Základy JavaScriptu

Co je JavaScript?

- JavaScript je *multiplatformní*, *objektově orientovaný skriptovací jazyk*, který se nejčastěji používá k programování dynamických a interaktivních WWW stránek.
- Jeho syntaxe patří do rodiny jazyků C, C++ a Java. Slovo Java je však součástí názvu pouze z marketingových důvodů.
- JavaScript byl vyvinut v 90. letech ve společnosti Netscape; v současné době je využívána standardizovaná verze jazyka, které je oficiálně pojmenovaná jako **ECMAScript**.
- Jde o skriptovací jazyk, který je obvykle interpretován na straně webového klienta („*on client side*“), tj. prohlížečem. Liší se tak od jiných interpretovaných skriptovacích jazyků (např. PHP a ASP), které se spouštějí na straně serveru („*on server side*“) ještě před stažením z Internetu.
- JavaScript dodržuje jistá bezpečnostní omezení (např. nelze v něm pracovat se soubory), aby nemohlo být ohroženo soukromí uživatele.
- Důležité upozornění: JavaScript může být v prohlížeči z bezpečnostních důvodů zakázán a není tedy vhodné podmiňovat zobrazení důležitých informací podporou JavaScriptu!

Využití JavaScriptu

- Stal se programovacím nástrojem pro tvůrce webových stránek – umožňuje jim vytvářet **interaktivní stránky (DHTML** = *dynamické HTML***)**, které reagují na požadavky uživatele (nejčastěji události vyvolané myší, klávesnicí apod.).
- Díky JavaScriptu lze aplikovat objektový přístup k webové stránce a využívat možností, které nabízí tzv. **DOM** – *Document Object Model* (objektový model stránky). Jednotlivé HTML elementy jsou v tomto případě pojímány jako samostatné objekty a prostřednictvím JavaScriptu můžeme měnit jejich vlastnosti a ovlivňovat jejich chování.
- JavaScript bývá také často využíván pro **ověřování (validaci) dat** zadávaných uživateli do webových formulářů.
- Pomocí JavaScriptu můžeme **detekovat typ uživatelova prohlížeče** a přizpůsobit podobu stránek konkrétní platformě.
- JavaScript je nezbytnou součástí **technologie AJAX**, která umožňuje stránkám komunikovat se serverem a zobrazovat uživateli nový obsah či provádět různé dílčí změny (např. zobrazení okamžitých náhledů na zboží v internetovém obchodě, využití našeptávače ve vyhledávačích apod.), aniž je nutné znovu načítat celou stránku.
- V současnosti se stále častěji využívají javascriptové knihovny typu **Prototype** a **jQuery**. Jejich cílem je usnadnit práci s DOM a skrýt některé rozdíly mezi prohlížeči. Příkladem mohou být např. interaktivní formulářové prvky pro výběr data pomocí kalendáře, WYSIWYG editory, inteligentní seznamy apod.

Vložení JavaScriptu do webové stránky

1. **Do párového elementu <script>**

```
<script type="text/javascript">
<!--
document.getElementById("demo").innerHTML='Ahoj';
//-->
</script>
```

Symbolický komentář uvnitř elementu <script> zabrání zobrazení příkazů v prohlížečích, které JavaScript nepodporují.

2. **Jako reakce na události**

```
<p onmouseover="this.style.colorBackground='yellow'"
onmouseout="this.style.colorBackground='white'">Odstavec</p>
```

3. **Do hypertextových odkazů**

```
<a href="javascript:alert('Ahoj');">Odkaz</a>
```

4. **Do externích souborů s příponou .js**

   Skripty uložené v externích souborech připojíme k webové stránce pomocí značky <script> a atributu src.

```
<script type="text/javascript" src="soubor.js"></script>
```

JavaScript: události a odkazy

Reakce na událost

- kód JavaScriptu může být vložen také jako **atribut elementu**
- většinou jde o **reakci na událost**, vyvolanou činností uživatele (pohyb myši, stisk klávesy, kliknutí na tlačítko, načtení stránky ...)

Ošetření události – pohyb myší

```
eover="alert('Myš na
obrázku')" />
<img src="o.jpg" onmouseout="alert('Myš vně
obrázku')" />
<img src="o.jpg" onclick="alert('Klik na obrázek')" />
```

Ošetření kliknutí na odkaz

- kód JavaScriptu může nahradit běžnou reakci kliknutí na odkaz
- na místo URL adresy je vložen kód uvozený `javascript:`

Ošetření kliknutí na odkaz

```
<a href="javascript: alert('Reakce na odkaz')">Odkaz</a>
```

Použijeme-li více příkazů, ukončujeme je středníky

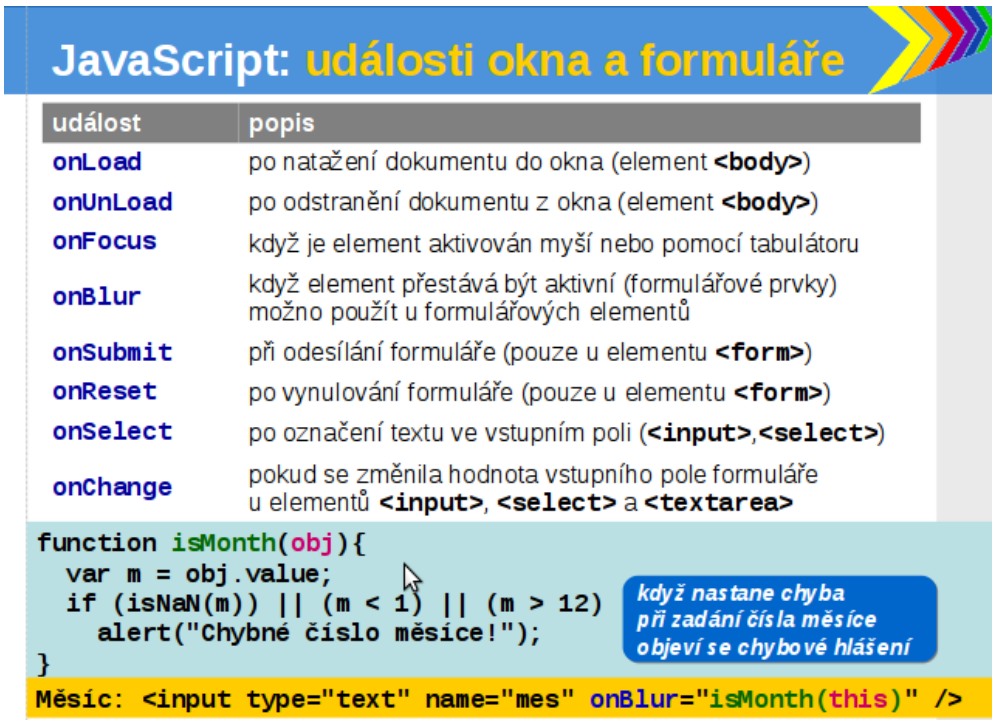
```
<a href="javascript:
x=Math.SQRT2(9);alert(x);">Odkaz</a>
```

JavaScript: události myši a klávesnice

událost	popis
onClick	po kliknutí myší na element
onDblClick	dvojité kliknutí myší
onMouseDown	stisknutí tlačítka myši nad elementem
onMouseUp	uvolnění tlačítka myši nad elementem
onMouseOver	přesunutí myši nad element
onMouseMove	pohybu myši nad elementem
onMouseOut	odsunutí myši z elementu
onKeyPress	po stisku a uvolnění tlačítka na klávesnici
onKeyDown	po stisku tlačítka na klávesnici
onKeyUp	po uvolnění tlačítka na klávesnici

```
function Klavesa(e) {
  k=(window.Event) ? e.modifiers & Event.SHIFT_MASK:e.shiftKey;
  if (k) alert("Byla stisknuta klávesa SHIFT");
}
<body onkeypress = "Klavesa(event)">
```

při stisku klávesy SHIFT se objeví informační zpráva



Zásady psaní kódu v JavaScriptu

- V JavaScriptu je nutné rozlišovat malá a velká písmena – např. příkazy `zmenBarvu()` a `zmenbarvu()` volají dvě odlišné funkce. Při vytváření identifikátorů vlastních proměnných a funkcí je vhodné dodržovat programátorské konvence; název začínáme malým písmenem, víceslovný název píšeme bez mezer, ale každé nové slovo zvýrazníme velkým počátečním písmenem – např. `var obvodRovnoramennehoTrojuhelnika`. Ve vlastních identifikátorech nepoužíváme klíčová slova, mezery, speciální znaky nebo diakritiku.
- Příkazy píšeme obvykle na samostatné řádky. V případě, že je na jednom řádku více příkazů, oddělujeme je středníky. Ukončení příkazu středníkem je však vhodné dodržovat vždy.
- Chceme-li vytvořit blok příkazů (tj. například v případě podmínky provést dva nebo více příkazů), používáme složené závorky:

```
<script type="text/javascript">
<!--
var x = 4;          // inicializace proměnné x
var barva = "red";  // inicializace proměnné barva
if (x > 0)
{
     // vložení odstavce do webové stránky
     document.write("<p id='vysledek'>X je větší než 0</p>");
     /* změna barvy odstavce, který má id="vysledek" */
     document.getElementById("vysledek").style.color = barva;
}
//-->
</script>
```

- Literáry složené ze znaků (znakové řetězce typu String) můžeme v JavaScriptu podle okolností zapisovat mezi dvojité nebo jednoduché uvozovky.
- Příkaz `document.write()` slouží k vkládání HTML kódu do webové stránky; ta je zastoupena objektem `document`.

POZOR - zapíšeme-li příkazem **document.write** do uzavřeného dokumentu (což je například již zobrazený dokument), jeho obsah se tím přepíše!

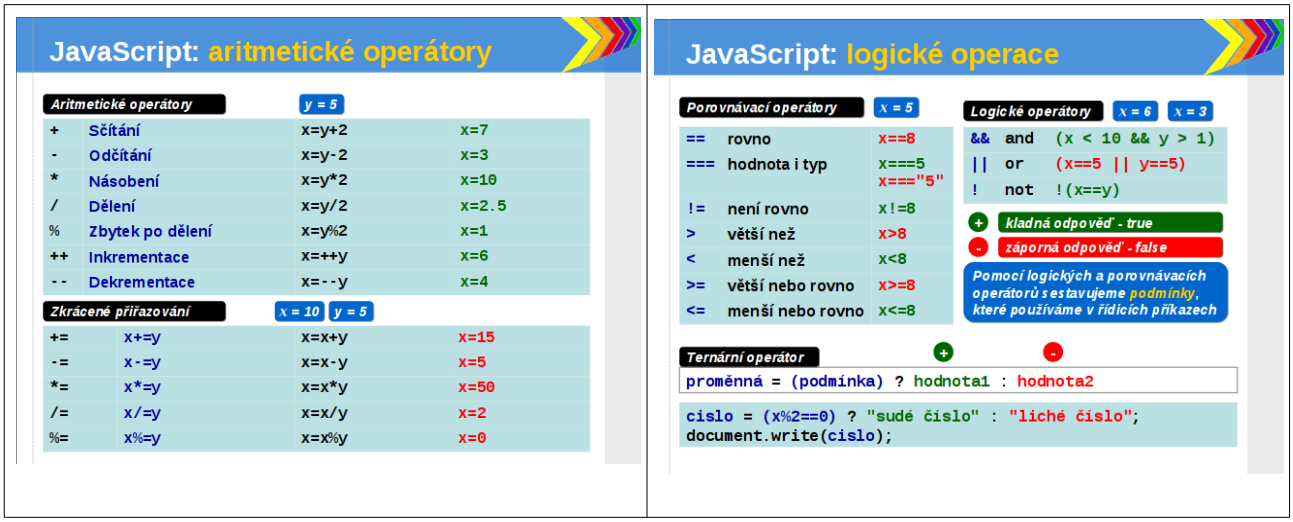
- Jednořádkové komentáře jsou vyznačeny **//**, víceřádkové **/* *komentář* */**.
- Pokud je nějaký speciální znak, který JavaScript interpretuje, potřeba zapsat do stránky, musí se využít tzv. escape sekvence. Před inkriminovaný znak se napíše zpětné lomítko. Nejčastěji se to používá u uvozovek.

```
<img onmouseover="alert(\"Ahoj světe\")">
```

Operátory

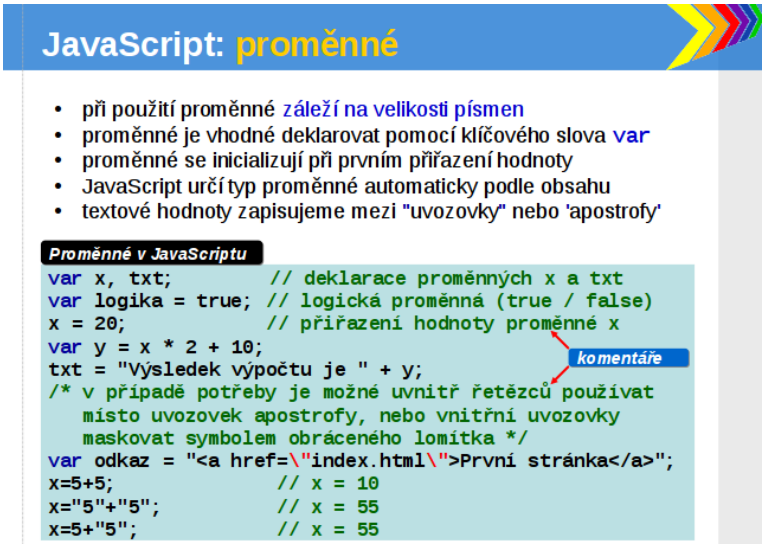
V JavaScriptu se používají operátory podobným způsobem jako v příbuzných jazycích. Např. operátor **%** se používá pro zjištění zbytku po celočíselném dělení. Operátor **+** je možné použít také pro spojování řetězců. Také je možné používat operátorů pro zkrácené přiřazování (např. +=, *= apod.).

```
var a = 15 % 4;     // proměnná a bude obsahovat číslo 3
// do proměnné vysledek bude uložen text "Výsledek = 3";
var vysledek = "Výsledek" + " = " + a;
a *= 4;   // do proměnná a bude vynásobena 4
```



Proměnné

Proměnné jsou inicializovány pomocí klíčového slova **var**. Pokud je proměnná inicializována uvnitř funkce, jde o proměnnou lokální (tj. platnou pouze v rámci dané funkce).



Datové typy

Proměnné mohou být různých datových typů. V JavaScriptu rozlišujeme jednoduché datové typy (integer, float, string, boolean) a složené datové typy (array, object, function).
Integer - kladné nebo záporné celé číslo (např. `x = 10, y = -103`). Číslo je možné zapsat také v osmičkové soustavě (např. **067**) a v hexadecimální soustavě (např. **0xE3**).
Float – desetinné číslo s plovoucí řadovou čárkou. Zapisuje se:

- v podobě čísla s **desetinnou tečkou** (např. `r = 4.167`),
- nebo v **exponenciální formě** (např. `e = 1.06e-3 → 1.06 * 10`$^{-3}$).

String – řetězec znaků (text); ohraničují jednoduché či dvojité uvozovky (např. `txt = "Ahoj"`).
Boolean - logické hodnoty **`true / false`**.
Null – proměnná bez hodnoty (nikoliv číselná 0!) .

Přetypování aneb konverze

O změnu typu proměnné (tzv. konverze) se (u námi používaných typů) většinou nemusíme starat, protože se provádí automaticky (tzv. **implicitní konverze**). Např. při výstupu číselných nebo logických hodnot dojde k okamžité konverzi na řetězec, a podobně při vstupu. Stejně tak při testu podmínky jsou nula a "prázdný" textový řetězec ("") (a také hodnota null) automaticky převedeny na hodnotu false (a jakékoliv jiné hodnoty na true).
Někdy je však třeba provést tzv. **explicitní konverzi**. Například operaci sčítání "+" lze v JavaScriptu použít pro čísla i pro řetězce (kde funguje jako spojování řetězců). Výraz 1+1 dá tedy 2, zatímco výraz "1"+"1" dá "11". Výrazy 1+"1" i "1"+1 dají rovněž 11, protože dojde k automatickému přetypování na řetězec. Toto vše samozřejmě platí, i když uvedené hodnoty jsou hodnotami nějakých dvou proměnných, které sčítáme (nebo spojujeme). Chceme-li provést úpravu typu podle svých potřeb, musíme použít explicitní přetypování:

- Převod čísla na řetězec: metoda **toString** - používá se např. takto:
  `Cislo=3.14; ...; retezCislo=Cislo.toString();`
- Jinou možností je použít trik s **přičtením prázdného řetězce**:
  `retezCislo=Cislo+""`.
- Metoda **toFixed** umožňuje specifikovat zobrazení desetinného čísla na daný počet míst: `Cislo.toFixed(n). Cislo.toFixed()` je zaokrouhlené celé číslo.
- Metoda t**oPrecision** obdobně určuje počet platných cifer, metoda **toExponential** převádí číslo do exponencíální notace. Výsledkem použití všech těchto metod jsou, jak uvedeno, řetězce.
- Převod řetězce na číslo: funkce **parseInt** (na celé číslo), **parseFloat** (na desetinné číslo) vrací numerický počátek řetězce konvertovaný na číselný typ. Například:
  `parseInt("12 opic")` vrátí 12,
  `parseInt("3.14")` vrátí 3,
  `parseFloat("12.34+8")` vrátí 12.34.
- Nezačíná-li konvertovaný řetěz číslem, vrátí obě funkce smluvenou hodnotu **`NaN`** ("*Not a Number*" → nečíslo).
- I zde je alternativní trik, který spočívá v tom, že se od řetězce odečte nula:
  `Cislo=retezCislo-0` (lze i násobit jedničkou, obojí vyvolá implicitní konverzi).

Základní příkazy jazyka

Podmíněné příkazy

If – jednoduchá podmínka

```
if(podmínka) {
příkazy prováděné při splnění podmínky;
}
else {
příkazy prováděné při nesplnění podmínky;
}
```

- Podmínkou musí být výraz, jehož logická hodnota je **true** nebo **false** (pravda nebo nepravda).
- Část **else** je možné vynechat, pokud mají být vykonány příkazy pouze pro kladnou odpověď podmínky.

Ternární operátor (? :)

Ternární operátor umožňuje rychlejší zápis rozhodování v případech, kdy chceme pouze přiřadit hodnotu proměnné.

Syntaxe:

```
proměnná = podmínka ? hodnota1 : hodnota2;
```

Příklad:

```
var vysledek = x > 0 ? "kladné číslo" : "nekladné číslo";
```

Switch – větvená podmínka

Pro větvení do více alternativ existuje příkaz switch

Syntaxe:

```
switch(proměnná) {
    case hodnota1:
         příkaz1;
         break;
    case hodnota2:
         příkaz2;
         break;
    ...
    default:
         příkaz;
}
```

Cyklické příkazy

For – cyklus pro předem určený počet opakování

Cyklus je obvykle určený pro předem daný počet opakování.

Syntaxe:

```
for (počáteční hodnota; podmínka; navýšení) {
  příkazy;
}
```

Příklad:

```
for (i=1; i <= 7; i++) {
  document.write("<h" + i + ">Nadpis " + i +"</h" + i + ">");
  // vypíše vzorek nadpisů na různých úrovních
}
```

While - cyklus s podmínkou na začátku
Sekvence vnitřních příkazů se provádí dokola, dokud platí podmínka. Jakmile podmínka neplatí, blok příkazů se už nevykoná a program bude pokračovat pod koncem sekvence while.
Syntaxe:
```
while (podmínka) {
  sekvence příkazů
}
```
Příklad:
```
var x = 0;
while(x < 30) {
  x += 5;
  document.write("<p>Hodnota x = " + x + "</p>");
  // vypisuje hodnotu x, dokud je nižší než 30
}
```

Do ... while - cyklus s podmínkou na konci
Je-li podmínka nepravdivá, cyklus se ukončí a program jde dál.
Syntaxe:
```
do {
   sekvence příkazů
} while (podmínka)
```

Break a continue
break - předčasně ukončí cyklus while nebo for.
continue - provede skok na začátek cyklu.

Funkce v JavaScriptu

Syntaxe deklarace funkce v JavaScriptu:
```
function jmenoFunkce(parametr, parametr, ...)
{
  tělo funkce
  return návratová_hodnota;
};
```
Příklad:
```
function upozorneni(stranka)
{
  alert("Tímto se dostanete na stránku s názvem " + stranka);
};
```
Volání funkce:
```
jmenoFunkce(hodnota, hodnota);
```
Velmi často se funkce volají na základě událostí dokumentu přímo z HTML kódu, například:
```
<a href="#" onclick="upozorneni('hlavní stránka');">Obsah</a>
```
Při kliknutí na slovo "Obsah" se vyvolá funkce upozorneni() s hodnotou parametru "hlavní stránka". Předtím samozřejmě musí být funkce inicializovaná (v předchozím příkladu).

Pokud funkce vrací hodnotu (deklarace obsahuje return hodnota), dá se funkce volat zápisem
```
proměnná = jmenoFunkce(parametry);
```

Proměnné ve funkci
Proměnná deklarovaná ve funkci klíčovým slovem var je **lokální**. Lokální proměnné jsou

i parametry funkce (to, co je v závorce za jménem funkce). Pokud se ve funkci použije jméno jiné nedeklarované proměnné, jde o proměnnou **globální**.

JavaScript a objekty

JavaScript je jazyk objektový, třebaže nevyužívá všechny možnosti OOP (objektově orientovaného programování).
Z pohledu objektového přístupu k programování JavaScript nabízí:

- tzv. **objektový model prohlížeče** (DOM = Document Object Model) – jednotlivé prvky okna prohlížeče a dokumentu se chovají jako samostatné objekty s vlastnostmi a metodami;
- **hotové („vestavěné“) objekty** (Math, Date, Array, String apod.);
- možnost vytváření **nových vlastních objektů** podle potřeb programátorů.

Základní terminologie objektového programování

Objekt (*object*). Základní pojem v objektovém programování. Objektem může být nejen viditelný prvek na stránce (např. celá stránka, okno, tlačítko, nadpis, titulek, odkaz, obrázek …), ale někdy také speciální abstraktní struktura umístěná v paměti (např. znakový řetězec, objekt datum, pole apod.). Objekt tvoří jeho *vlastnosti* (např. barva, nápis, velikost, umístění …) a *metody* (změna barvy, přesun na nové místo, vrácení informace o aktuálním času …). Především s viditelnými objekty mohou být rovněž spojeny některé *události* (kliknutí myší, stisk klávesy, změna stavu objektu …).

Vlastnost (*property*). Vlastnosti objektů jsou spojeny s konkrétní hodnotou určitého datového typu (např. x-ová souřadnice umístění objektu je celé číslo, nápis tlačítka je text, viditelnost objektu je logická hodnota true nebo false apod.). Hodnota se dá číst nebo zapisovat, některé vlastnosti jsou jen pro čtení, některé jen pro zápis.

Příklad objektových vlastností:

```
alert(window.screenLeft);
/* vypíše umístění okna prohlížeče vůči obrazovce zleva (v
pixelech) */
document.title = "Můj titulek";
// nastaví nový titulek stránky
```

Metoda (*method*). Metody objektů jsou ve své podstatě funkce, které zajišťují provedení určité akce (např. zjištění názvu objektu, změnu polohy objektu apod.). Mnohé z těchto metod-funkcí vracejí hodnotu (jejich názvy obvykle začínají slovem *get*), jiné naopak hodnotu nastavují (jejich názvy obvykle začínají slovem *set*).

Příklady použití metod:

```
window.open('http://www.google.cz','','width=400,height=200');
// otevře nové okno se zadanou adresou a rozměry
window.close();
// uzavře aktuální okno prohlížeče
```

Konstruktor (*constructor*). Jedná se o speciální metodu, jejíž úlohou je vytvoření („zkonstruování“) samotného objektu a přidělení paměťového prostoru pro jeho existenci. Některé objekty mohou být používány teprve po inicializaci – tedy volání konstruktoru. Volání konstruktoru v JavaScriptu zajišťuje klíčové slovo **new** a *název objektu*.

Příklady použití konstruktoru:

```
var d = new Date();
// konstruktor vytvoří nový datový objekt se systémovým časem
var d = new Date(milliseconds);
```

```
// bude vytvořen nový datový objekt podle zadaného údaje
// v milisekundách
```

Událost (event). Zejména vizuální objekty mohou být připraveny reagovat na akci uživatele (případně samotného systému). Tyto akce jsou nazývány událostmi a ošetřovány (samozřejmě uzná-li to programátor za nutné) pomocí tzv. ohlasových metod. K typickým událostem v JavaScriptu patří reakce na kliknutí myši (onmouseclick), pohyb kurzoru myši nad aktivní oblastí objektu (onmouseover), opuštění této aktivní oblasti (onmouseout) atd.

Podobjekty. Objekty mohou být komponovány (složeny) z menších objektů – tzv. *podobjektů*. Nadřízený objekt v takovém případě vystupuje symbolicky v roli **rodiče** (*parent*) a podřízený objekt v roli **dítěte** (*child*).

Příklady použití podobjektů:

```
window.history.back();
```

Je zápis příkazu, který funguje stejně jako tlačítko zpět v prohlížeči. Objekt *window* má podobjekt *history*. History má metodu *back()*. Je to metoda, protože to něco dělá (vrací historii).

```
window.location.href = "http://www.google.cz";
```

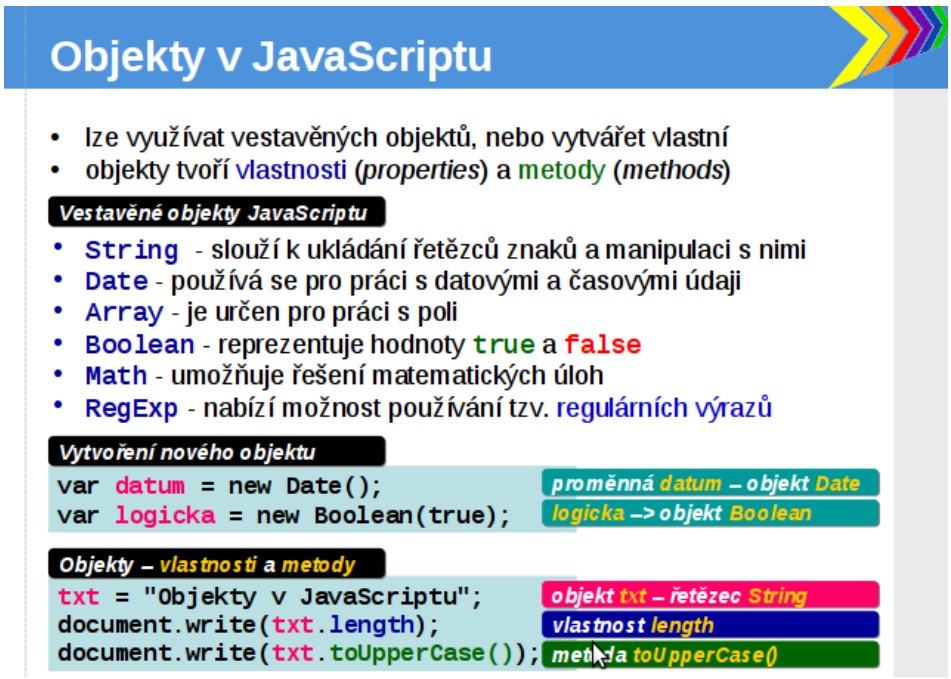
Načte do okna prohlížeče novou stránku. Objekt *window* je okno prohlížeče, který má podobjekt *location* a ten má vlastnost href .

Zpřístupnění podbjektů, vlastností a metod

Z výše uvedených příkladů je již asi jasné, že pro zpřístupnění vlastností, metod i podobjektů se v JavaScriptu používá syntaxe s oddělovačem v podobě **tečky**.
Kromě toho můžeme použít i příkaz **with**.

With - příkaz pro jednodušší práci s objekty
Používáme-li opakovaně metody či vlastnosti stejného objektu, můžeme jeho název „vytknout“ pomocí příkazu with. V takovém případě je poté možné přistupovat k metodám a vlastnostem přímo, bez uvedení názvu objektu.

Příklad:
`x = Math.cos(Math.PI/2)` lze zapsat jako `with(Math) x = cos(PI/2)`.

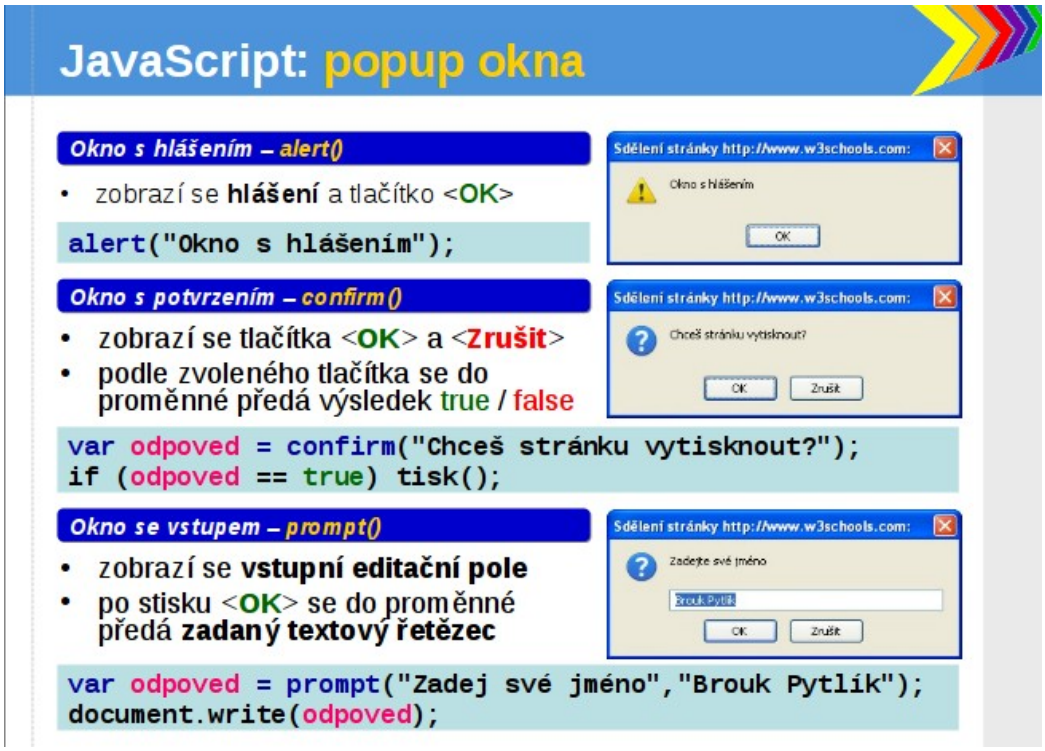


Vyskakovací (popup) okna

Jde o tři speciální metody objektu **`window`**; v případě, že se jedná o aktuální okno prohlížeče, není nutné název objektu uvádět (úplný zápis by mohl vypadat `window.alert("Zpráva")`).
alert(...) - vypíše - ve formě upozornění - hodnotu výrazu v závorce. To lze využít například pro různé kontrolní výpisy.
prompt(text1, text2) - metoda, která zobrazí dialogové vstupní okno, a jako svou hodnotu vrátí řetězec, který do něj uživatel vložil. Nejčastější použití je takové, že funkční hodnota je uložena do nějaké proměnné, a s ní se potom pracuje (jako se řetězcem). Význam parametrů funkce prompt:

- text1 - co se vypíše v okně (informace pro uživatele, co má do okénka zadat)
- text2 - předvolená hodnota (když uživatel nezadá nic).

Například: `jmeno=prompt("Jak se jmenuješ?", "Pepík")`.
Při zrušení tlačítkem *Cancel* vrátí funkce prompt hodnotu **null**.
confirm(text) - funguje podobně jako prompt, ale dává uživateli možnost pouze "text" potvrdit nebo zrušit, neboli vrací logickou hodnotu **true** po stisku *OK* a **false** po stisku *Cancel*. Tlačítka (nápisy na nich) bohužel změnit nelze.



Matematické funkce (objekt Math)

JavaScript má řadu matematických funkcí a konstant v tzv. vestavěném objektu **Math**. V praxi to znamená, že před každou z dále uvedených funkcí i konstant je třeba napsat posloupnost **Math.** (zakončenou tečkou), např. **Math.abs**, **Math.PI**.

- Goniometrické funkce (úhly v radiánech): **cos**(x), **sin**(x), **tan**(x), **acos**(x), **asin**(x), **atan**(x), **atan2**(x,y) ... poslední určuje úhel od osy x k bodu [x,y].
- Ostatní matematické funkce:
  abs(x), **exp**(x), **log**(x), **sqrt**(x) - absolutní hodnota, e^x, ln x, druhá odmocnina,
  min(x_1,x_2,...), **max**(x_1,x_2,...) - tyto funkce mají libovolný počet argumentů,
  pow(x,y) - umocní x na y,

random() - vrátí náhodné číslo mezi 0.0 a 1.0.

- Zaokrouhlení (na celé číslo): **ceil**(x) - nahoru, **floor**(x) - dolů, **round**(x) - aritmeticky.
- Konstanty (pozor - píší se velkými písmeny):
  E, **PI**, **LN2**, **LN10**, **LOG10E**, **LOG2E**,
  SQRT2, **SQRT1_2** (odmocnina ze 2, resp. 1/2)

Datové a časové funkce (objekt Date)

Datum a čas je v JavaScriptu reprezentován objektem Date.
Než začneme s datem pracovat, musíme nejprve příkazem **new** vytvořit nový datový objekt a přiřadit ho proměnné.

```
datum = new Date(); // vytvoří proměnnou obsahující aktuální datum
mesic = datum.getMonth() + 1;//leden je 0
denVMesici = datum.getDate();
alert("Dnes je " + denVMesici + ". " + mesic + ".");
```

Pokud se new Date() zavolá bez parametrů, ukazuje aktuální systémový datum a čas klientského počítače.
Datum je v proměnné uloženo jako počet milisekund *od 1.1. 1970*, navenek se ale chová jako řetězec. Pomocí metod objektu Date jsme schopni zjistit některé detailní informace o datu (čase).

metoda	návratová hodnota	poznámka
`getYear()`	rok dvěma číslicemi	Problematická metoda, doporučuje se nahradit metodou getFullYear().
`getFullYear()`	rok	totéž jako getYear, ale vždy 4 číslice
`getMonth()`	měsíc	leden je nula, takže to chce přičítat 1
`getDate()`	číslo dne v měsíci	1 je prvního (překvapivě)
`getDay()`	číslo dne v týdnu	neděle je nula
`getHours()`	počet hodin od půlnoci	0 je půlnoc a hodina po ní
`getMinutes()`	počet minut od začátku hodiny	0 je první minuta, 59 poslední
`getSeconds()`	počet sekund od začátku minuty	
`getMilliseconds()`	počet milisekund od začátku sekundy	
`getTime()`	počet milisekund od 1. 1. 1970	

Datum může být zobrazeno jako řetězec metodami **toString()** a **toGMTString()**. Obě vracejí datum jako textový řetězec. Metoda toString() ve formátu UTC, toGMTString() ve formátu GMT.

```
Sun Oct 16 2011 17:44:35 GMT+0200 (CET)      // toString
Sun, 16 Oct 2011 15:44:35 GMT                // toGMTString
```

Práce s textem (objekt String)

JavaScript má zabudováno několik užitečných metod pro práci s řetězci. S jejich pomocí lze s řetězcem udělat prakticky cokoliv. Jediným problémem zůstává to, že JavaScript indexuje znaky v řetězci od nuly, což může některé uživatele zmást.

- Novou proměnnou (objekt) typu řetězec vytvoříme:

```
var txt1 = "JavaScript";
```
nebo `var = new String;`

- řetězce lze snadno spojovat operátorem +
- **length** - délka řetězce (je to vlastně vlastnost), používá se takto: *řetězec*.length
- **charAt** - znak na n-té pozici řetězce je ***řetězec*.charAt(n)**
  např. `"JavaScript".charAt(1)` vrátí `"a"`
- **substring** - část původního řetězce od znaku s indexem ***a*** do znaku s indexem ***b*** včetně (resp. do konce): ***řetězec*.substring(a,b+1)**, resp. ***řetězec*.substring(a)**
  např. `"JavaScript".substring(1,4)` vrátí `"ava"`
  podobné (novější) funkce:
  - **slice** je definována shodně, navíc jsou povoleny záporné argumenty - ty značí pozici od konce (-1 je poslední, -2 předposlední …)
  - **substr** je definována shodně jako **slice** s tím rozdílem, že druhý argument určuje počet znaků
- **indexOf** - pozice podřetězce **x** v řetězci; v případě, že funkce má dva argumenty, prohledává se řetězec až od **n**-tého znaku včetně: ***řetězec*.indexOf(x)**, resp. ***řetězec*.indexOf(x,n)**
  např. `"JavaScript".indexOf("a")` vrátí 1,
  `"JavaScript".indexOf("a",1)` vrátí 1,
  `"JavaScript".indexOf("a",2)` vrátí 3
  když podřetězec není v řetězci obsažen, vrátí metoda hodnotu -1
- **lastIndexOf** – obdobná funkce jako předchozí, ale řetězec se prohledává odzadu, případně se vše od pozice **n** (včetně) až do konce ignoruje:
  např. `"JavaScript".lastIndexOf("a")` vrátí 3,
  `"JavaScript".lastIndexOf("a",3)` vrátí 3,
  `"JavaScript".lastIndexOf("a",2)` vrátí 1
- **toLowerCase**, **toUpperCase** - převede řetězec na malá, resp velká písmena, formát je např. ***řetězec*.toUpperCase()**.

Práce s polem (objekt Array)

- Objekt **Array** (pole) je proměnná, která může obsahovat více hodnot stejného typu (např. celá čísla, desetinná čísla, objekty typu String, ba i další objekty typu Array).
- Každý prvek pole je označen číselným indexem; v JavaScriptu jsou pole indexována od nuly.
- Ke každé hodnotě uložené v poli tedy můžeme přistupovat pomocí názvu proměnné a příslušného číselného indexu (zapisuje se do hranatých závorek).
- Délkou pole se rozumí počet uložených hodnot. Pole může být i prázdné (např. po své inicializaci) .

Syntaxe deklarace proměnné typu pole pomocí konstruktoru:

```
var pole = new Array(délka);
```

Příklad:

```
var den = new Array(7);
```

Prvním prvkem pole je `den[0]`, posledním `den[6]`.

Jednotlivé dny můžeme do pole vložit:

```
den[0] = "pondělí";
den[1] = "úterý";
```

Hodnoty můžeme do pole vložit hned při jeho vytvoření:

```
var den = new Array("pondělí","úterý","středa");
// v tomto případě je pondělí uloženo pod indexem 0
dnyVTydnu= {1:["pondělí"], 2:["úterý"], 3:["středa"]}
```

```
// v tomto případě je pondělí uloženo pod indexem 1
```
Přidání nového prvku na konec pole:
```
den[] = "Čtvrtek";
```
Každé pole má vlastnost .length, která určuje počet hodnot v poli – tj. délku pole. Ta se využívá i pro procházení pole cyklem:
```
for(i=0;i<den.length;i++){
    document.write(den[i]);
}
```
S polem je v JavaScriptu spojena ještě celá řada vlastností a metod. Jsou podrobněji dokumentovány v referencích jazyka.

DOM – objektový model dokumentu