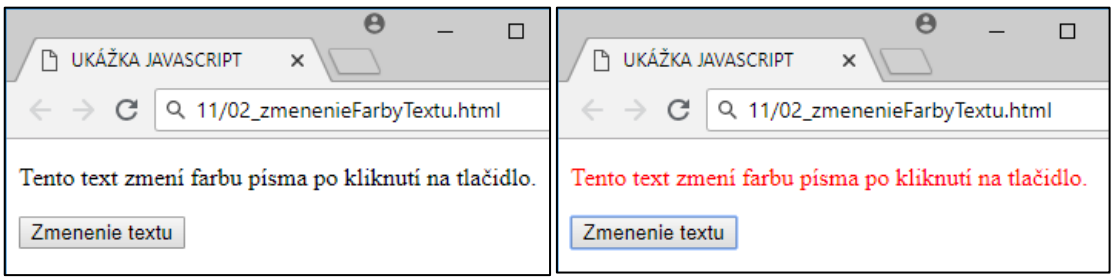
11/02_zmenenieFarbyTextu.html

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:



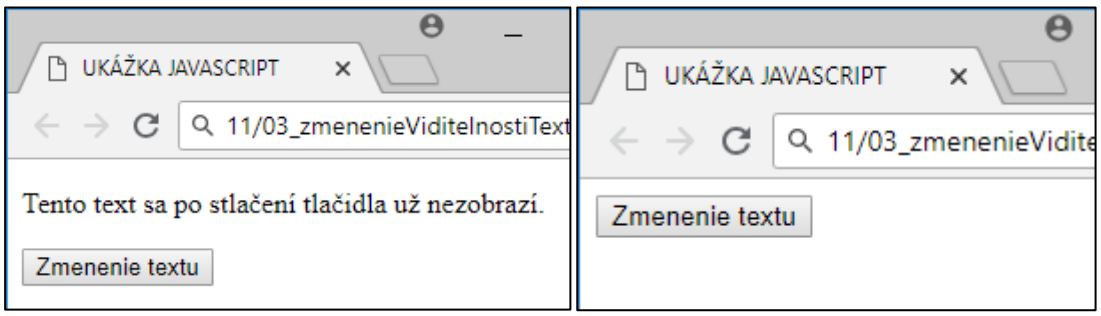
PRÍKLAD 11.3

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';
```

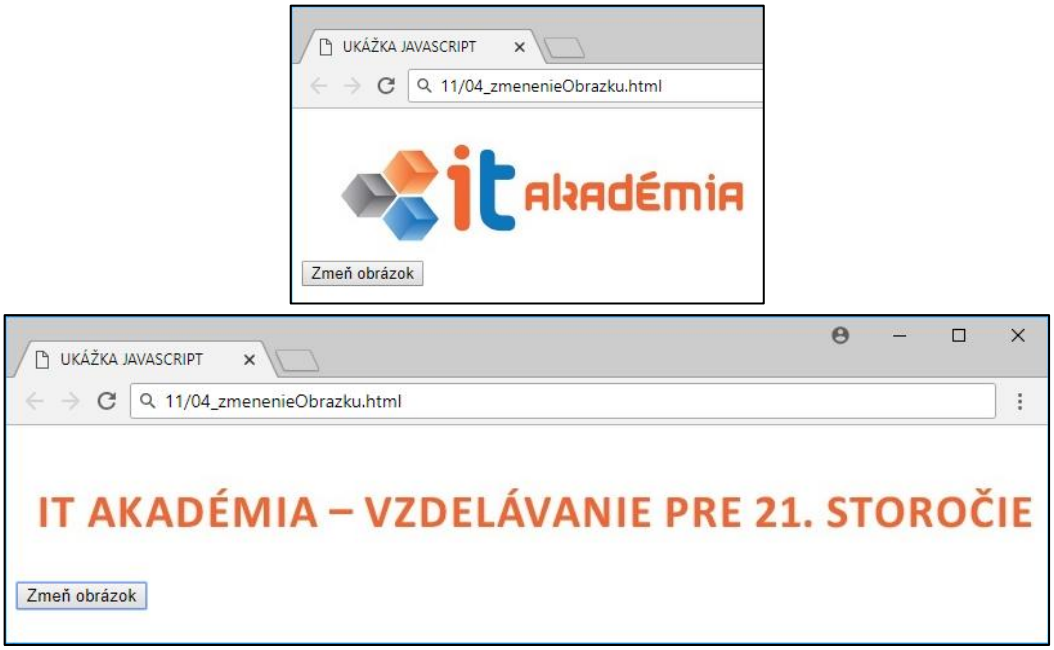
PRÍKLAD 11.4

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";
```

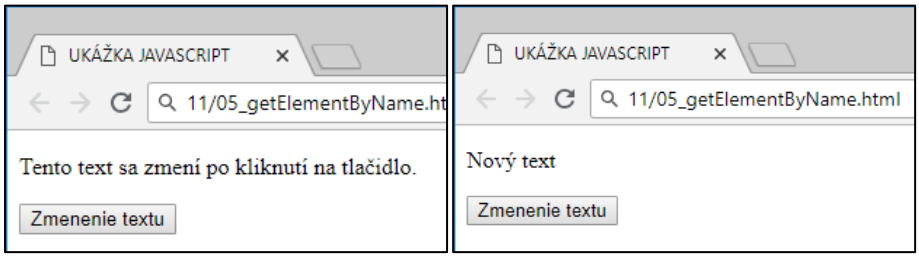
11/05_getElementByTagName.html

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

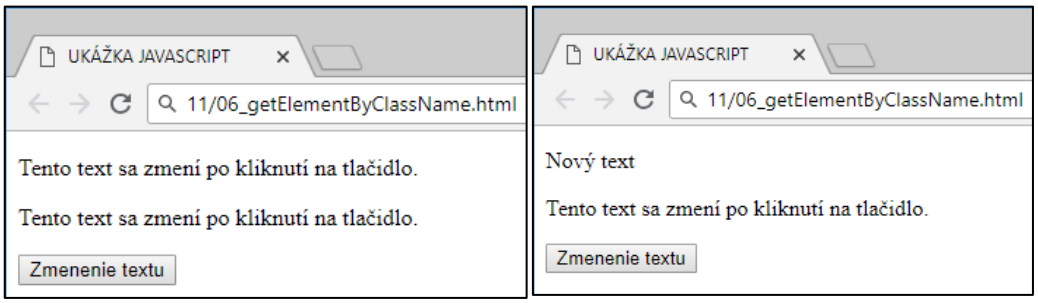
Vytvorte stránku, ktorá bude obsahovať:

- 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:



TODO: pridanie a odstránenie HTML elementov

11.2 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.

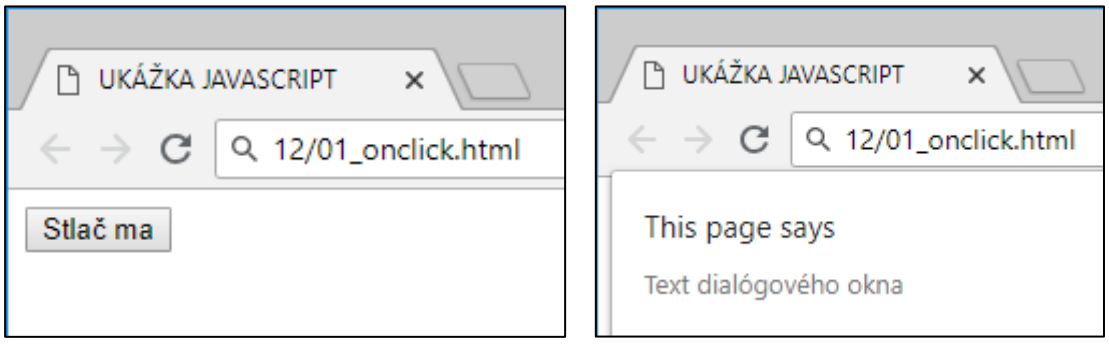
PRÍKLAD 12.01

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 `&quot;`):

```
<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>
```

12/02_prihlaseni.html

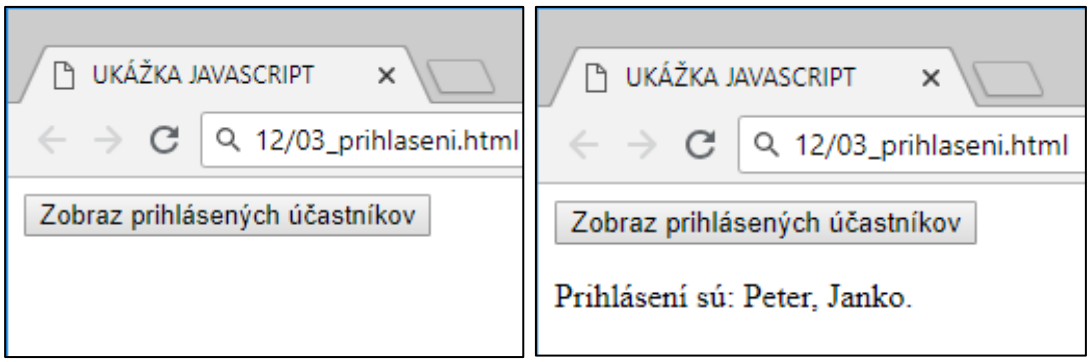
PRÍKLAD 12.02

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 12.03

12/03_onchange.html

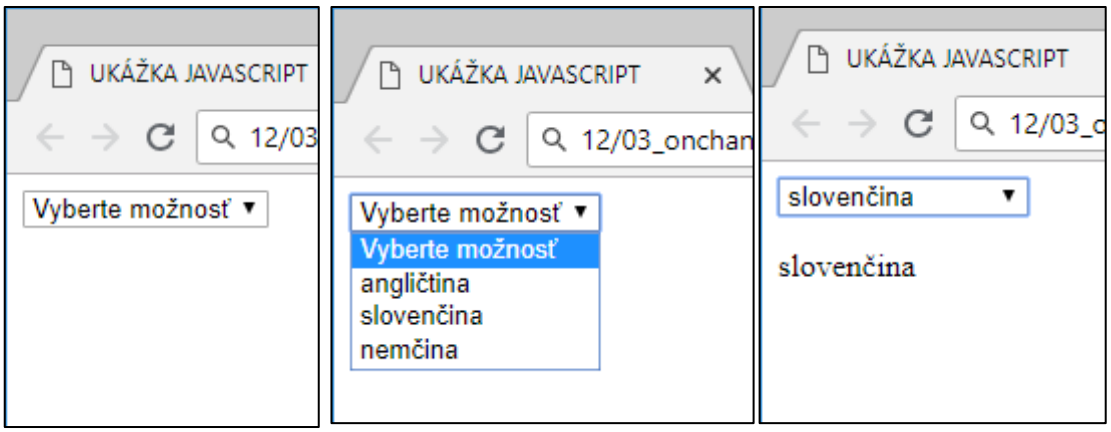
Vytvorte stránku, v ktorej bude vysúvací list `<select>` s hodnotami:

- 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.

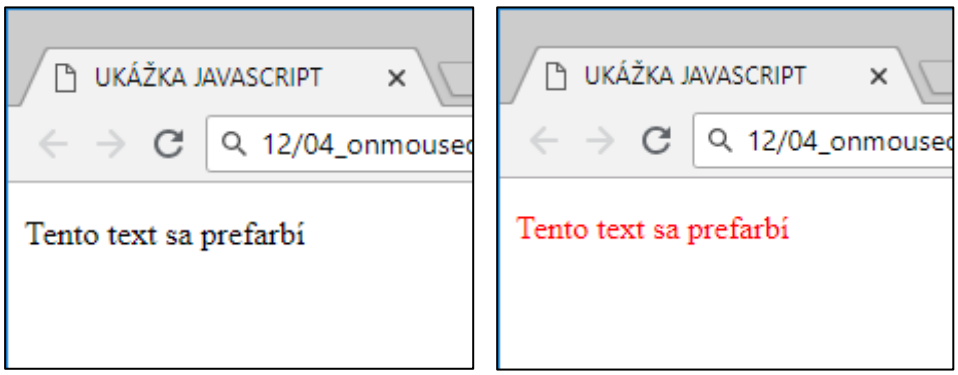
Udalosť `onmouseout` sa spustí, keď sa kurzor myši presunie preč z elementu, ktorý má nastavený tento atribút na iný element.

12/04_onmouseover.html

PRÍKLAD 12.04

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`.

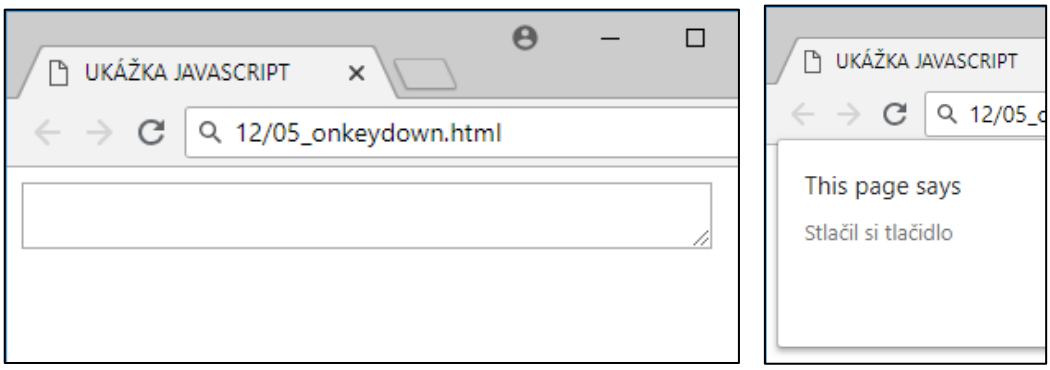
PRÍKLAD 12.05

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`.

12/05_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>`.

12/06_onload .html

PRÍKLAD 12.06

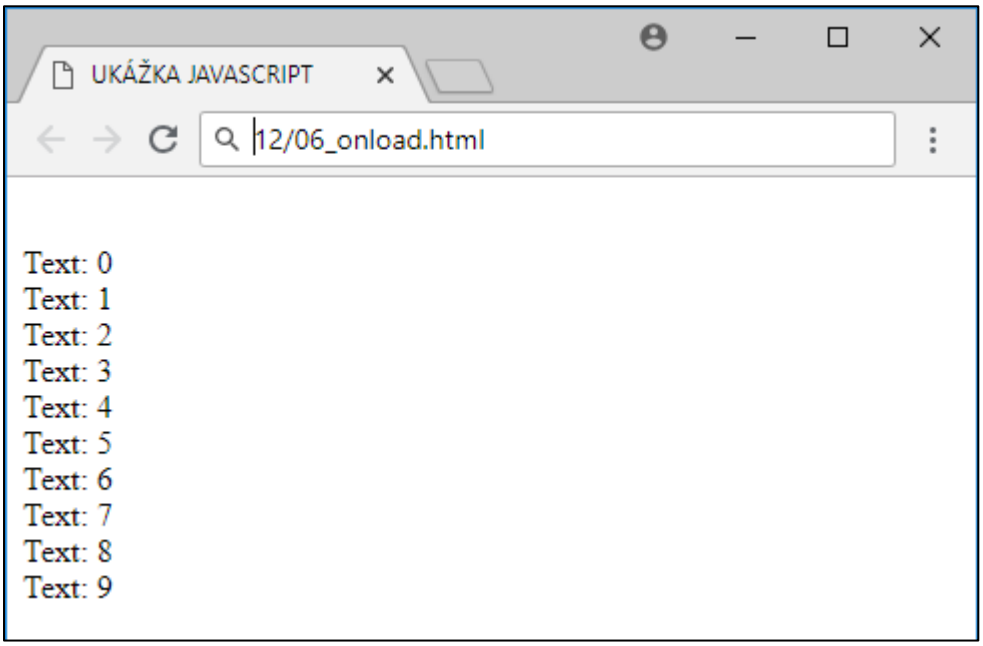
Vytvorte stránku, v ktorej ku značke `<body>` pridajte atribút `onload` s hodnotou zavolaním funkcie `vypisText()`. Do stránky pridajte paragraf s *id text*. V časti `script` vytvorte funkciu `vypisText()`. V tejto funkcii vypíšte 10-krát pod sebou text:

```
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

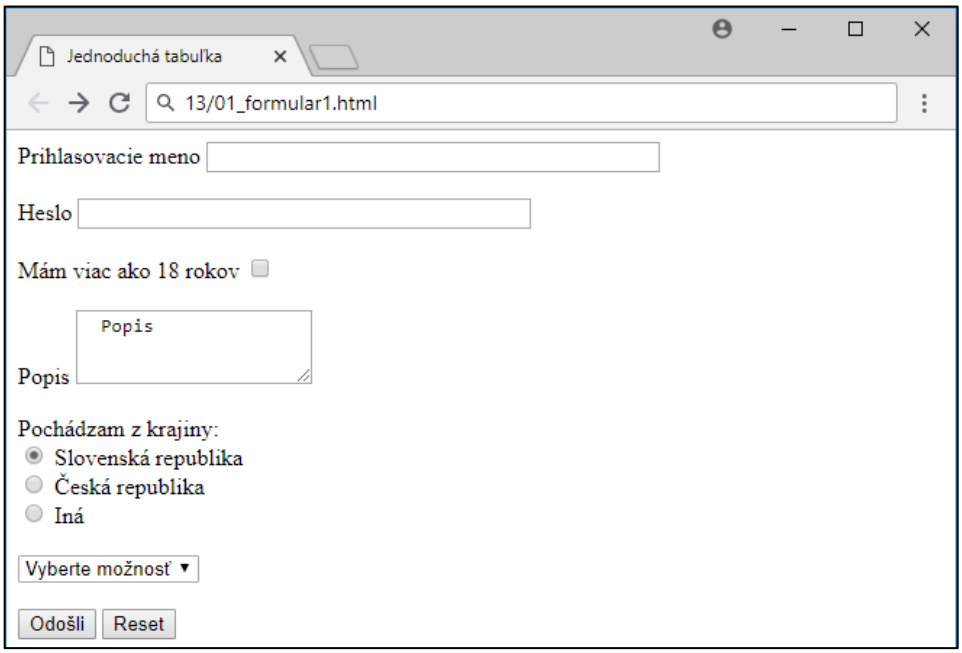
Vytvorte HTML stránku so základnou štruktúrou. V časti `body` vytvorte formulár, ktorý bude obsahovať:

- 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

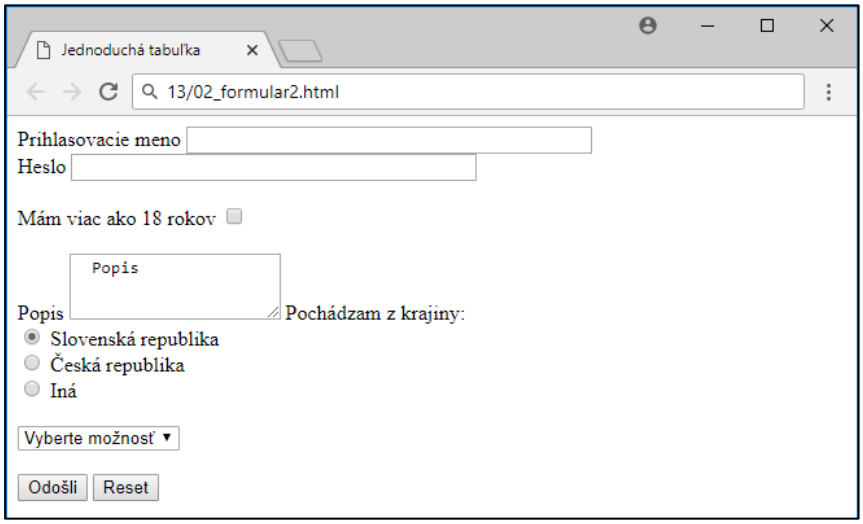
PRÍKLAD 13.02

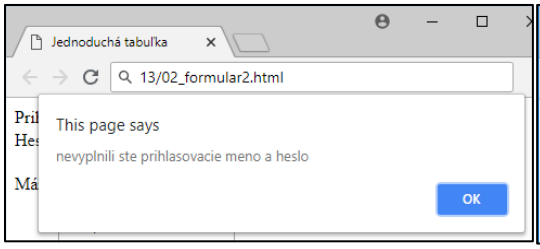
Otvorte si predchádzajúci formulár 13.1 (súbor *13/01_formular1.html*). Do stránky vložte párovú značku `script`, do ktorej pridajte validáciu, či sú zadané hodnoty prihlasovacie meno a heslo:

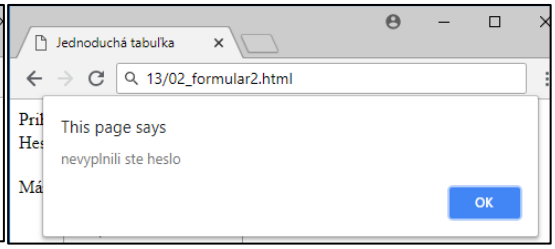
- Ak nie je vyplnené prihlasovacie meno, tak vypíš v dialógovom okne upozornenie `nevyplnili ste prihlasovacie meno`.
- Ak nie je vyplnené heslo, tak vypíš v dialógovom okne upozornenie `nevyplnili ste heslo`.
- Ak nie je vyplnené prihlasovacie meno a súčasne heslo, tak vypíš v dialógovom okne upozornenie `nevyplnili ste prihlasovacie meno a heslo`.
- Ak je vyplnené prihlasovacie meno a heslo, tak vypíš v dialógovom okne text `Vitajte prihlasovacieMeno`.

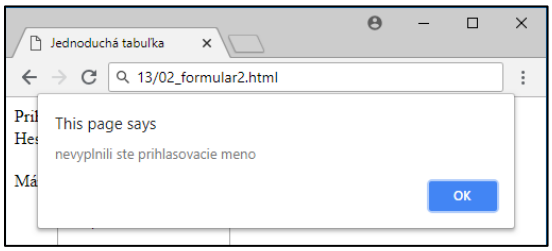
```
<script>
function validaciaRegistracie()
{
if (document.formular.prihlasovacieMeno.value == "" &&
document.form.heslo.value == "")
     alert("nevyplnili ste prihlasovacie meno a heslo");
else if (document.formular.prihlasovacieMeno.value == "")
     alert("nevyplnili ste prihlasovacie meno");
else 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>
```

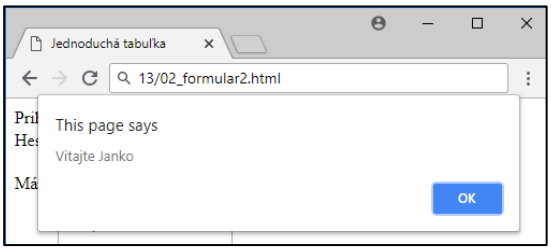
Tento kód zobrazí stránku:











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
```

13/03_formular3.html

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`.

```
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:

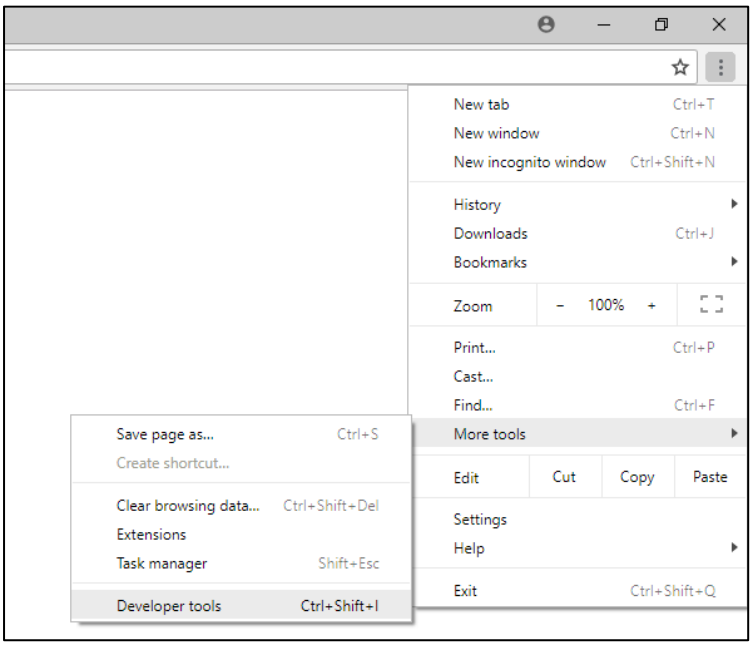
1) Postupovať po častiach – začať kontrolovať jednoduchšie veci a postupne pokračovať kontrolou náročnejších častí. Robiť menšie zmeny v kóde a postupne testovať.
2) Zakomentovať časť kód a postupne odkomentovávať jednotlivé časti, kým sa neobjaví problém.
3) Použiť vývojárske nástroje na preskúmavanie 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).

Chrome

Vývojárske nástroje pre prehliadač Chrome sú dostupné v menu prehliadaču → `More tools`“→ `Developer tools`. Pre spustenie nástrojov je tiež možné použiť klávesovú skratku CTRL+SHIFT+I.

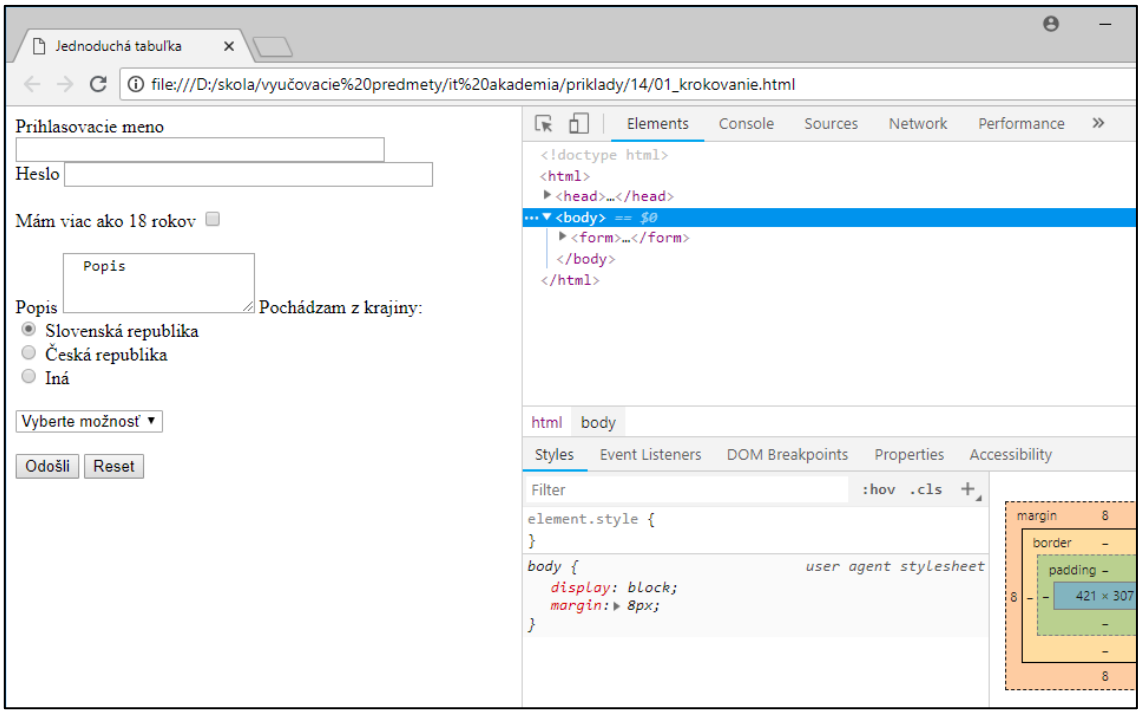


Po spustení vývojárskych nástrojov sa nám sprava vysunie okno, ktoré je vertikálne rozdelené na dve časti (záleží podľa veľkosti obrazovky, môže byť rozdelené aj horizontálne). Prvá časť obsahuje okno so záložkami:

- 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
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

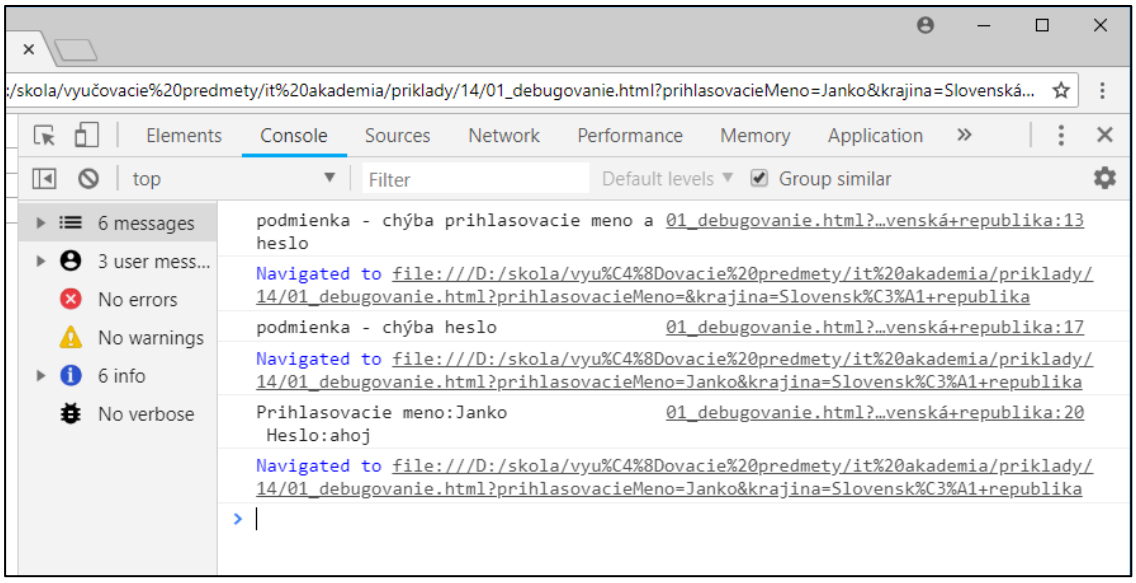
Otvorte si príklad 13.3 (súbor *13/03_formular3.html*). Do formuláru pridajte v časti `<script>` kód, ktorý bude vypisovať do konzoly správy:

- 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()`).

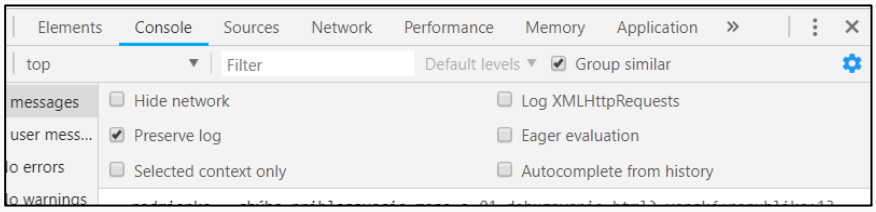
```
<script>
  function loginForm() {
   if ((document.forms[0].elements[0].value == "") &&
     (document.forms[0].elements[1].value == "")) {
      console.log("podmienka - chýba prihlasovacie meno a
      heslo");
   }
   else if (document.forms[0].elements[0].value == "") {
      console.log("podmienka - 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.

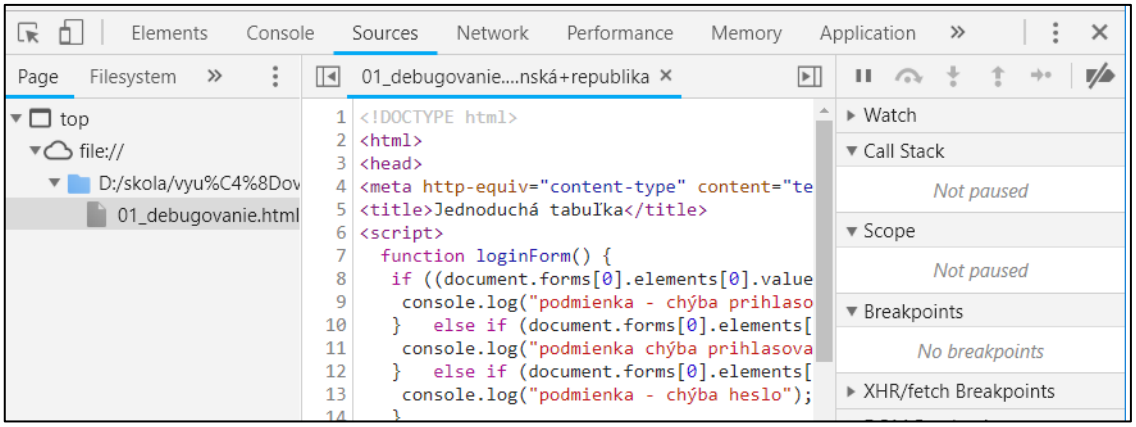
Možnosť vypisovania správ do konzoly je vhodné používať spolu s krokovaním (nastavovaním tzv. bodov prerušenia (breakpoints). V tomto prípade sa správa zobrazí na čas, dokým nestlačíme tlačidlo ukončenie krokovania (prípadne pokračovať na ďalší bod prerušenia).

ÚLOHA 14.1

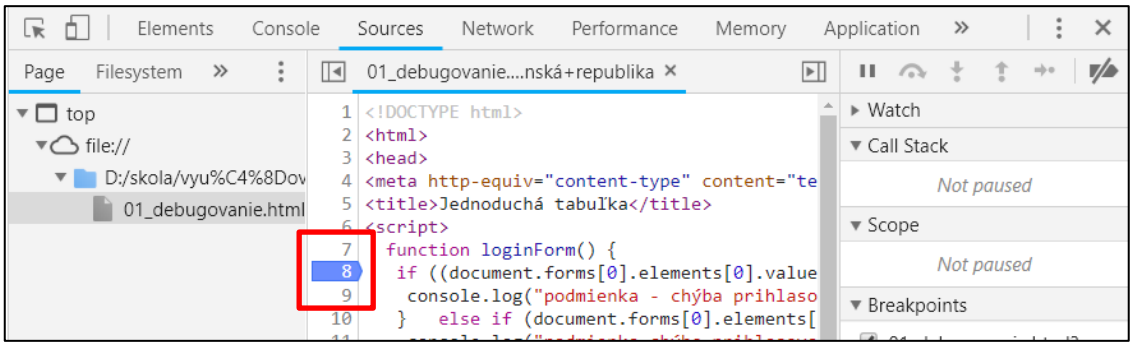
Upravte príklad 13.1 tak, aby ste nahradili všetky funkcie `alert()` výpisom správ do konzoly.

14.2.2 Nastavovanie bodov prerušenia

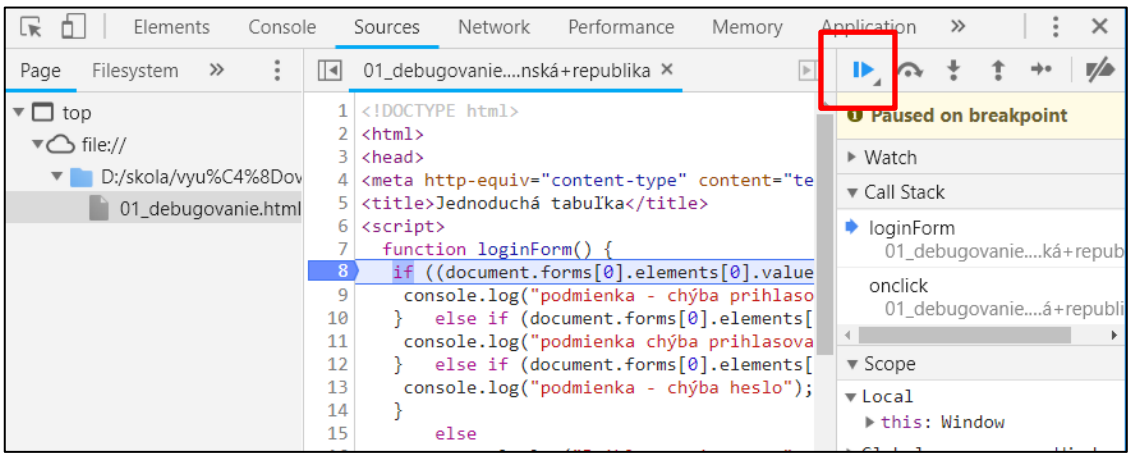
Vypisovanie správ, hodnôt do konzoly môže pomôcť identifikovať chybu. V prípade, že nechceme zasahovať do zdrojového kódu – pridávať a následne odstraňovať metódy na výpis do konzoly, môžeme nastaviť tzv. body prerušenia (angl. breakpoints). Tieto body prerušenia nastavujeme priamo v nástroji vývojára. 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).

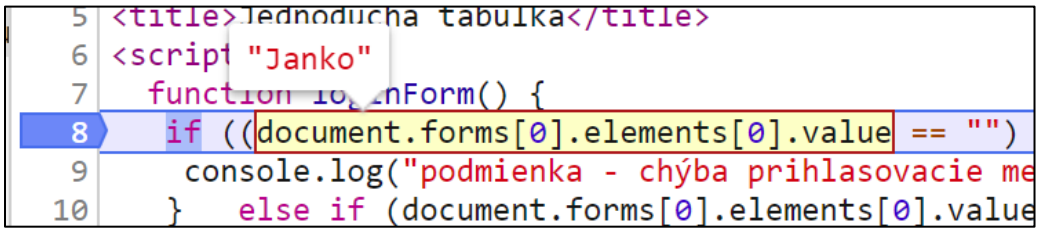
Okrem možnosti pokračovať vo vykonávaní zdrojového kód máme na výber:

- 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");
```

PRÍKLAD 14.2

14/02_debugger.html

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.

15 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 vrámci 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 je aj o 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 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 a 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 nejak 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:

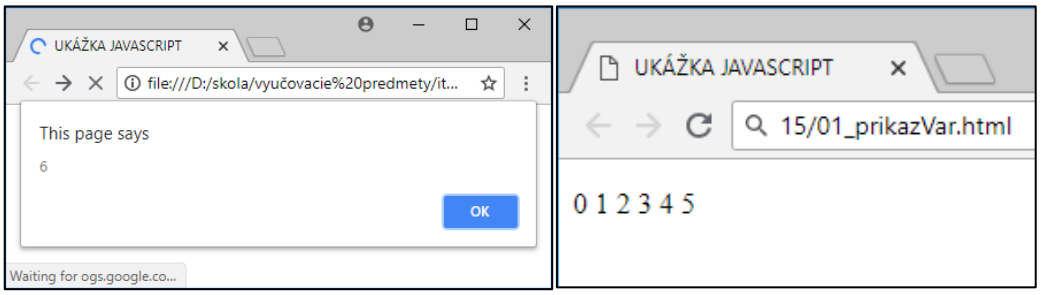
- `body` vložte odstavec s atribútom `id` `cisla`,
- `<script>` vložte cyklus s pevným počtom opakovaní (`for`), ktorý bude postupne vypisovať čísla od `0` po `5`, čísla budú oddelené medzerou,
- za cyklom `for` zavolajte funkciu na zobrazovanie dialógového okna `alert()`, ktorá bude obsahovať parameter 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`.



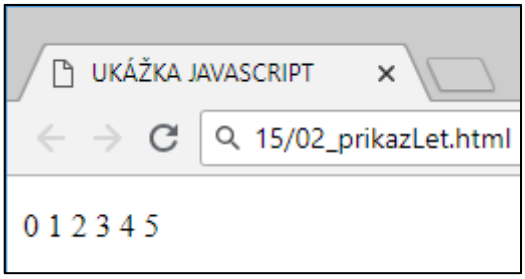
15/02_prikazLet.html

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
```