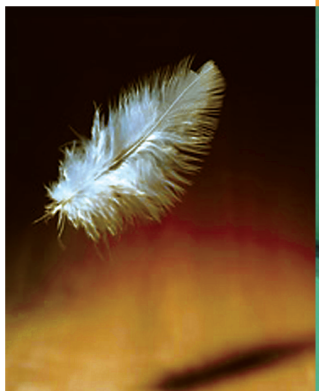
EDICE

INTERNET

RYCHLE A JISTĚ

# Programování WWW stránek

## pro úplné začátečníky



Petr Broža

- základy HTML
- kaskádové styly
- JavaScript
- objektové programování
- dynamické HTML

computer press

VŠECHNY CESTY K INFORMACÍM

Petr Broža

# Programování WWW stránek pro úplné začátečníky

Computer Press
Praha
2000

*Tuto knihu věnuji své přítelkyni Daně,*
*bez jejíž podpory a pochopení by pravděpodobně ani nevznikla.*

# Programování www stránek

## pro úplné začátečníky

Petr Broža

- formuláře
- kaskádové styly
- JavaScript
- objektové programování
- dynamické HTML

computer press
VŠECHNY CESTY K INFORMACÍM

# Programování WWW stránek pro úplné začátečníky

**Petr Broža**


Vydavatelství a nakladatelství Computer Press®,
Hornocholupická 22, 143 00 Praha 4, http://www.cpress.cz

ISBN 80-7226-278-5

Prodejní kód: K0316

**Autor přílohy o Hyperlinku:** Petr Král
**Odborná korektura:** Vlastimil Sapák
**Jazyková korektura:** Barbora Antonová, Josef Novák
**Vnitřní úprava:** Petr Broža
**Sazba:** Petr Broža
**Rejstřík:** Petr Broža
**Obálka:** Jaroslav Novák
**Komentář na zadní straně obálky:** Vlastimil Sapák

**Technická spolupráce:** Jiří Matoušek, Petr Klíma, Mirek Zachrdle
**Odpovědný redaktor:** Ivana Auingerová
**Vedoucí technické redakce:** Martin Hanslian
**Vedoucí knižní redakce:** Ivo Magera
**Vedoucí produkce:** Kateřina Vobecká



**Veškeré dotazy týkající se distribuce směřujte na:**

**Computer Press Brno,** náměstí 28. dubna 48, 635 00 Brno-Bystrc, tel. (05) 46 12 21 11, e-mail: distribuce@cpress.cz

**Computer Press Bratislava,** Hattalova 12/A, 831 03 Bratislava, Slovenská republika,
tel.: +421 (7) 44 45 20 48, e-mail: distribucia@cpress.sk

**Nejnovější informace o našich publikacích naleznete na adrese: http://www.cpress.cz/knihy/bulletin.html. Máte-li zájem o pravidelné zasílání bulletinu do Vaší e-mailové schránky, zašlete nám jakoukoli i prázdnou zprávu na adresu bulletin@cpress.cz.**

# OBSAH

## Část první: Úvod do Internetu

# Část druhá: První kroky do života stránky



# Část třetí: Úvod do JavaScriptu

# Část první: Úvod do Internetu

**Internet: od statiky k dynamice**

**Internet Explorer 5.0**

**Základy HTML**

# 1. Internet: od statiky k dynamice

Tato kniha je volným pokračováním bestselleru *Tvorba WWW stránek pro úplné začátečníky*. Předpokládá tedy, že máte základní znalosti vytváření webových stránek – práce s textem, obrázky, odkazy, tabulek, používání rámů. Kde předchozí díl skončil, tam tato kniha plynule navazuje. Naučí vás základy programování webů a vytváření dynamických stránek – stránek, na které se budou jejich návštěvníci pravidelně a rádi vracet.

Konkrétně se seznámíte s vytvářením a užitím formulářů, které zajišťují základní a prakticky jediný feedback (odezvu) mezi čtenáři vašich stránek – toho lze využít buďto u různých anket nebo třeba u vyplňování objednávek v elektronickém obchodu. Naučíte se používat webové styly, CSS, které výrazně pomohou výslednému vzhledu stránku k lepšímu; podobně se používají i styly třeba ve Wordu. A konečně se naučíte základy programovacího jazyka JavaScript, který v kombinaci se styly a dalšími prvky vytváří opravdové dynamické HTML, pojem, který dnes hýbe webovým světem. Všude tam, kde to bude možné a zajímavé budu také uvádět další zdroje rozšíření vašich znalostí z oblasti, ať už to budou odkazy na webové stránky nebo třeba na knížky z vydavatelství Computer Press. Nejdříve ale malý úvod do tématu.

## 1.1 Jak se na webu zrodila interaktivita

Když na přelomu osmdesátých a devadesátých let několik vědeckých odborníků spustilo vlastní počítačovou síť, kterou byli propojeni mezi sebou a na níž publikovali výsledky svého zkoumání, vědecké zdroje a další informace, nevěděli, že dali základ největší počítačové síti na světě – Internetu.

Tehdejší Internet byl velice chudý. Byl totiž vytvořen pro vlastní potřebu vědeckých odborníků, laická veřejnost k těmto stránkám neměla přístup. Nebylo účelem jej používat pro zábavu a komerci, ale pouze a jen pro odborníky. Webové stránky byly jednoduché, striktní, informace co nejvíce „stlačeny“ – přenos dat po tehdejší síti nebyl zdaleka tak rychlý, jak je dnes. Princip tehdejšího Internetu byl tehdy stejný jako dnes: z drahých a výkonných serverů byla data přenášena na klientské počítače a zde zobrazena.

Protože byl kladen důraz na jednoduchost a rychlost komunikace mezi serverem a klientem, webové dokumenty nemohly obsahovat obrázky – vše bylo tvořeno pouze textem a odkazy. To znamená, že z jedné stránky vedl odkaz na jinou stránku, která buďto nějaký pojem na původní stránce vysvětlovala nebo byla s tímto původním dokumentem nějak informačně provázána. Uživatel se tedy mohl „proklikat“ přes množství stránek až k informaci, kterou hledal. Tato provázanost pomocí odkazů se nazývá hypertext (to proto, že tehdejší stránky byly textové).

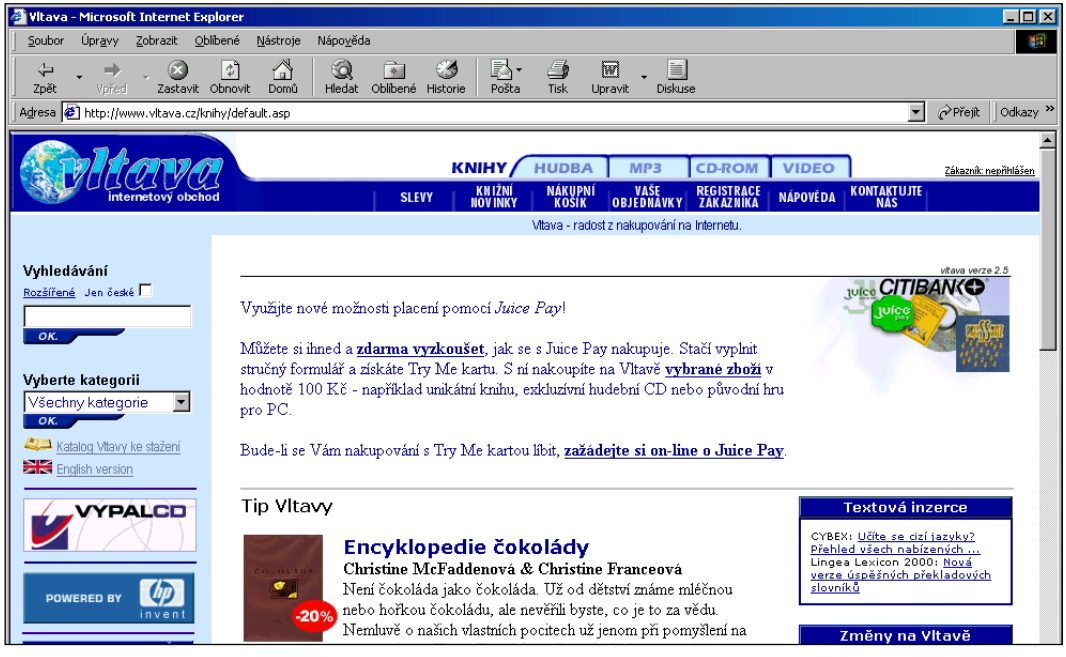
Oproti dnešku tu však byly i další viditelné rozdíly. Stránky postrádaly jakoukoliv interaktivitu, spolupráci s uživatelem. Uživatel tedy nemohl ovlivnit děj na stránce, byl jen pasivním uživatelem a vstřebavatelem informací. Nemohl např. napsat názor na danou problematiku a umístit ji na stránku (dnes známá diskusní fóra). Nebylo také možné přímo na webové stránce provádět např. výpočty, nebylo možné objednávat zboží, nešlo hlasovat v různých anketách; všechno to jsou věci, bez nichž si dnešní web těžko dokážeme představit. Tehdejší

web prostě nebyl živý a dynamický, jaký je teď. Veškeré zásahy šlo provádět pouze ze strany serveru, něco jako dynamicky generované stránky (ASP) prostě neexistovalo. Web tedy nešlo označit ani s notnou dávkou představivosti jako „živý“.

Původní webové stránky byly vytvářeny pouze v jazyku, který popisoval, co se na stránce zobrazí – *HTML*, *Hypertext Markup Language*. Tento jazyk je složen ze značek, které říkají klientskému počítači, jak má data, která mu server pošle, zobrazit, kde bude jaký text a kde odkazy na další stránky. Dnešní výpis zdrojového kódu moderní stránky (např. objednávky na zboží v elektronickém obchodu) je sice na HTML postaven, prakticky se však původní verzi vůbec nepodobá.

V průběhu času byl totiž jazyk HTML rozšířen tak, aby byla prohlížečům umožněna i další činnost než jen tupé zobrazování dat a putování po odkazech. Na straně serveru se dal spustit jazyk, který komunikoval přes prohlížeč s klientem a na základě jeho požadavků zpracoval výsledek, který potom klient zobrazil. Toto řešení umožňovalo zpracovávat data zadaná uživatelem na straně serveru – tato metoda se nazývá *CGI*, *Common Gateway Interface*. Skripty CGI mohou být napsány v jakémkoliv jazyce, který server zná, běžně to bývá Perl. Skripty CGI tak umožnily vygenerovat webovou stránku na základě požadavku klienta za běhu, nešlo už jen o statické stránky předem vytvořené a na serveru uložené. Klepnutí na hypertextový odkaz tak klidně mohlo vést k tomu, že byla vytvořena nová webová stránka, a ta potom na klientském počítači zobrazena. Této interakce se dalo využít u zpracování anket, při nákupu ve virtuálním obchodu apod.

Používání skriptů CGI má však jednu velkou nevýhodu, a tou je běh na straně serveru. Zpracováním dat je server hodně vytížen, zvláště sejde-li se na stránce více uživatelů, kteří




Internetový obchod Vltava kombinuje skripty a ASP (Active Server Pages)

služeb tohoto skriptu využívají. Odezvy pak mohou být tak dlouhé, že uživatelé stránky opustí. Proto je současná tendence co nevíce práce přesouvat na klientský počítač, který nemusí být tak výkonný jako server, protože jeho strojový čas má k dispozici výhradně jeden uživatel. A tady se rodí prostor pro skripty běžící na klientu, mezi nejznámější patří *JavaSript* a *VBScript*.

Abych mohl lépe vysvětlit, jakým způsobem fungují skripty zpracovávané na straně serveru a na straně klienta, uvedu příklad. Dejme tomu, že existuje elektronický obchod, kde si lze koupit rohlíky. Jakým způsobem nákup probíhá u skriptu běžícím na serveru?

1) Vejdu do prodejny a do košíku přidám deset rohlíků.
2) Klepnu myší na tlačítko objednávky, zobrazí se objednávka s deseti rohlíky.
3) Počítač požaduje vyplnit adresu a způsob úhrady nákupu.
4) Vyplníte požadovaná data a odešlete je tlačítkem.
5) Server po obdržení objednávky zkontroluje správnost vámi uvedených dat.
6) V případě chyby vás požádá o doplnění nebo opravu dat v objednávce a skočí zpět na bod 3.
7) Pokud jsou data správně, zašle potvrzení o přijetí objednávky a zboží expeduje.

Všimněte si, že po každém vyplnění dat je server nucen objednávku zkontrolovat a při chybě požádat znovu o vyplnění. Pokud se nějaká chyba opakuje, je server neustálou kontrolou a komunikací s klientským počítačem docela vytížen.

A teď se podívejme, jako to bude vypadat, když přímo na klientském počítači poběží skript, který před odesláním objednávky data zkontroluje a serveru ji odešle pouze správně vyplněnou:

1) Vejdu do prodejny a do košíku přidám deset rohlíků.
2) Klepnu myší na tlačítko objednávky, zobrazí se objednávka s deseti rohlíky.
3) Počítač požaduje vyplnit adresu a způsob úhrady nákupu.
4) Vyplníte požadovaná data a odešlete je tlačítkem.
5) Skript běžící ve vašem prohlížeči zkontroluje správnost vámi uvedených dat.
6) V případě chyby vás požádá o doplnění nebo opravu dat v objednávce a skočí zpět na bod 3.
7) Pokud jsou data správně, zašle objednávku serveru.
8) Ten protože ví, že je objednávka vyplněna dobře, sice provede kontrolu, ale nemusí se zdržovat generováním nové stránky. Pak už je postup stejný jako výše.

Z příkladu je patrné, kolik dokáže klientský počítač ušetřit práce serveru, který se tak může věnovat daleko bohulibějším činnostem. Část práce za něj odvede skript, který je zpracováván prohlížečem v klientském počítači.

## 1.2 Skripty a jejich využití

Ve zkratce tedy víme, co jsou skripty. Protože je tato kniha především o nich a o jejich propojení na objekty prohlížeče, povíme si o nich něco více.

Skript je svým způsobem plnohodnotný programovací jazyk upravený tak, aby jej prohlížeč stíhal překládat a provádět v reálném čase. Jde tedy o jazyk interpretovaný, nikoliv kompilovaný, jako třeba Pascal nebo C++. Zatímco kompilované jazyky jsou před spuštěním přeloženy do strojového kódu, který je následně spuštěn, skript je prováděn přímo ze svého zdrojového kódu. Do zdrojového textu webové stránky tedy není vkládán odkaz na nějaký binární soubor, ale přímo kód skriptu. Na rozdíl od značek HTML neříká prohlížeči, kde a jaký prvek má zobrazit, ale říká, co má prohlížeč udělat, jaká data zpracovat apod.

Oba používané skripty, JavaScript a VBScript, mají svoje analogie ve světě skutečných programovacích jazyků. U JavaScriptu je to Java, u VBScriptu pak Visual Basic. V těchto jazycích se vytvářejí klasické programy spustitelné i mimo web; oba mají svá jedinečná specifika, proto je stručně představím:

*Java* je programovací jazyk velice podobný známému C++. Jeho základní a velice dobrou vlastností je hardwarová a platformová nezávislost. Nezáleží na tom, na jakém počítači program napsaný v Javě spouštíte, poběží totiž všude. To je dáno zvláštním přístupem ke kompilaci. Program napsaný v Pascalu nebo C musíte před použitím zkompilovat do strojového kódu, což omezuje použití tohoto programu pouze na jednu platformu: program pro Windows 98 nepoběží na OS/2 nebo Linuxu a naopak. Program v Javě se však zkompiluje do zvláštního kódu, tzv. *Java Byte Code*, který je následně kompilován za běhu na dané platformě. Jednoduše řečeno, program je interpretován a prováděn zároveň. Takový kód je stejný pro všechny platformy a je tedy přenositelný z jedné platformy na druhou. Musíte však mít v počítači interpret Javy – *Java Virtual Machine* (JVM). Tohle všechno zmiňuji z jednoho důvodu: JVM je totiž implementován do většiny současných prohlížečů, včetně Internet Exploreru a Netscape Navigatoru. Důsledky z toho plynoucí jsou nasnadě – program napsaný v Javě může být součástí webových stránek. Ovšem Java je velice silným nástrojem, který se pro jednoduché oživení stránky nehodí; proto jsou více využívány skripty, pro svoji jednoduchost, rychlost a hlavně optimalizaci pro web. Použít na tyto jednoduché operace Javu je, jako kdybyste šli s dělem na zajíce.

*Visual Basic* je stejně jako Java vývojovým nástrojem, na němž lze postavit jakoukoliv aplikaci. Omezen je však pro použití ve Windows, jeho otcem je totiž společnost Microsoft. VB existuje ve dvou mutacích: *Visual Basic* pro tvorbu plnohodnotných programů, které běží v prostředí Windows, a *Visual Basic for Applications*, který je určen pro programování rozšiřujících možnost již existujících aplikací Microsoftu. Lze v něm např. programovat makra pro programy Microsoft Office (Excel, Word, Outlook…).

Přestože oba zmíněné skripty vycházejí z těchto programovacích jazyků, společného za tu dobu vývoje už mnoho nemají. Více než JavaScript Javě se podobá VBScript Visual Basicu. V žádném případě však není nutné mít jakoukoliv znalost nějakého programovacího jazyka, abyste se mohli naučit skripty používat; je to však pro vás malou výhodou.

# 2. Internet Explorer 5.0

Na obsah webu se díváme skrze program zvaný prohlížeč. Ten je vstupní branou do tohoto virtuálního světa, proto musíte dobře znát jak se ovládá, abyste jej sami později mohli efektivně využít jako tvůrci tohoto obsahu. V této kapitole vás stručně seznámím s programem Internet Explorer 5.0, který je spolu s verzí 4.01 nejpoužívanějším prohlížečem na světě.

Už ve verzi 4.0 byl Internet Explorer velice kvalitním webovým prohlížečem. Integroval v sobě kromě vlastního prohlížení webových stránek také mnoho dalších komponent, které přinesly uživatelům webu snazší a jednodušší život. Program je ve verzi 5.0 stále komplexnější, což jej dělá podstatně větším. Už to není jenom prosté prohlížení Internetu – je to také e-mailový klient (*Outlook Express*), klient FTP, editor WYSIWIG HTML kódu (*Frontpage Express*), přehrávač multimediálních souborů (*Přehrávač záznamů*) a hodně dalších malých programů a plug-inů (zásuvné moduly).

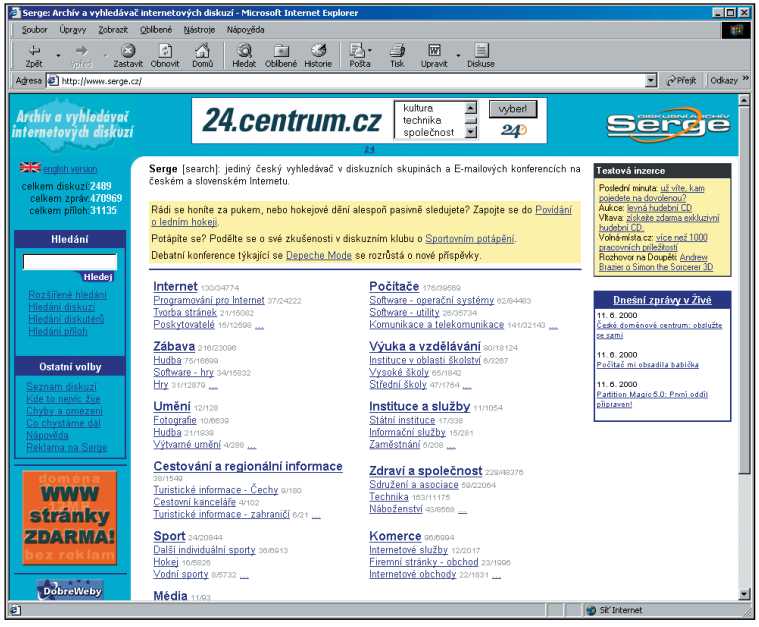
V současné době je Internet Explorer nejlepším a nejrozšířenějším webovým prohlížečem. Jeho konkurenti, kterých kdysi bylo jako máku, se dnes dají spočítat na prstech ruky a navíc kvalit Exploreru nedosahují. Ač by se mohlo zdát, že se od poslední verze nezměnil, není to pravda; výrazné změny proběhly zejména uvnitř. Tvář zůstala skoro stejná. Uživateli by se tak mohlo zdát, že má v ruce jeden a týž produkt. I další profesionální programy (např. *Office 2000*) se na první pohled neliší, a přitom doznaly obrovských změn. Ovládání programů zůstává stejné nejen proto, že si na něj lidé zvykli, ale proto, že je velmi blízké ideálu a tudíž na něm není potřeba téměř nic měnit. I Windows zůstávají téměř stejná už od verze 95; dostala se do stádia, kdy už není potřeba vylepšovat ovládání a vzhled, ale rozšiřovat funkce a možnosti. Takže ačkoliv řada uživatelů (mezi něž jsem patřil i já) nepozná, že místo čtverky má na počítači pětku, Internet Explorer 5.0 (verze 5.5 je už v přípravě a venku je první betaverze) je novým produktem s mnoha novými vylepšeními. Proto se na tuto verzi přesedlat vyplatí.

**Poznámka:** *Internet Explorer 5 je integrován v operačních systémech Windows 98 Second Edition a Windows 2000; pokud máte Windows 95 nebo Windows 98 prvního vydání, musíte si Internet Explorer 5 nainstalovat dodatečně, instalace má necelých 70 MB a lze ji stáhnout z adresy* http://www.microsoft.com.

## 2.1 Co najdeme v okně Internet Exploreru a jak ho nakonfigurovat

Okno aplikace Internet Explorer má dvě základní plochy – jednou je plocha, kde se zobrazuje vlastní obsah webové stránky, druhou *základní nabídka*, *panel nástrojů*, *adresní řádek* s webovou adresou aktuální stránky, *panel rádio* a *panel odkazy*. Explorer umožňuje nezávislou konfiguraci těchto panelů, takže je nemusíte mít zobrazeny všechny.

Pokud klepnete kdekoliv na panelu pravým tlačítkem, objeví se kontextová nabídka, kde si zapnete nebo vypnete zobrazení jednotlivých prvků na panelu. To provedete zatrhnutím nebo naopak odškrtnutím dané položky. Jediné, co nejde vypnout, je *hlavní nabídka*.



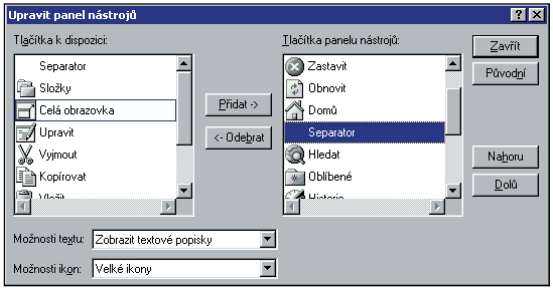

Hlavní okno aplikace Internet Explorer 5.0

Explorer jde dokonce tak daleko, že si můžete naprosto volně nakonfigurovat a poskládat tlačítka na panelu nástrojů. Nepoužíváte tlačítko *Domů*? Odstraňte je! Chybí vám tlačítko pro přepnutí do celoborazovkového režimu? Přidejte si je! Stačí, když zvolíte z kontextové nabídky položku vlastní, čímž se dostanete do konfiguračního okna panelu nástrojů.

Nastavení tlačítek je naprosto jednoduché. V levé části máte okno, kde se nacházejí tlačítka, která nejsou na panelu umístěná, v pravé části je pak okno s tlačítky, která na panelu nástrojů jsou. Přesouváním tlačítek z levého do pravého pruhu, nejlépe tažením, protože díky němu můžete i řídit pořadí tlačítek, podle toho, kam při tažení tlačítko vložíte, můžete ale také tlačítka vybrat a použít příkazy *Přidat* nebo *Odebrat*. Můžete dokonce vytvářet i skupiny těchto tlačítek, které jsou odděleny svislou čárkou. Tento prvek se nazývá *Separátor* a vkládá se na patřičné místo stejně jako tlačítka.

Kromě konfigurace tlačítek jsou v tomto dialogovém okně také dvě roletkové nabídky. První z nich je nabídka s názvem *Možnosti textu*, která nastavuje textové popisky u tlačítek – možnosti *Zobrazit textové popisky*, *Výběr textu vpravo* a *Bez textových popisek*. První zapne textové popisky pod tlačítky implicitně, čímž se musí tlačítka automaticky zvětšit, aby se tam popisky vešly. Třetí možnost pak texty úplně vypíná, což tlačítka naopak zmenší. Prostřední, která je kompromisem mezi oběma uvedenými, se text zobrazuje vpravo od tlačítka. Tlačítko je menší, ale s popiskem delší. Zajímavé je, že se popisek zobrazuje pouze u ně-kterých tlačítek – to není chyba programu, ale jeho vlastnost.

Třetí roletka *Možnosti ikon* je spíše jenom doplňující k předchozí. Přepíná ikony z malých na velké (*Velké ikony*, *Malé ikony*). Velikost ikon se však stejně přepíná automaticky při nastavení předchozí volby, takže tuto roletku nemusíte vůbec používat. Platí totiž, že pokud vypnete popisky, zmenší se ikony, a pokud je zapnete, ikony se zvětší. Tato možnost tak



Výběr tlačítek do panelu nástrojů

ztrácí trochu na svém významu, takže ji vlastně ani není nutné použít (stejně jako většinu pokročilých funkcí Exploreru).

## 2.2 Jak zobrazit stránku adresou, kterou znáte?

Znáte-li adresu webu, na který se chcete podívat, vepíšete jeho adresu do okénka Adresa v horní části okna Exploreru. Pokud se tam již nějaká adresa nachází, označte ji do bloku a smažte ji.

Samotná adresa se skládá z předpony *http://*, která označuje použitý komunikační protokol, a tedy zároveň i službu (v tomto případě služba webu), *www.vltava.cz* už je vlastní adresa serveru, jehož obsah chcete zobrazit. Při vepisování adresy můžete klidně předponu *http://* vynechat, protože ji Explorer před adresu implicitně doplní. Pokud totiž neurčíte, o jakou službu máte zájem (a tedy nedefinujete žádný protokol), pak prohlížeč automaticky předpokládá, že chcete využít služeb webu, a tedy protokolu HTTP. Druhá část adresy, která následuje za specifikací protokolu, se nazývá URL a skládá se z několika částí, které jsou oddělené tečkami. Nyní stiskněte klávesu ENTER pro potvrzení této volby. Začne probíhat připojování k danému serveru a posléze přenos stránky, okno Exploreru se začíná plnit daty a informacemi ze serveru. Během stahování stavový pruh zároveň vypisuje, co Explorer momentálně dělá a co stahuje.

Při otevírání stránky však nemusíte čekat, až se na disk stáhnou veškeré prvky stránky a stránka se tak zobrazí celá. Pokud již na stránce vidíte informace, které hledáte, můžete načítání zastavit nebo klepnout na odkaz, což automaticky přeruší načítání aktuální stránky a zahájí otevírání stránky nové. Stahování stránky zastavíte klepnutím na tlačítko *Stop*, které se nachází na panelu nástrojů třetí zprava (je červené s bílým křížkem). Naopak, pokud chcete stránku načíst znovu nebo pokračovat v načítání zastavené stránky, stiskněte tlačítko *Obnovit*, které se nachází na panelu hned vedle tlačítka *Stop*. Vždy se ale stránka začne načítat celá znovu. Toho se využívá pro aktualizaci stránky. Pokud víte, že se za dobu, co stránku máte zobrazenu v prohlížeči, něco změnilo, klepněte na *Obnovit* a načte se stránka v aktuální podobě.

Pokud jste se v zadání adresy spletli nebo server, který hledáte, neexistuje, popř. pokud server existuje, ale není na něm požadovaná stránka, ohlásí vám to Internet Explorer chybovým hlášením.

Neexistuje-li vámi zadaný server vůbec, v okně Exploreru se objeví hlášení, že server DNS nemohl najít zadanou adresu a že ji tím pádem máte zkontrolovat a popř. zadat správně. Pokud server sice existuje, ale zadaná stránka na něm není, objeví se chyba protokolu HTTP, že požadovaná stránka není k dispozici.

## 2.3 Jak ovládat stránku v prohlížeči

Pokud je všechno v pořádku, zadali jste správné jméno existující stránky, stránka se načte do prohlížeče a můžete s ní začít pracovat. Pokud stránka přesahuje z obrazovky ven, jsou zaktivněny postranní lišty (jedna vpravo, jedna dole), kterými lze obsah stránky posouvat nahoru a dolů nebo vlevo a vpravo uchopením a tažením příslušné lišty. Lze také používat šipky na klávesnici, pokud je však dokument delší, doporučuji používat lištu, protože ta umožňuje měnit rychlost rolování v závislosti na rychlosti pohybu myši. Šipky mají rychlost konstantní a nelze ji měnit.

Pokud jste šťastní majitelé myši Microsoft Intellimouse (nebo jakékoliv jiné s kolečkem místo prostředního tlačítka), můžete posouvat stránkou také rolováním kolečka – to považuji za velice šikovné řešení, sám je hodně využívám. Kolečko Intellimouse má ještě další využití, ale o tom se zmíním až v příslušené části této kapitoly.

Pro pohyb na stránce můžete využít také klasické klávesy HOME, END, PGUP a PGDOWN stejně jako v textových editorech. Klávesa HOME přesune dokument na začátek, END na konec, kláves PGUP posune dokumentem o stránku nahoru, PGDOWN o stránku dolů.

Na ovládání stránky v prohlížeči není tedy složitého, ovládání velice rychle přejde každému do krve.

## 2.4 Jak se vrátit na již navštívenou stránku?

Základní pohyb po stránkách se provádí pomocí odkazů. Klepnete myší na odkaz a v okně Exploreru se zobrazí stránka, na niž odkaz vedl. Pokud třeba zjistíte, že tato stránka není to pravé, co jste hledali, můžete se na původní stránku vrátit klepnutím na tlačítko *Zpět*. To vás vrátí přesně o jednu načtenou stránku. Pokud se chcete vrátit o více než jednu stránku, stačí, když místo na tlačítko *Zpět* klepnete na malou šipku, která se nachází vpravo od tohoto tlačítka. Vyroluje se seznam devíti posledně navštívených stránek, které můžete vyvolat pouhým klepnutím na požadovaný název. Vzhledem k tomu, že si Internet Explorer již navštívené stránky ukládá na váš pevný disk, nemusíte vždy čekat, až se tato stránka stáhne celá znovu z Internetu. Někdy se však stránka za dobu, co jste byli pryč, změní, a protože to Internet Explorer nepozná, zobrazí opět tu, co předtím uložil na disk. To se dá vyřešit buďto stiskem klávesy F5 nebo klepnutím na tlačítko *Obnovit* na hlavním panelu. Obojí začne stahovat aktuální stránku znovu z Internetu, přičemž na disk se uloží tato aktualizované verze.

V okamžiku, kdy se vrátíte na navštívenou stránku, přestane být tlačítko *Vpřed* zašedlé a „rozsvítí se“ (tedy zčerná). Nyní na něj můžete klepnout myší. Co se stane? Dostanete se opět na stránku, kterou jste před chvilkou opustili. Pomocí tlačítek *Zpět* a *Vpřed* se můžete pohybovat tam a zpět po celé historii vašich návštěv; z každého mezikroku samozřejmě můžete „odbočit“ jinam, a tím změnit celou strukturu historie. Jak to přesně funguje, zjistíte experimentováním. Stejně jako u vracení i postupu vpřed lze klepnutím na šipku vedle tlačít-

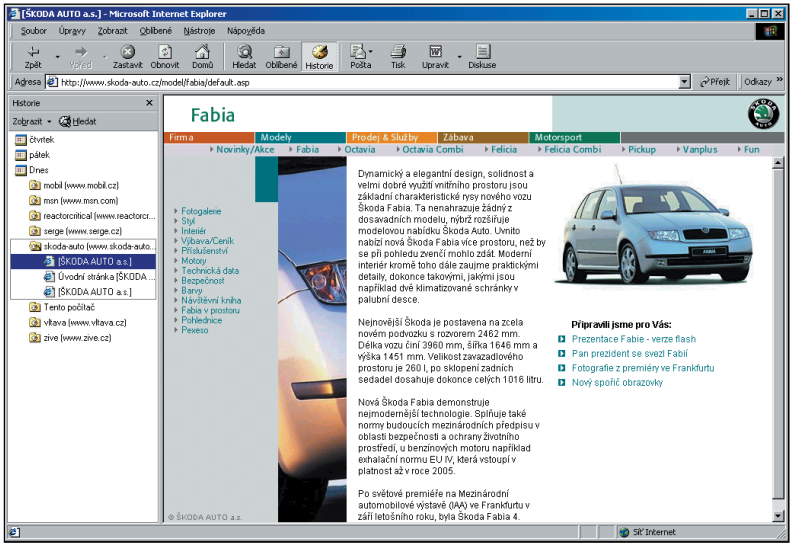

Zpět se můžete vrátit až o devět stránek

ka *Vpřed* vyvolat seznam devíti stránek, z nichž jste se vrátili. Obě tlačítka tak velice usnadňují surfování v případě, že na neznámem webu hledáte hodně zakuklené informace.

Internet Explorer ukládá celou historii všech stránek, které uživatel navštívil, po celou dobu používání programu. Tato historie se ukládá na disk a po startu Exploreru se otevírá, takže se neztrácí ani po uzavření programu či vypnutí počítače. Tato historie je však něco trošku jiného, než ta, která funguje v případě tlačítek *Zpět* a *Vpřed*. Ta se při vypnutí Explorer ztrácí a vytváří se vždy znovu, když Explorer spustíte.

Vlastní historii všech navštívených stránek zapnete tlačítkem *Historie* na panelu nástrojů. V levé části okna Exploreru se otevře seznam navštívených stránek. Tento seznam lze také otevřít z nabídky *Zobrazit*, položka *Panel aplikace Explorer* a příkaz *Historie*, nebo klávesovou zkratkou CTRL-H.

Základní zobrazení tohoto panelu je podle data. Na obrázku vidíte rozbalenou položku *Dnes* – to jsou adresy, které jsem navštívil za dnešní den. Nad dneškem vidíte položky *Středa*, *Úterý*, *Pondělí*, *Minulý týden* apod., kde zase najdete adresy, které jsem navštívil v dané dny. Pokud klepnete na jednu z těchto položek, rozbalí se patřičný seznam stránek. Pokud klep-




Panel historie navštívených stránek za dnešní den

nete na stránku, stránka se zobrazí v okně Exploreru. Výhodou je, že je uložena na disku počítače a není ji tedy nutné stahovat zdlouhavě z Internetu. V okně Exploreru se objeví tedy téměř ihned.

**POZNÁMKA:** *Pokud ukážete myší na jakoukoliv položku v seznamu, objeví se pod ní bublinkové okénko se specifikací konkrétní adresy dané stránky.*

Způsob zobrazení historie si můžete sami nastavit. To provedete klepnutím na tlačítko Zobrazit vlevo nahoře nad sloupcem *Historie*. Rozvine se nabídka s těmito čtyřmi funkcemi:

- *Podle data* – Tato možnost již byla popsána.
- *Podle serveru WWW* – Seznam se vytvoří podle abecedy adres webů.
- *Podle nejnavštěvovanějších serverů* – V okně *Historie* zobrazí dvacet serverů, které navštěvujete nejčastěji.
- *Navštívené dnes podle pořadí* – Zobrazí seznam stránek navštívených v daném dni podle časové posloupnosti, nejstarší je dole, nejmladší nahoře.

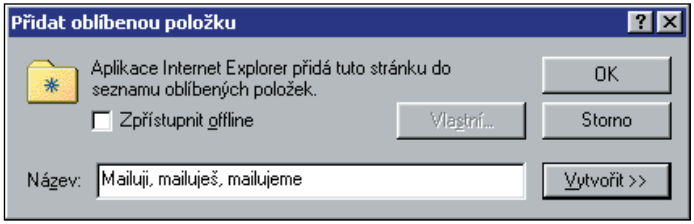
Historie samotná může obsahovat obrovské množství navštívených stránek. Jak dlouho se však bude uchovávat, záleží na tom, jak často historii potřebujete a jak hluboko zpět se chcete pohybovat. Abyste se mohli v historii lépe orientovat, umožňuje ji Explorer prohledávat. Stiskněte tlačítko *Hledat*, které se nachází vedle tlačítka *Zobrazit*. Místo panelu *Historie* se ukáže malé políčko, do něhož napíšete hledané slovo a stisknete tlačítko *Hledat*. Jakmile prohlížeč začne s hledáním, zaktivní se tlačítko *Zastavit*, které hledání přeruší. Seznam nalezených stránek se zobrazí tam, kde předtím byl seznam navštívených stránek historie.

## 2.5 Jak se podívat na svoji oblíbenou stránku?

Pokud máte několik oblíbených stránek, které často čtete, např. denní zpravodajství nebo ekonomické aktuality, můžete, abyste si nemuseli pamatovat všechny jejich adresy a dlouze je vypisovat do adresního řádku, využít tzv. *Oblíbených položek*. Jde o jakýsi seznam adres, které jednoduše vyvoláte přímo z nabídky Exploreru a klepnete na tu vybranou – je to pohodlné a hlavně rychlé. Tento seznam lze třídit do kategorií a ty do dalších podkategorií, a to s prakticky neomezeným vnořováním a neomezeným počtem položek. U katalogu oblíbených položek je relativně pracný jen první úkon, a tím je jeho vytvoření; používání je pak maximálně jednoduché.

Jakmile se načte stránka, kterou chcete do oblíbených položek přidat, klepněte v nabídce *Oblíbené* na položku *Přidat k oblíbeným…* Objeví se dialogové okno *Přidat oblíbenou položku*, v němž se do políčka automaticky vyplní políčko *Název*. Tento název bude stejný, jako má aktuální stránka v záhlaví okna. Můžete jej samozřejmě jakkoliv upravit tak, abyste přesně věděli, jaká adresa se pod danou oblíbenou položkou skrývá. Nyní klepněte na tlačítko *OK*.

Tato stránka se nyní objeví v seznamu *Oblíbených položek*, který vyvoláte klepnutím na nabídku *Oblíbené*, stejně jako když jste přidávali adresu do seznamu. Pod volbami *Přidat oblíbenou položku* a *Uspořádat oblíbené položky* se nachází dosavadní stránky, které jste do seznamu přidali buď vy, nebo byly přidány výrobcem programu.

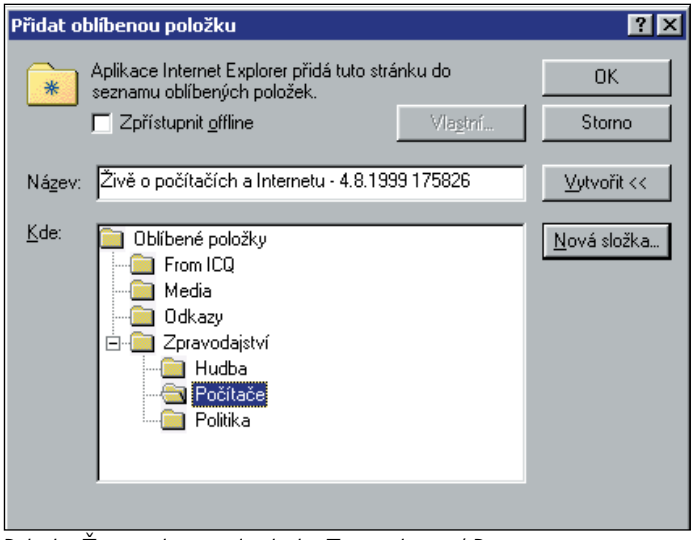



Přidání oblíbené položky do seznamu

Když v této nabídce klepnete na jakýkoliv zde se nacházející příkaz, bude patřičná stránka automaticky zobrazena v okně Exploreru (pokud existuje).

Přidávání položek do seznamu je tedy velice jednoduché. Explorer však umožňuje tyto položky různě kategorizovat a uspořádávat. Abyste mohli přímo při vytváření oblíbené položky umístit stránku do příslušné složky (např. odkaz na zpravodajství Živě přidat do složky *Zpravodajství*), musíte v dialogu na obrázku 26 klepnout na tlačítko *Vytvořit* >>, které rozbalí další okénko. V něm vidíte aktuální strukturu oblíbených položek.

Máte-li zde již vytvořené další složky pro oblíbené položky, můžete klepnout na požadovanou složku (popřípadě na složku v ní zanořenou, používáte-li několik úrovní vnoření) a pak stisknout tlačítko *OK*. Potřebujete-li k tomuto účelu vytvořit novou složku (kdekoli v hierarchii), vyberte složku bezprostředně nadřazenou a zadejte příkaz *Nová složka*, pak jméno nové složky a stiskněte *OK*.



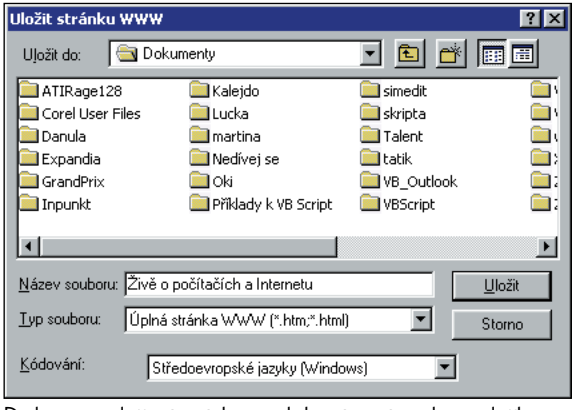

Položku Živě přidáváme do složky Zpravodajství / Počítače

# 2.6 Jak uložit aktuální stránku na disk?

Kromě zobrazení zdrojového kódu stránky umožňuje Internet Explorer také tuto stránku uložit na disk včetně všech objektů a prvků, které jsou ve stránce obsaženy. Z nabídky *Soubor* vyberte příkaz *Uložit jako*, objeví se standardní dialogový panel Windows.

Kromě standardních voleb jako název apod. je důležitá volba *Typ souboru*, který můžete zvolit z následujících možností:

- *Úplná stránka WWW* – Na disk se uloží jak dokument HTML, tak i veškeré prvky, které jsou ve stránce obsaženy (obrázky, banery). Stránka bude upravena pro lokální použití, tzn. při otevření se nebudou tahat obrázky a další prvky z webu.
- *Archiv WWW pro elektronickou poštu* – Uloží se pouze jeden soubor s příponou MHTa tedy jako formát, který se dá poslat elektronickou poštou (Outlook, Outlook Express). Příjemce této pošty dopis otevře jako internetovou stránku i s obrázky a dalšími prvky.
- *Stránka WWW, pouze HTML* – Na disk se uloží pouze dokument HTML, bez prvků na obrázku. Je to podobné, jak kdybyste si stránku nechali zobrazit v Poznámkovém bloku a následně uložili na disk.
- *Pouze text* – Vytvoří soubor, v němž budou uloženy pouze viditelné texty stránky, bez jakéhokoliv formátování.




Dialog pro uložení stránky na disk má navíc jednu položku

Základy práce s Internet Explorerem tedy máme za sebou. Pokud chcete jít více do hloubky, doporučuji knihu *Používáme Internet s programem Internet Explorer 5.0* od Jiřího Hlavenky. Můžeme se tak podívat na základy tvorby stránky; kapitola je shrnutím důležitých kapitol knihy *Tvorba WWW stránek pro úplné začátečníky* s tím, že jsou omezeny příklady, chybí části o seznamech, rámech a tabulkách. Orientuje se totiž spíše na základní znalosti, bez kterých se při studiu dalších částí neobejdete; pokud je však znáte, klidně přeskočte na druhou část.

# 3. Základy jazyka HTML

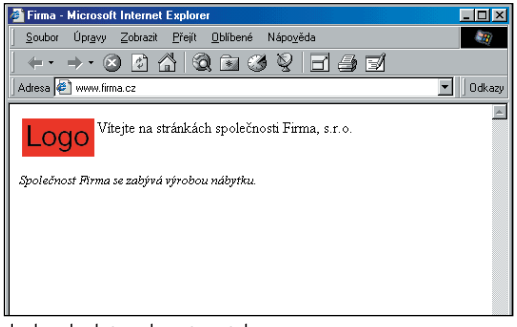
V této kapitole v rychlosti proberu základní značky pro tvorbu stránky. Velice podrobně s mnoha příklady byly probrány v předchozím díle této knihy, *Tvorba WWW stránek pro úplné začátečníky*. Proto berte tuto kapitolu spíše jako rekapitulaci, oživění vašich znalostí, pokud nějaké máte. Pokud ne, může tato kapitola posloužit jako malý rychlokurs.

## 3.1. Co je HTML

HTML (Hypertext Markup Language) je v kód, který slouží k zobrazování dat předem zadaným způsobem; je to vlastně podrobný návod, jak zobrazit přijatá data na obrazovce – kód HTML je tedy přesným a stručným vyjádřením toho, co chceme zobrazit v programu obecně zvaném prohlížeč, jedná se o jazyk, který slouží k popisu webové stránky:

```
<IMG SRC="logo.gif" ALIGN="left">
<FONT SIZE="3" COLOR="black">
  Vítejte na stránkách společnosti Firma, s.r.o.
</FONT>
<BR><BR>
<FONT SIZE="2">
  <I>Společnost Firma se zabývá výrobou nábytku.</I>
</FONT>
```

Ač se vám bude vyjádření téhož v kódu HTML zdát podstatně nesrozumitelnější, věřte, že za chvíli budete takový zdrojový text číst téměř jako knihu a poznáte z něj, jak bude taková stránka v prohlížeči vypadat:




Jednoduchá webová stránka

V průběhu let vývoje Internetu se samozřejmě vyvíjel i jazyk HTML. Nejedná se v žádném případě o jednou navržený kód – HTML se dynamicky mění v souladu s technologickým vývojem a požadavky producentů softwaru a jeho uživatelů. Na jeho vývoji se však nepodílí jen jedna skupina lidí, svou troškou do mlýna přispívají prakticky všichni, zejména dvě vedoucí

firmy na poli prohlížečů – Microsoft a Netscape, které svými prohlížeči ovládají téměř sto procent trhu. Každá tedy do svých programů implementuje vlastní značky, takže konečná podoba stránky vypadá v každém rozdílně. Aby se tomuto předešlo, vznikla W3C (WWW Consorcium), která konečnou podobu HTML schvaluje. Existují tak jednotlivé verze HTML, každá je schválena po určité době používání. Postup je tak opačný, než by každý očekával – místo aby něco vstoupilo v platnost a pak bylo užíváno, nejdříve se užívá a pak teprve oficiálně schválí (nebo také neschválí). V současnosti je poslední schválená verze HTML 4.0, kterou lze v kompletní podobě dokumentovaného výpisu nalézt na internetové adrese

http://www.w3c.org

Abyste si mohli stránky vytvořené v jazyce HTML potřebujete již zmíněný program, obecně nazývaný prohlížeč. Ten interpretuje zdrojovou podobu HTML do grafické podoby, která je čitelná všem uživatelům Internetu (což se o zdrojovém textu říci nedá). Formát HTML je maximálně úsporný, a sice z důvodu zatím nedostatečné propustnosti internetových linek. Pokud vezmu tutéž stránku se stejným nebo podobným formátováním, bude tato stránka v HTML řádově desetinásobně úspornější než tatáž vytvořená programem Microsoft Word. Tato úsporná data jsou také dekódována do grafické podoby podstatně rychleji než z již zmíněného dokumentu Wordu. Nevýhodou HTML jsou zatím omezené formátovací možnosti – na Internetu je totiž vše orientováno na rychlost. Na rozdíl od textového editoru je prohlížeč skutečně pouze prohlížeč – slouží tedy jen k prohlížení dat, ne k jejich vytváření.

## 3.2 Zdrojový text HTML

Každý programovací jazyk má svůj zdrojový text vytvořený programátorem. Stejně tak HTML má svůj zdrojový text, který popisuje grafickou podobu stránky v prohlížeči. Jako každý jazyk i HTML má svoji přesnou syntaxi, kterou je nutno dodržovat, ovšem zároveň je velice přizpůsobivý. I když se dopustíte nějakých chyb, dokáže většinou prohlížeč zobrazit prakticky celou stránku tak, jak vypadat má. Více než tři čtvrtiny stránek na Internetu nemají zcela správnou syntaxi, a přesto jsou zobrazeny správně. Tato flexibilita prohlížečů je dána rychlým vývojem HTML, prohlížeč tedy přímo předpokládá, že se setká s nesprávnou syntaxí, jako je vynechání značky, špatné formátování, chybějící uvozovky u hodnot parametrů apod. Ovšem pokud narazí na zásadní chybu, nedokáže si s ní poradit, a pak je stránka zobrazena špatně. Naštěstí se zdrojový text HTML velice době ladí, protože jediný výsledek je zobrazení, kdežto u opravdového jazyka jde o složité operace, výpočty, které lze odhalit až při důkladném prozkoumání celého programu. U HTML stačí jediný pohled a víte, co není v pořádku. Také je podstatně snadnější naučit se HTML, než jakýkoliv programovací jazyk. V podstatě vám na zvládnutí základů postačí jediný den důkladného studia a vlastního praktického zkoušení a experimentování.

Zdrojový text HTML je vždy pouze textový formát ASCII, který má příponu HTM nebo HTML. V tomto textovém formátu je také interpretován prohlížečem – podoba ASCII je tedy finální a není dále kompilována do žádného binárního souboru, jako EXE nebo COM. Toto je zásadní rozdíl od klasických programovacích jazyků, kdy se program musí před použitím zkompilovat. Všechny ostatní součásti stránky, jako obrázky, zvuky, videoklipy nebo jakéko-

liv jiné binární soubory, nejsou umístěny ve zdrojovém kódu HTML, ale v externích souborech, na něž ze zdrojového textu odkazujete. Takovýto odkaz, konkrétně na obrázek, byl uveden i v příkladě v první kapitole:

```
<IMG SRC="logo.gif" ALIGN="left">
```

kdy jde o obrázek se jménem logo.gif, který je umístěn ve stejném adresáři jako kód HTML. Při zobrazování stránky v prohlížeči je tento obrázek načten do paměti počítače a zobrazen na obrazovce.

Veškeré příkazy, v HTML nazývané *značky* (dříve se používalo názvu tag), jsou uzavřeny ve špičatých závorkách **<>** včetně všech svých parametrů, které značku ovlivňují. Všechno ostatní, co se mimo tyto značky nachází, je text, který prohlížeč zobrazí.

HTML používá párové a nepárové značky, kdy párové mají vliv na určitou část dokumentu, např. na text (kurziva, tučnost, zarovnání apod.) – jeho první část se nachází před danými prvky a druhá za nimi, což přesně vymezuje část, na niž má mít tato párová značka vliv. Nepárová značka naopak má vliv sama na sebe, definuje v HTML nějaký prvek, například obrázek, nebo má vztah k celému dokumentu.

Párová značka tedy obecně vypadá takto

```
<značka>
  kód HTML, na nějž má mít značka vliv
</značka>
```

kdežto nepárová takto

```
<značka>
```

Pokud tyto znalosti adaptujete na příklad v první kapitole:

```
<IMG SRC="logo.gif" ALIGN="left">
<FONT SIZE="3" COLOR="black">
  Vítejte na stránkách společnosti Firma, s.r.o.
</FONT>
<BR><BR>
<FONT SIZE="2">
  <I>Společnost Firma se zabývá výrobou nábytku.</I>
</FONT>
```

pak **<FONT></FONT>** je příklad párové a **IMG** a **BR** nepárové značky. Zjednodušeně lze tedy říci, že celý zdrojový text HTML se skládá z párových a nepárových značek a textu, který je umístěn mimo tyto značky, tedy mimo špičaté závorky.

Z předchozích odstavců je vidět, že HTML má dány přesné principy stavby zobrazované stránky a že je podstatně jednodušší než programovací jazyky, neboť v něm jde pouze o zobrazování, nikoliv výpočty, skoky, podmínky, funkce, podprogramy aj. Lze jej přirovnat

k textovému souboru RTF, který také používá k formátování obsahu dokumentu jednoduché značky, i když jde o formát mnohem komplexnější a nabízející lepší výstup.

Jednotlivých značek má HTML velice mnoho a není účelem této knihy vám je všechny ukázat a popsat. Stále totiž vznikají nové, které nabízejí bohatší zobrazovací schopnosti – HTML se stále vyvíjí a to, co je dnes složité, bude se za rok provádět jednou značkou.

# 3.3 Práce s textem

## 3.3.1 Neformátovaný text

Základním prvkem, kterým se publikují data na Internetu, je text. K jeho zobrazení není potřeba žádné zvláštní značky, za text se tedy považuje cokoliv, co není součástí žádné značky (parametry, hodnoty) uvnitř sekce **<BODY></BODY>**.

Protože prohlížeč nerozlišuje, zda ve zdrojovém kódu stisknete mezi jednotlivými slovy klávesu ENTER a tak vlastně odsunete zbytek textu za klávesou ENTER na další řádek, používá se pro oddělení řádku značka **BR**. Není párová a má jednorázový účinek – odsune následující text na další řádek.

Stejně jako odřádkování klávesou ENTER ve zdrojovém kódu nemá vliv na formátování zobrazovaného textu v prohlížeči, nemá na něj vliv ani vícenásobná mezera. Ta je totiž prohlížečem vždy interpretována jako jediná. Proto se pro pevnou mezeru musí použít speciální značka, ** **, přičemž je nutné dodržet tuto syntaxi včetně středníku, jinak nedosáhnete kýženého efektu.

## 3.3.2 Odstavce a jejich zarovnání

Pro definici odstavců je nepohodlné používat značky **BR**. Proto HTML definuje další dvojici užitečných značek, za jejichž pomoci je celý tento úkon podstatně jednodušší a hlavně zdrojový text se stává čitelnějším.

První z nich je **DIV**, druhou pak **P**. Obě jsou párové a ve svém těle definují odstavec. To znamená, že další text začne na novém řádku. Rozdíl mezi oběma značkami je ten, že **P** navíc vynechá jeden prázdný řádek – to proto, že HTML nepoužívá pro rozlišení odstavce odsazení jeho prvního řádku.

Obě značky mají také jeden důležitý parametry, který definuje zarovnání odstavce v textu: **ALIGN**. Jeho hodnoty jsou následující:

- **left** – pro zarovnání odstavce doleva (je nastaveno jako implicitní);
- **right** – pro zarovnání odstavce doprava;
- **center** – pro zarovnání odstavce na střed, vycentrování.

Zde bych rád uvedl ještě jednu zajímavou párovou značku, která vystředí veškerý obsah, který uzavírá, **CENTER**. Má tedy vliv jak na text, tak na tabulky nebo obrázky – jde tedy o univerzální způsob, jak dostat daný objekt do středu okna prohlížeče.

Mezi značky pro definici odstavce patří také tzv. *citace*. Ta daný text odřádkuje, tj. vynechá před ním prázdný řádek, a provede jeho odsazení o tabulátor. Pro lepší představu příklad:

Dnes máme samozřejmě jiné představy o temperamentech, než ukazuje "šťávová" teorie Hippokratova a později Galéna, protože víme, že chování a prožívání jsou závislá především na nervové soustavě a na sociálně psychologických zákonitostech chování člověka. Uvedeme proto několik definic temperamentů jednotlivých autorů:

> "Temperament je dynamický aspekt osobnosti, charakterizující dynamiku psychické činnosti."
>
> S. L. Rubištejn, 1964

> "Temperament se kryje s pojmem typu vyšší nervové činnosti, který je dán vztahem procesu podráždění a útlumu."
>
> I. P. Pavlov

Citace se definuje značkou **BLOCKQUOTE** nebo **Q**, syntaxe pak vypadá takto:

```
<BLOCKQUOTE> Citovaný text </BLOCKQUOTE>
```

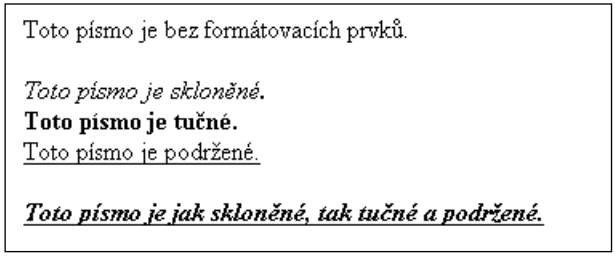
## 3.3.3 Efekty a zvýrazňování písma

Každé písmo v HTML, které použijete, může mít v rámci zdůraznění několik zvýrazňujících prvků. Každý z nich má svoji vlastní značku, která má vliv na veškerý text nacházející se uvnitř těchto značek. Samozřejmostí je kombinace, tzn. písmo může být jak podtržené, tak tučné a skloněné.

Pro tyto základní efekty nabízí tedy HTML tyto značky:

- **B** – tučné písmo;
- **I** – kurziva (nakloněné písmo);
- **U** – podtržené písmo;
- **STRIKE** – přeškrtnutí písma;
- **BIG** – ohraničuje písmo, které bude o jeden bod větší než standardní;
- **SMALL** – ohraničuje písmo, které bude o jeden bod menší než standardní;
- **SUB** – dolní index;
- **SUP** – horní index.

**Poznámka:** *Pokud budete používat kombinace více zvýrazňujících prvků, nezapomeňte na správný postup při zadávání a ukončování platnosti značek, a sice prvně zadaná značka musí být ukončena jako poslední a naopak, posledně zadaná značka ukončena jako první. Bude-li tedy prvně zadána kurziva, následně tučnost a nakonec podtržení, pak nejdříve musíte ukončit účinek podtržení, následně tučnosti a teprve nakonec kurzivy! Jiný postup než tento je sice přípustný, vnáší však do zdrojového textu spíše chaos než tolik potřebný pořádek pro snadnou orientaci v něm.*



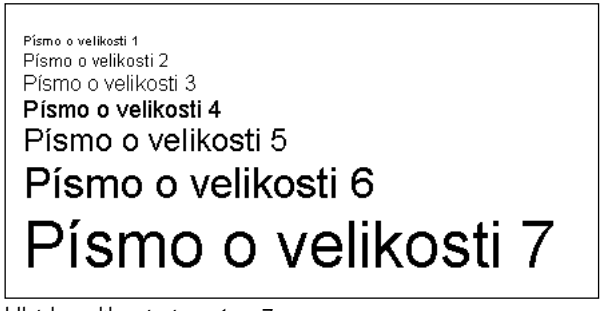

Ukázka základních řezů písma a jejich kombinace

## 3.3.4 Fonty a jejich používání

V textu se navíc nemusíte omezovat pouze na změnu řezu písma, lze zaměnit dokonce i celý font za jiný, a to včetně jeho velikosti. Vynikající vlastností je možnost zadávat velikost písma i relativně, vzhledem k velikosti aktuálně definované. Pokud je například aktuální velikost fontu 2 (standardní nastavení, odpovídá zhruba velikosti 10 ve Wordu), lze ji změnit pouhým přičtením nebo odečtením požadovaného čísla.

Font písma a jeho vlastnosti se definují značkou **FONT**, která má tyto parametry:

- **SIZE="velikost"** – Velikost zvoleného písma od 1 do 7. Pokud zadáte před velikostí znaménko + (plus) nebo - (minus), změníte velikost písma vzhledem k aktuální tak, že se zadaná hodnota přičte (popř. odečte) od této velikosti.
- **FACE="písmo"** – Název písma, který bude použit. Toto písmo musí být v seznamu nainstalovaných písem a musí být zadán jeho přesný celý název; pokud jej zadáte špatně, bude použito písma standardního (Times New Roman).
- **COLOR="barva"** – Definuje barvu písma.




Ukázka velikostí písem 1 – 7

**POZNÁMKA:** *Všechny změny dané touto značkou budou platné pouze do jejich adekvátního ukončení značkou* **</FONT>***, stejně jako platí závislost ukončování u všech ostatních značek. Značka* **FONT** *umí kromě zadání specifické velikosti písma také písmo relativně zvětšovat či zmenšovat. Provádí to vzhledem velikosti písma, které je nastaveno implicitně, v případě Internet Exploreru, to je 2. Ovšem pokud používáte jiný prohlížeč, může se stát, že se impli-*

*citně nastavená velikost může lišit – pak všechny velikosti, které nastavíte značkou* **FONT**, *budou samozřejmě odlišné. Aby se tomuto precedentu zabránilo, definuje HTML značku* **BASEFONT**, *která nastaví základní velikost všech písem. Pokud tedy tuto velikost značka* **FONT** *změní, vrátí se ukončením platnosti této změny značkou* **</FONT>** *velikost zpět na tu, kterou definovala značka* **BASEFONT**.

*Syntaxe zápisu je* **<BASEFONT SIZE="n">**, *kde* **n** *je velikost písma od 1 do 7, tedy podobně jako u značky* **FONT**. *Na rozdíl od ní zde nelze zadat relativní velikost.*

### 3.3.5 Používání nadpisů

V HTML je k dispozici šest úrovní nadpisů, každý se svou specifickou značkou, v níž je přímo definovaná úroveň velikosti nadpisu:

**H1**, **H2**, …, **H6**

přičemž značka **H1** zobrazí největší nadpis, jehož velikost se plynule zmenšuje až k **H6**.

Stejně jako u normálního textu můžete nadpis zarovnat doleva, doprava a na střed, a to stejným parametrem **ALIGN** a stejnou syntaxí. Každý parametr **ALIGN** má vliv pouze na ten nadpis, v jehož značce byl umístěn a jeho platnost jím také končí.

Na vzhled nadpisu, tedy jeho font a barvu, má vliv jednak globální nastavení barev (viz dále), jednak poslední nastavení značky **FONT**. Bylo-li tedy touto značkou nastaveno písmo Arial a barva červená, tak i nadpis bude zobrazen písmem Arial a červenou barvou. Na nadpis samozřejmě nemá vliv nastavení velikosti fontu.

## 3.4 Barvy webové stránky

### 3.4.1 Zadávání barev

HTML definuje pro základní barvy (bílá, černá, modrá, červená atd.) přímé názvy odpovídající anglickému ekvivalentu jména barvy, tedy bílá=white, černá=black, modrá=blue, červená=red apod. Tyto barvy se zadávají jako hodnota parametrů značek. Jak jste si jistě všimli v předchozí kapitole, barva textu se zadává značkou **<FONT COLOR="barva">**.

Protože nelze pokrýt názvy všechny možné barvy a jejich kombinace, lze barvy „míchat“ pomocí hexadecimálního vyjádření. Např. modrá vypadá takto: #0000FF, jednotlivé složky barvy – červená, zelená a modrá (red, green, blue – RGB) se tedy zadávají pomocí osmibitových čísel (8 bitů = 256 kombinací = dvojciferné hexadecimální číslo), tedy v tomto konkrétním případě je červená reprezentovaná dvojicí 00, zelená také dvojicí 00 a nakonec modrá dvojicí FF. Je nasnadě, že změna poměru barev RGB změní i výslednou barvu.

Základní barvy a jejich hexadecimální vyjádření:

| | | |
|---|---|---|
| Aqua | tyrkysová | #00FFFF |
| Black | černá | #000000 |
| Blue | modrá | #0000FF |
| Fuchsia | fialová | #FF00FF |

| | | |
|---|---|---|
| Gray | šedá | #808080 |
| Green | tmavě zelená | #008000 |
| Lime | zelená | #00FF00 |
| Maroon | tmavě červená | #800000 |
| Navy | tmavě modrá | #000080 |
| Olive | tmavě žlutá | #808000 |
| Purple | tmavě fialová | #800080 |
| Red | červená | #FF0000 |
| Silver | stříbrná | #C0C0C0 |
| Teal | tmavě tyrkysová | #008080 |
| White | bílá | #FFFFFF |
| Yellow | žlutá | #FFFF00 |

Abyste nemuseli potřebné barvy míchat přímo v HTML podle oka, je dobré je vytvořit nejdříve v nějakém grafickém editoru, který umožňuje editovat barevnou paletu. Hodnoty tyto barvy tvořící si zapíšete a teprve pak umístíte do HTML. Upozorňuji, že se musí jednat o barevný model RGB, nikoliv CMYK!

## 3.4.2 Výchozí nastavení barev stránky

Každý dokument HTML má svoje výchozí barevné nastavení – barva pozadí, implicitní barva textu, barva odkazy na jinou stránku, barva odkazu již navštívené stránky apod. Všechny tyto implicitní barvy se nastavují na začátku zdrojového textu ve značce **BODY**, která uvozuje tělo vlastního kódu, který bude prohlížeč zobrazovat (viz kapitola 2).

Jednotlivé jeho parametry jsou následující:

- **BGCOLOR** – Barva, která bude použita na pozadí, standardně bílá.
- **TEXT** – Implicitní barva textu, která se použije, nestanoví-li uživatel jinak, standardně černá.
- **LINK** – Barva odkazu, který jste ještě nenavštívili, standardní nastavení je modrá.
- **VLINK** – Barva již navštíveného odkazu, implicitní nastavení fialová.
- **ALINK** – Barva odkazu, na nějž jste právě klepnuli, standardně červená.

Pokud tedy použijete ve značce **BODY** všechny uvedené parametry, vypadá to např. takto:

```
<BODY BGCOLOR="black" TEXT="white" LINK="yellow" VLINK="green"
ALINK="red">
```

Tato značka nastaví následující barvy: barva pozadí černá, barva textu bílá, barva odkazu nenavštívené stránky žlutá, barva odkazu navštívené stránky zelená a konečně barva odkazu, na nějž jste právě klepnuli, červená.

# 3.5 Obrázky na stránkách

Při umísťování používejte co největší kompresi, neumísťujte obrázky zbytečně velké a hlavně se vyvarujte zbytečných a matoucích obrázků, a to nejen kvůli vyšší celkové velikosti stránky – dbejte na grafickou vyváženost, střídmost a úpravu. Internet by totiž měl být přímočarý a přehledný, ne plný všelijakých obrázků, které nemají pro uživatele význam.

## 3.5.1 Umísťujeme obrázky

Značka **IMG** je jedna z mála nepárových značek HTML. Má základní syntaxi

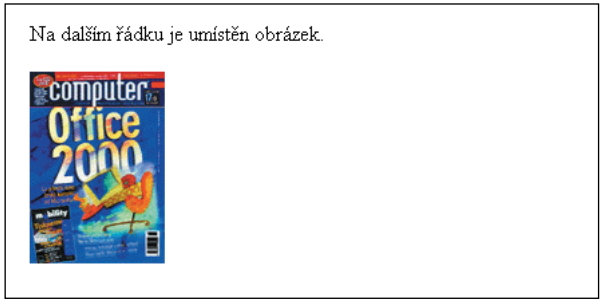
```
<IMG SRC="obrázek">
```

a definuje soubor s obrázkem, který se má na stránku umístit. Jméno souboru je uvedeno v hodnotě parametru **SRC**; pokud není u jména souboru uvedena cesta, jak relativní nebo absolutní, prohlížeč hledá obrázek v aktuálním adresáři, tedy v tom, kde je umístěn dokument, který obrázek zobrazuje. Celkem lze použít čtyři způsoby zadávání cesty:

- **<IMG SRC="obr.gif">** – Obrázek je umístěn v aktuálním adresáři.
- **<IMG SRC=../obr.gif">** – Relativní zadání lokální cesty, obrázek umístěn v nadřazeném adresáři.
- **<IMG SRC="c:/gifs/obr.gif">** – Absolutní zadání lokální cesty, kdy je obrázek na místním pevném disku v určeném adresáři.
- **<IMG SRC="http://www.stranka.cz">** – Cesta k obrázku je zadána s celou URL, kde se obrázek nachází, tzn. lze zobrazit jakýkoliv obrázek na kterémkoliv serveru na celém světě.

Nejčastěji používané grafické soubory jsou typu JPG a GIF, které zároveň nabízejí nejvyšší kompresi a tedy i úsporu místa, občas se používá i formátu PNG. Každý z těchto formátů má svoje výhody a nevýhody, např. GIF může používat průhledné barvy, JPG zase až 16,7 miliónu barev; GIF pouze 256. Dané formáty se tedy používají tam, kde se více hodí. Potřebujete-li celou barevnou paletu, například u fotografií, použijete JPG. Potřebujete-li naopak obrázek s průhledným pozadím, sáhnete po GIFu. Formát GIF se také hodí zejména pro různá tlačítka akcí a jiné grafické prvky.

```
<HTML>
<HEAD>
</HEAD>
<BODY>
  <P>Na dalším řádku je umístěn obrázek.</P>
  <IMG SRC="obrazek.gif">
</BODY>
</HTML>
```




Jednoduché zobrazení obrázku v HTML

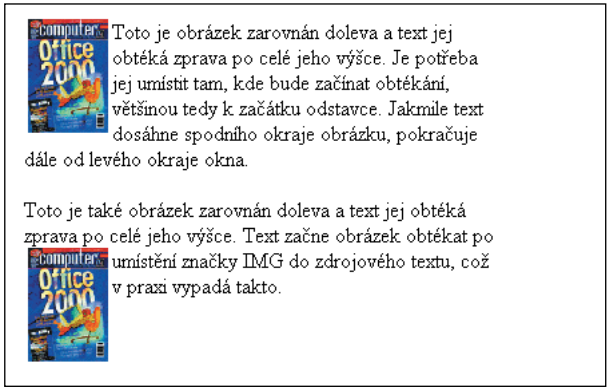
## 3.5.2 Zarovnání obrázku a textu

Obrázky lze také s omezenými možnostmi zarovnávat vzhledem k okolnímu textu, který obrázek obtéká. Jelikož standardní HTML nepodporuje takové možnosti formátování jako standardní textový editor, neočekávejte žádné zázraky. Jedná se pouze o základní zarovnání, které je dosaženo následujícím parametrem **ALIGN** (stejný parametr jako u zarovnání odstavců) značky **IMG**:

- **left** – Obrázek je umístěn zcela vlevo a je obtékán textem z pravé strany. Jakmile je dosaženo spodního okraje obrázku, pokračuje text zleva doprava na celou šíři okna prohlížeče.
- **right** – Obrázek je umístěn zcela vpravo a je obtékán textem z levé strany. Jakmile je dosaženo spodního okraje obrázku, pokračuje text zleva doprava na celou šíři okna prohlížeče.
- **texttop** – Zarovná horní okraj obrázku s nejvyšším písmenem v řádku, u něhož je umístěn.
- **top** – Zarovná horní okraj obrázku s nejvyšším prvkem (většinou písmenem – tehdy je zarovnání stejné jako u předchozí hodnoty **texttop**) v řádku, u něhož je umístěn.
- **middle** – Zarovná linku, na níž je posazen text, na střed daného obrázku.
- **absmiddle** – Zarovná prostředek daného řádku na střed obrázku.
- **bottom** – Zarovná spodní okraj obrázku s linkou, na níž je posazen text.
- **absbottom** – Zarovná spodní okraj obrázku s nejspodnějším písmenem v textu, tedy písmena, které je posazeno pod linkou (např. j,p,y).

Standardně je v prohlížeči navoleno zarovnání **bottom**, tedy spodní okraj obrázku s daným řádkem, dále pak text pokračuje pod obrázkem normálně. Většinou se používá zarovnání **left** nebo **right**, aby text obrázky hezky obtékal. Způsob obtékání vidíte na následujícím příkladu.

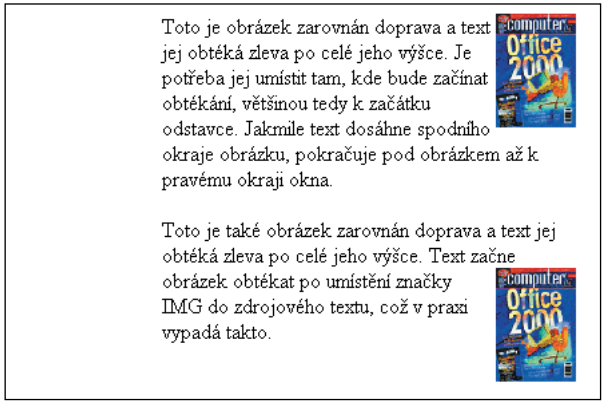
```
<BODY>
  <P><IMG SRC="obrazek.gif" ALIGN="left">
     Toto je obrázek zarovnán doleva a text jej obtéká zprava po
     celé jeho výšce. Je potřeba jej umístit tam, kde bude
     začínat obtékání, většinou tedy k začátku odstavce. Jakmile
     text dosáhne spodního okraje obrázku, pokračuje dále od
     levého okraje okna.</P>
  <P>Toto je také obrázek zarovnán doleva a text jej obtéká
     zprava po celé jeho výšce.
     <IMG SRC="obrazek.gif" ALIGN="left">
     Text začne obrázek obtékat po umístění značky IMG do
     zdrojového textu, což v praxi vypadá takto.</P>
</BODY>
```




Dva způsoby obtékání obrázku zarovnaného vlevo

Podobně pracuje prohlížeč s textem, pokud je obrázek zarovnán doprava, viz příklad.

```
<BODY>
  <P><IMG SRC="obrazek.gif" ALIGN="right">
     Toto je obrázek zarovnán doprava a text jej obtéká zleva po
     celé jeho výšce. Je potřeba jej umístit tam, kde bude
     začínat obtékání, většinou tedy k začátku odstavce. Jakmile
     text dosáhne spodního okraje obrázku, pokračuje pod
     obrázkem až k pravému okraji okna.</P>
  <P>Toto je také obrázek zarovnán doprava a text jej obtéká
     zleva po celé jeho výšce.
     <IMG SRC="obrazek.gif" ALIGN="right">
     Text začne obrázek obtékat po umístění značky IMG do zdro-
     jového textu, což v praxi vypadá takto.</P>
</BODY>
```




Dva způsoby obtékání obrázku zarovnaného vpravo

Pokud nepoužijete žádného zarovnávacího parametru, je obrázek vložen do textu přesně tam, kde jej umístíte ve zdrojovém kódu. Ovšem zarovnání je implicitně nastaveno na **bottom**, text tedy bude zarovnán na spodní stranu obrázku.

```
<BODY>
  Tohle je obyčejný text, do něhož
  <IMG SRC="obrazek.gif">
  vložen obrázek. Text se zarovná ke spodnímu okraji tohoto
  obrázku a dál bude pokračovat pod ním.
</BODY>
```


Implicitní zarovnání textu a obrázku je nastaveno na bottom

Jediný způsob, jak nastavit obtékání obrázku textem podél celé jeho výšky, je tedy nastavit mu zarovnání doleva či doprava, jinak bude text vždy srovnán se spodním okrajem. Jedno z řešení, jak toto obejít, je použití tabulek (viz knihu *Tvorba WWW pro úplné začátečníky*).

## 3.5.3 Jak zadat velikost obrázků

HTML umožňuje zadávat jako další parametry značky **IMG** také velikost obrázku. Pokud ji nezadáte, zobrazí se obrázek normálně, budou dodrženy jeho rozměry. Pokud má tedy obrázek výšku 100 bodů a šířku 50 bodů, pak se tak i zobrazí na obrazovce. Někdy je potřeba zobrazit obrázek větší nebo menší, než je jeho skutečná velikost. Pokud tedy nastavíte rozměry jiné, než je pravá velikost obrázků, prohlížeč jej adekvátně zvětší nebo zmenší do

vámi určené velikosti. Pro nastavení velikosti obrázku jsou definovány tyto parametry značky **IMG**:

```
WIDTH="šířka v obrazových bodech"
HEIGHT="výška v obrazových bodech"
```

Lze také zadat pouze jeden parametr, výšku nebo šířku. Pokud takto učiníte, bude zbývající rozměr obrázku automaticky dopočítán do daného poměru.

## 3.5.4 Popisek obrázku

HTML umožňuje přiřadit každému obrázku jeden jakýkoliv popisek, který je využit v následujících případech:

- zobrazí se nad obrázkem ve formě bublinky, kdykoliv nad něj najedete kurzorem myši,
- ukáže v rámečku vyhrazeném pro načítání během stahování obrázku, které někdy trvá opravdu dlouho (nemusíte pak čekat, až se načte celý, stačí se orientovat podle těchto popisků),
- zobrazí na místě obrázků, pokud vypnete zobrazování obrázků kvůli rychlejšímu zobrazování.

Tento parametr má syntaxi

```
<IMG SRC="obrazek.gif" ALT="Tohle je popisek obrázku">
```

**Doporučení:** *Tyto popisky používejte vždy, protože předem nevíte, jak dlouho se u případného uživatele bude stránka s obrázky načítat nebo zda nemá vypnuto zobrazování grafiky. Popisky na takových stránkách usnadňují orientaci.*

## 3.5.5 Rámeček okolo obrázku

Obrázek také může mít volitelně okolo sebe rámeček různé velikosti – to se hodí zejména tehdy, nemá-li okraj vlastní a tento okraj by přispěl k přehlednosti celého dokumentu (např. je bílý jak pozadí obrázku, tak pozadí stránky, a tudíž není patrné, kde se nachází ohraničení obrázku).

Rámeček se definuje parametrem **BORDER**:

```
<IMG SRC="obrazek.gif" BORDER="n">
```

kde **n** je šířka rámečku v obrazových bodech. Barva rámečku je vždy černá, výjimku tvoří případ, kdy je obrázek zároveň odkazem, tehdy je barva rámečku modrá (či má barvu odkazu definovanou ve značce **BODY**) a má velikost rámečku dva body. Pokud neuvede parametr **BORDER**, nebude rámeček zobrazen.


Obrázek s rámečkem o velikosti 2 body

## 3.5.6 Volné okraje okolo obrázku

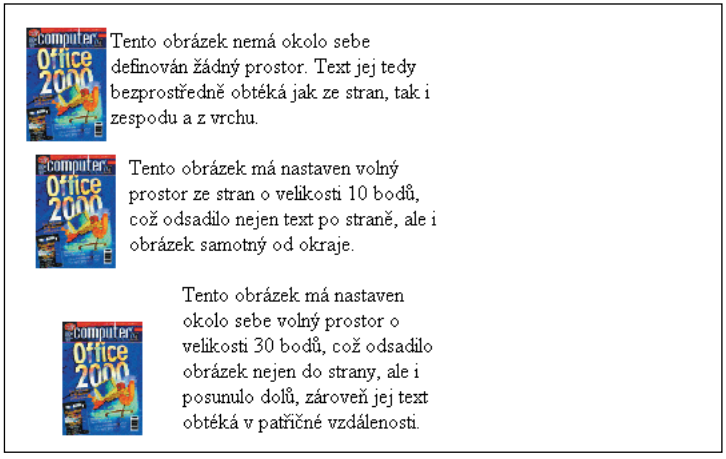
Určitě jste si již všimli, že pokud text obtéká obrázek, je na něj vždy těsně nalepen - mezi ním a obrázkem tedy není žádná mezera, a to nepůsobí moc esteticky. Tomu lze zabránit dvěma parametry, které nastavují volný prostor mezi obrázkem a dalšími prvky, které se okolo nich vyskytují, v našem případě textem:

**HSPACE="n"** Nastaví velikost volného prostoru v bodech na levé a pravé straně.
**VSPACE="n"** Nastaví velikost volného prostoru v bodech nad a pod obrázkem.

Různá nastavení volného prostoru okolo obrázku jsou patrná z příkladu:

```
<BODY>
  <P><IMG SRC="obrazek.gif" ALIGN="left">
     Tento obrázek nemá okolo sebe definován žádný prostor. Text
     jej tedy bezprostředně obtéká jak ze stran, tak i zespodu
     a z vrchu.</P>
  <P><IMG SRC="obrazek.gif" ALIGN="left" HSPACE="10">
     Tento obrázek má nastaven volný prostor ze stran o
     velikosti 10 bodů, což odsadilo nejen text po straně, ale
     i obrázek samotný od okraje.</P>
  <P><IMG SRC="obrazek.gif" ALIGN="left" HSPACE="30" VSPACE="30">
     Tento obrázek má nastaven okolo sebe volný prostor o
     velikosti 30 bodů, což odsadilo obrázek nejen do strany,
     ale i posunulo dolů, zároveň jej text obtéká v patřičné
     vzdálenosti.</P>
</BODY>
```

**Tip:** *Práce s obrázky je mnohem pestřejší, než je ukázáno v této podkapitole. Jejím cílem však není toto všechno obsáhnout; pokud chcete vědět o práci s obrázky a vůbec webovou grafikou více, nalistujte si knihy* Tvorba WWW stránek pro úplné začátečníky *nebo* Vytváříme WWW stránky a spravujeme moderní web site *z našeho nakladatelství.*




Implicitní zarovnání textu a obrázku je nastaveno na bottom

# 3.6 Odkazy

Odkazy zpřehledňují práci s Internetem. Rozdělují jej na množství různých kapitola a podkapitol, takže se uživatel snadno orientuje ve velkém množství informací. Díky tomu můžete na Internetu také velice snadno vyhledávat, třeba jako u knihy: Pokud hledáte konkrétní kapitolu, podíváte se do obsahu a nalistujete si patřičnou stránku, kde kapitola začíná. Něco podobného, a dokonce efektivnějšího, lze díky odkazům vytvořit i v HTML.

## 3.6.1 Základ práce s odkazy

Aby se daný prvek v HTML stal odkazem, je nutné jej uzavřít do párové značky

```
<A HREF="stranka.html"></A>
```

kde **A** je vlastní značka a **HREF** parametr definující stránku, která se má po kliknutí na tento odkaz zobrazit v prohlížeči. Hodnota parametru **HREF** může být i jakýkoliv binární soubor (videoklip, archív ZIP, aj.) nebo URL (adresa Internetu), zatím však bude odkaz směřovat pro jednoduchost pouze na konkrétní stránku, tedy soubor HTML.

Odkazem, tedy to, co bude na stránce vidět jako odkaz (text, obrázek), je tedy prvek uzavřen značkou **A**. Pokud chcete, aby se odkazem stal text „Toto je odkaz na další stránku“, vložte do zdrojového textu řádek

```
<A HREF="stranka.html">Toto je odkaz na další stránku</A>
```

Odkaz (text) bude v prohlížeči podtržen a zobrazen modře (nebo barvou, která je definovaná na začátku sekce BODY). Když na tento text klepnete kurzorem myši, načte prohlížeč do aktuálního okna (nebo toho, které definujete) soubor *stranka.html*, pokud se nachází

v aktuálním adresáři (pokud se nachází jinde, je třeba zadat cestu). Odkaz také může být pouze součástí věty, nejen věta celá. Lze tak vytvořit i několik odkazů v jednom odstavci:

```
V textu může být i více <A HREF="stranka.html">odkazů</A>. Toto
je <A HREF="stranka.html">první</A> odkaz, toto pak <A
HREF="stranka.html">druhý</A> a nakonec i <A
HREF="stranka.html">třetí</A>.
```

To, co prohlížeč zobrazí jako odkaz, může být i obrázek nebo kombinace textu a obrázku. Pokud tedy chcete, aby se odkazem stal obrázek, zkuste napsat do zdrojového kódu

```
<A HREF="stranka.html">
  <IMG SRC="obrazek.gif">
</A>
```

Daný obrázek (*obrazek.gif*) se v prohlížeči zobrazí s modrým rámečkem o velikosti dvou bodů, pokud to nezakážete:

```
<A HREF="stranka.html"><IMG SRC="obrazek.gif" BORDER="0"></A>
```

Samozřejmě orámování může mít libovolnou velikost, kterou nastavíte parametrem **BORDER**.

Z předchozích příkladů tedy vyplynuly dva shrnující fakty:

- Odkaz může být umístěn uvnitř věty, může tvořit samostatnou větu nebo dokonce i celý odstavec. Kromě textu může být součástí odkazu i obrázek.
- Text, který je ohraničen značkou **A** může být jakýmkoliv způsobem, dostupným v HTML, formátován a zvýrazňován. Značka **A** totiž definuje pouze ten fakt, že tento text bude odkazem na nějakou stránku či soubor. Na text definovaný jako odkaz nebude mít platnost pouze nastavení barvy značkou **FONT**, ta je totiž definována na začátku dokumentu ve značce **BODY** parametrem **LINK**.

Jako odkaz nemusí být pouze soubor nebo stránka HTML, ale také adresa elektronické pošty:

```
<A HREF="mailto:broza@cpress.cz">Pošli vzkaz</A>
```

Klepnete-li na tento odkaz, spustí se klient elektronické pošty (který je nastaven jako výchozí) a umožní vám na zadanou adresu zaslat e-mail. Aby byl odkaz odlišen, nemá modrou, ale tmavě červenou barvu.

## 3.6.3 Odkaz na soubor

Kromě elektronické pošty a webové stránky může být odkazem také jakýkoliv jiný soubor, třeba obrázek nebo archiv ZIP. Syntaxe takového odkazu je úplně stejná jako na webovou stránku:

```
<A HREF="archiv.zip">Archív</A>
```

Rozdíl je v tom, že zatímco stránka je načtena a zobrazena, jedná-li se o soubor, zobrazí prohlížeč nabídku, v níž máte možnost buďto daný soubor stáhnout z Internetu a uložit na lokální disk, nebo spustit přímo z internetového serveru; lze také nastavit, že se soubory daného typu budou rovnou spouštět nebo rovnou stahovat.


Internet Explorer se ptá, co má udělat se souborem

Pokud zvolíte uložení, zjistí prohlížeč od serveru informace o délce souboru a začne jej ukládat na váš pevný disk, což trvá úměrně tomu, jak je soubor dlouhý a jak rychlé je spojení se serverem.

Využijete-li druhé možnosti, spuštění z Internetu, je potřeba mít v systému asociovanou aplikaci, tedy aplikaci, která umí s daným typem souboru pracovat. Tento soubor bude následně stažen na lokální disk, pak se spustí patřičná aplikace, do níž bude tento soubor načten. Výjimku tvoří spustitelný soubor (EXE, COM), který se po stažení spustí sám.

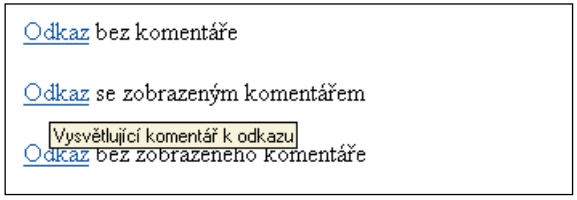
Existují však i soubory, s kterými umí pracovat i samotný prohlížeč. Jedná se např. o obrázky, videoklipy, zvuky, různé animace. Pokud klepnete na odkaz, jehož cílem je právě takovýto soubor, pak se již prohlížeč na nic neptá a přímo jej sám spustí (přehraje, zobrazí). Pokud chcete tento soubor, který je přiřazen k nějaké akci přímo v prohlížeči, stáhnout na lokální disk, musíte na to jít přes pravé tlačítko, které zobrazí lokální nabídku, v níž kromě jiného bude možnost uložení cílového souboru.

## 3.6.4 Doplňkový komentář odkazu

Podobně jako je možné u obrázku zadat popisek, který se vypíše formou bublinky, pokud nad obrázkem zobrazeným v prohlížeči podržíte chvíli kurzor myši, lze totéž provést i u odkazu. Funguje to analogicky s komentáři v textové editoru Microsoft Word – tam pokud chcete vložit komentář k nějakému slovu nebo větě, aniž byste narušili vzhled stránky, lze vložit bublinkový komentář, který se zobrazí jakmile přejedete takto označené slovo myší. Pokud tedy nechcete takovýmito komentáři rušit vzhled stránky, nadefinujte odkazům vysvětlující nebo doplňující popisky:

```
<A HREF="archiv.zip" TITLE="Toto je komentář k souboru
archiv.zip">Archiv</A>
```

přičemž v prohlížeči se odkaz vykreslí stejně jako kdyby tento parametr chyběl.




Komentář k odkazu v bublině

Z předchozího příkladu je zjevné, že komentář u odkazu v žádném případě nenaruší vzhled stránky. Kdo si bude chtít komentář přečíst, ten si ho také přečte, a kdo ne, nechá jej bez povšimnutí.

Komentář lze také zobrazit u odkazu, který představuje obrázek, avšak pouze tehdy, není-li definován popisek parametrem **ALT** značky **IMG**. Pokud ano, má přednost popisek obrázku a komentář odkazu se nezobrazí.

## 3.6.5 Otevření stránky do nového okna

Určitě jste se setkali s tím, že po kliknutí na odkaz se stránka otevře v novém okně, aby ta původní zůstala viditelná, místo aby se načetla do okna původního, jak je zvykem. Toho se dociluje posledním velice důležitým parametrem **TARGET**.
Abyste tedy dosáhli výše zmíněného efektu, přidejte do značky **A**, která odkaz definuje, parametr

```
TARGET="_blank"
```

Další hodnoty parametru **TARGET** souvisí s rámy, kterými se podrobně zabývá kniha *Tvorba WWW stránek pro úplné začátečníky*, proto je zde nebudu dále rozebírat.

Takže to bychom měli. Teď máte základy vytváření stránek v HTML, umíte ovládat Internet Explorer. Není nic jednoduššího než jít dále. Předmětem další kapitoly budou formuláře, jak se tvoří a k čemu slouží, jak vytvořit a začlenit skript vytvořený ve skriptovacím jazyku do webové stránky a základy práce s kaskádovými styly, CSS.

# Část druhá: První kroky do života stránky

**Formuláře v HTML**

**Skripty – interaktivita na Internetu**

**Kaskádové styly**

Druhá část této knihy je určena těm, kteří mají pouze základní znalosti, zhruba ty, co byly zrekapitulovány ve třetí kapitole předchozí části knihy, popř. těm, kteří mají znalosti zhruba v rozsahu předchozího dílu, *Tvorba WWW stránek pro úplné začátečníky*. Půjde tu o práci s formuláři, které poskytují tvůrci webové stránky základní možnost odezvy návštěvníků, o seznámení se skripty a jejich začlenění do webové stránky a konečně o kaskádové styly, které samy o sobě vnesou do vašich stránek život. Je to takový rychlý start do virtuálního světa Internetu plného pohybu a života.

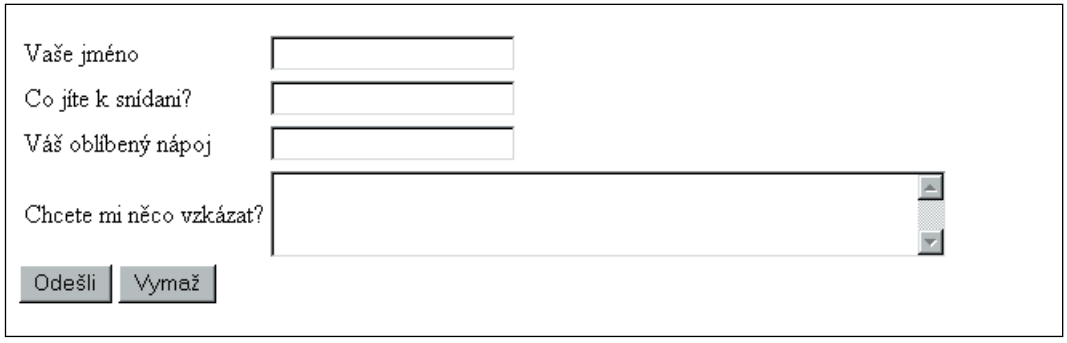
# 1. Formuláře v HTML

Skripty se často využívají ke zpracování dat z formulářů, prvku webové stránky, který umožňuje základní feedback mezi vámi a uživateli vašich stránek. Určitě jste se již s nimi setkali, kdykoliv jste např. hlasovali v nějaké v anketě (např. jak se vám stránky líbí nebo jaký máte názor na nějakou věc), mohli jste tak posílat zprávy do e-mailové schránky tvůrce stránky nebo třeba nakupovat zboží v internetovém obchodě. V této krátké kapitole popíšu základní prvky výstavby formulářů – tlačítka, okna pro vepsání textu a roletky, zatím bez jakéhokoliv programování. To přijde na řadu až v příští části věnované JavaScriptu. Zatím tedy budou sloužit pouze k vytvoření vzhledu stránky a k vytvoření formuláře jako takového.

Pokud se podíváte na následující příklad, uvidíte, k čemu lze také takový formulář využít. Je to lepší než jakékoliv vysvětlování a popis.

```
<HTML>
<HEAD>
  <TITLE>Formulář</TITLE>
</HEAD>
<BODY>
<FORM NAME="Formular">
<TABLE>
  <TR>
    <TD>Vaše jméno</TD>
    <TD><INPUT TYPE="text" NAME="Jmeno"></TD>
  </TR>
  <TR>
    <TD>Co jíte k snídani?</TD>
    <TD><INPUT TYPE="text" NAME="Snidane"></TD>
  </TR>
  <TR>
    <TD>Váš oblíbený nápoj</TD>
    <TD><INPUT TYPE="text" NAME="Napoj"></TD>
  </TR>
  <TR>
    <TD>Chcete mi něco vzkázat?</TD>
    <TD><TEXTAREA NAME="Vzkaz" COLS="50" ROWS="3"></TEXTAREA>
```

```
  </TR>
</TABLE>
<INPUT TYPE="submit" VALUE="Odešli">
<INPUT TYPE="reset" VALUE="Vymaž">
</FORM>
</BODY>
</HTML>
```




Takto může vypadat jednoduchý formulář na webové stránce

Jak takový formulář může vypadat, víte. Při letmém pohledu na zdrojový kód a výsledné zobrazení v prohlížeči vidíte, že se skládá z tabulky, která jej formátuje (ale není pro samotný formulář důležitá), a ze značek definujících jednotlivá políčka a tlačítka.

Celý formulář je ohraničen párovou značkou **<FORM></FORM>**, která zejména definuje jméno formuláře parametrem **NAME**. Parametr **NAME** je také důležitý u všech prvků vlastního formuláře. Umožňuje jeho provázání s akcí definovanou skriptem. Pojmenováním prvku se tak prvek stává objektem, kterému uživatel přiřazuje svojí odezvou nějakou hodnotu, nějaké vlastnosti. Můžete tak třeba napsat skript, který provede přesměrování na nějakou jinou webovou stránku v případě, že klepnete na tlačítko *Další*.

Parametr **NAME** se používá jednotlivých položek formuláře, abychom mohli později vyhodnocovat i hodnoty jednotlivých položek. I tyto položky formuláře vytvoří na stránce objekty, jejichž hodnoty můžeme číst, zpracovávat a dokonce ovlivňovat.

Značka **FORM** má i další parametry, které např. definují akce, jež s formulářem zamýšlíte (např. zaslání dat elektronickou poštou nebo zapsání do databáze). Těm se budu věnovat až v další kapitole, zatím by to bylo jen mlácení prázdné slámy, bez jakékoliv vazby.

# 1.1 Textová políčka

Základním prvkem formuláře je textové políčko, do něhož se vepisují požadované informace. V předchozím případě to byla například otázka na oblíbený nápoj:

```
<INPUT TYPE="text" NAME="Napoj">
```

Značkou, která políčko definuje, je tedy

```
<INPUT>
```

Jak je patrné z překladu anglického názvu značky, jde o značku, která očekává vstup od uživatele. Ať už je to text napsaný z klávesnice, nebo klepnutí na tlačítko. **INPUT** má mnoho různých parametrů, nyní se zaměřím pouze na ty, které umožňují definovat textové políčko. Prvním takovým parametrem je

```
TYPE="text"
```

**Poznámka:** *Parametr **TYPE** obecně určuje charakter, typ políčka. Může jít buď o textové políčko, tlačítko, zaškrtávací políčko (tzv. checkbox) apod.*

Políčku můžete také navolit jeho velikost, která se zadává parametrem

```
SIZE="délka"
```

Pokud nezadáte velikost políčka, bude nastavena standardní délka zhruba 20 písmen. Protože však můžete psát do políčka text delší, než je jeho velikost, lze nastavit i maximální počet znaků, které lze do políčka vepsat. Tohoto se dociluje parametrem

```
MAXLENGTH="počet slov"
```

Do políčka pak nelze zapsat text delší, než jste zadali. Samozřejmě že při zadávání parametru **MAXLENGTH** nejste omezeni velikostí políčka, což by bylo trošku nelogické. Políčko tedy může být klidně kratší než počet zadávaných znaků.

Zajímavým doplňkem definice textového políčka je možnost přednastavit text, který bude v políčku vepsán již při příchodu uživatele na zadanou stránku, a sice parametrem

```
VALUE="přednastavený text"
```

Celá definice textového políčka může vypadat např. takto:

```
<INPUT TYPE="text" SIZE="50" MAXLENGTH="30" VALUE="Petr Broža">
```


Do textového políčka lze vložit přednastavenou hodnotu

**Tip:** *Chcete-li použít políčka např. pro nastavení hesla, kdy nechcete, aby někdo mohl text do políčka napsaný (heslo) opsat, použijte značku INPUT s parametrem*

**TYPE="password"**

*To, co píšete, se bude zobrazuje jako série hvězdiček. I zde je možné dodat implicitní text, ten se také zobrazí jako hvězdičky.*

Políčko typu *password* se hodí pro zadávání hesla

# 1.2 Políčka o větším počtu řádků

Nestačí-li vám jednořádkové pole, můžete použít značky

**TEXTAREA**

Umožňuje vytvořit pole o libovolném počtu řádku. To se hodí např. tehdy, když chcete, aby vám uživatel napsal svůj názor, vyslovil kritiku nebo naopak pochvalu. Psát delší text do jednořádkového pole je totiž krajně nepřehledné. Na rozdíl od značky **INPUT**, která je nepárová, je **TEXTAREA** značkou párovou. Mezi značkami je totiž uložen implicitní text, který bude do textové oblasti vyplněn ještě před návštěvou stránky uživatelem.

```
<TEXTAREA>Implicitní text</TEXTAREA>
```

Značka **TEXTAREA** má podobné parametry jako jednořádkové políčko (**<INPUT TYPE="text">**), nezadává se však typ prvku; ten je totiž přímo určen značkou. Stejně jako u políčka má víceřádkové pole svoje jméno (**NAME**) a velikost. Ta se však zadává trošku jiným způsobem – parametry

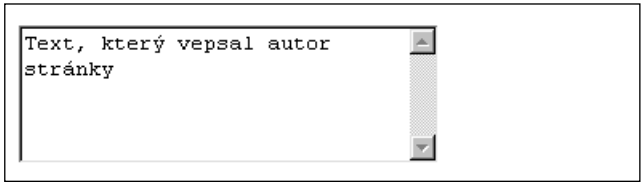
```
COLS="počet sloupců"
ROWS="počet řádků"
```

Parametr **COLS** je defacto shodný s parametrem **SIZE** u jednořádkového políčka – udává jeho šířku (počet sloupců) ve znacích. Navíc tu je parametr **ROWS**, který udává výšku (počet řádků) oblasti, opět ve znacích. Chcete-li např. textovou oblast s šířkou 30 znaků a výškou 5 znaků, napíšete do zdrojového kódu:

```
<TEXTAREA COLS="30" ROWS="5">Text, který vepsal autor
stránky</TEXTAREA>
```

Jistě jste si všimli, že při psaní textu delšího než je jeden řádek, se text automaticky zalamuje. To je dáno implicitním nastavením parametru

**WRAP="on"**




Značka TEXTAREA – textové pole o větším počtu řádku

Pokud chcete zalamování vypnout (i když, proč byste to dělali – tehdy je lepší použít klasického jednořádkového políčka), zadejte jako hodnotu parametru **WRAP="off"**.

# 1.3 Roletky – výběr ze seznamu

Někdy není nutné všechny hodnoty vepisovat do formulářů ručně, jsou totiž vybírány z několika přednastavených hodnot. Máte-li třeba dotaz na velikost monitoru, víte, že mohou být pouze 14–24palcové. V tomto případě se nehodí používat klasických textových políček, ale roletku s výběrem možných voleb. V praxi to vypadá tak, že vedle textového políčka se objeví i malá šipka ukazující směrem dolů. Pokud na ni klepnete myší, ukáže se seznam možných hodnot, z nichž lze jednu vybrat.

K tomu slouží značka

```
<SELECT NAME="jméno výběru">
```

Následuje seznam položek, které lze ze seznamu vybrat; ty jsou definovány další značkou, a to ve sledu, v jakém budou zobrazeny. Ty jsou uzavřeny v párové značce **OPTION**:

```
<OPTION>Výběr 1</OPTION>
<OPTION>Výběr 2</OPTION>
<OPTION>Výběr 3</OPTION>
```

Následuje ukončení celého seznamu:

```
</SELECT>
```

Jde tedy o párovou značku, kde jsou jednotlivé položky voleb uzavřeny otevírající a uzavírající značkou **SELECT**. Zajímavým je parametrem značky **SELECT** je

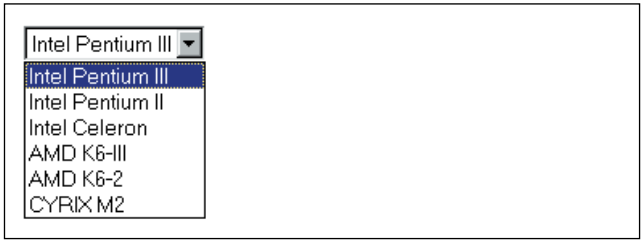
```
MULTIPLE
```

který určuje, že půjde vybrat více položek seznamu současně (přidržením klávesy CTRL nebo SHIFT, podle standardu Windows). V praxi se však tento parametr moc nepoužívá, seznam se většinou používá tam, kde je možná pouze jedna položka, a mnoho uživatelů vlastně ani neví, že může tímto způsobem vybrat položek více, pokud to není explicitně řečeno. Pokud chcete nabídnout uživateli více možností, použijte raději zatrhávací políčka, tzv. checkboxy (viz dále).

Část formuláře, v níž je obsažen výběr ze seznamu, může vypadat třeba takto:

```
<SELECT NAME="Procesor">
  <OPTION>Intel Pentium III</OPTION>
  <OPTION>Intel Pentium II</OPTION>
  <OPTION>Intel Celeron</OPTION>
  <OPTION>AMD K6-III</OPTION>
  <OPTION>AMD K6-2</OPTION>
  <OPTION>CYRIX M2</OPTION>
</SELECT>
```

Tento kousek kódu HTML vytvoří seznam, který můžete vidět na obrázku.

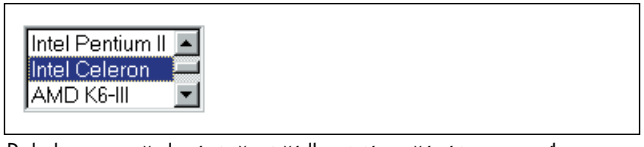



Výběr položek (typ procesoru) ze seznamu

V praxi však není potřeba používat uzavírací značky **</OPTION>**, stejně jako není nutné používat ukončovací značky u definice políček tabulky **TD** nebo **TR**. Ukončení roletkového výběru **</SELECT>** je však nutné použít.

Dalšími zajímavými a mnohdy užitečnými parametry značky **SELECT** jsou:

- **SIZE="n"** – Počet řádků, které jsou v roletce implicitně zobrazeny. Roletka tak má předem danou velikost a je-li počet položek větší, vytvoří se v pravé části roletky posuvník, kterým můžete seznamem rolovat. Z vysouvací roletky se tak vytvoří roletka se stálou velikostí. Je to vidět na obrázku, kde je vytvořena roletka s pevně danými třemi řádky, tj. **SIZE="3"**.




Roletka s pevně danými třemi řádky; její součástí je posuvník

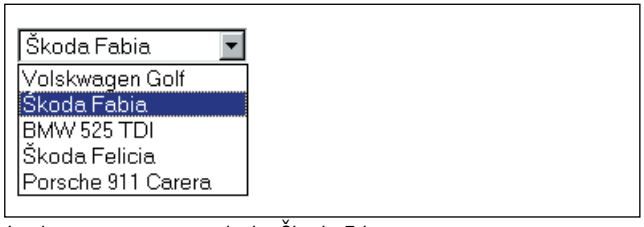
- **DISABLED** – Použití roletky bude zakázáno, bude zobrazená zašedlou barvou (stejně jako u nepoužitelné položky v nabídce Wordu nebo Excelu).


Výběr je po použití parametru DISABLED zašedlý a nelze jej použít

I značka **OPTION**, která definuje jednu položku seznamu, má svoje parametry, které nemohu pominout. A sice:

- **SELECTED** – Hodnota takto označená je implicitně vybrána, tj. je nastavena do okna seznamu a zbarvena tak, jako kdyby na ni uživatel klepl.
- **VALUE –** Parametr **VALUE** předává serveru nebo skriptu jinou hodnotu, než je zobrazena uživateli. Vidí-li uživatel v seznamu položek v roletce třeba „Automobil Volkswagen“, obdrží server místo této hodnoty třeba jen „Volks“. Nejjednodušeji to vysvětlí tento příklad (a také předchozí parametr, **SELECTED**):

```
<SELECT NAME="Automobily">
  <OPTION VALUE="Golf">Volskwagen Golf</OPTION>
  <OPTION SELECTED>Škoda Fabia</OPTION>
  <OPTION>BMW 525 TDI</OPTION>
  <OPTION>Škoda Felicia</OPTION>
  <OPTION>Porsche 911 Carera</OPTION>
</SELECT>
```




Implitně označená je položka Škoda Fabia

Z obrázku je patrné, že implicitně je vybraná hodnota seznamu „Škoda Fabia“, protože má ve značce **OPTION** umístěn parametru **SELECTED**.

Ale zpět k parametru **VALUE**. Vybere-li ze seznamu položku „BMW 525 TDI“, je tato hodnota, tento celý řetězec předán skriptu nebo serveru pro zpracování. Totéž platí i pro Fabii, Felicii a Careru. Pokud však zvolíte „Volkswagen Golf“, bude skriptu nebo serveru předána pouze hodnota „Golf“, protože je-li přítomen parametr **VALUE**, má jeho hodnota přednost a tudíž je řetězec „Volkswagen Golf“ ignorován. Parametr **VALUE** se tedy hodí všude tam, kde není potřeba serveru předávat žádné dlouhé řetězce dat, navíc to mnohdy zjednodušuje zpracování.

# 1.4 Přepínače, zatrhávaná políčka

Dalším hodně používaným prvkem formulářů jsou bezesporu zatrhávací políčka. Ta se používají především tam, kde stačí pouze souhlasit nebo nesouhlasit s daným tvrzením, není potřeba nic sáhodlouze vypisovat a použít výběr (**SELECT**) pouze pro hodnoty *ano/ne* je zbytečné.

Pro zobrazení zatržítka se používá klasická značka **INPUT**, pouze s jinou hodnotou parametru **TYPE**:

```
<INPUT TYPE="checkbox">
```

Toto jednoduché zatržítko má kromě klasických parametrů společných pro značku **INPUT** ještě jeden specifický, **CHECKED**, který určuje, zda má být políčko implicitně zatrhnuto či nikoliv. Je-li parametr přítomen, bude zaškrtnuto, jinak ne.

```
<INPUT TYPE="checkbox" CHECKED>
```

Zatrhnutí políčka není nevratné; pokud je jednou zaškrtnete, lze jej opět odškrtnout a naopak. Nejlépe funkci vysvětlí následující příklad – můžeme jej zařadit třeba do jednoduchého dotazníku:

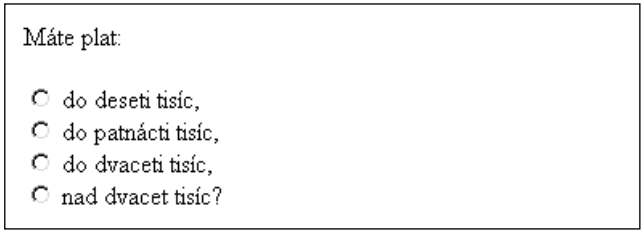
```
<INPUT TYPE="checkbox" CHECKED NAME="Dotaz"> Máte rádi maso?<BR>
<INPUT TYPE="checkbox" NAME="Dotaz2"> Máte rádi zeleninu?<BR>
```


Zeptejte, co lidé rádi jedí

Druhým typem zaškrtávacích tlačítek je tzv. *radiobutton*, který podobně jako *checkbox* může mít dva stavy: zaškrtnuto a nezaškrtnuto. Používá se však odlišně: jako výběr jednoho z mnoha. Pokud třeba chcete zjistit, v jakých mezích se pohybuje uživatelův plat, vymezíte např. tyto hranice:

1) do deseti tisíc,
2) do patnácti tisíc,
3) do dvaceti tisíc,
4) nad dvacet tisíc.

Pokud necháte pomocí radiobuttonů tento výběr zobrazit na stránce, může pak uživatel klepnutím myši na kolečko zatrhnout jeden z výběrů, výběrem jiného rozmezí se původní výběr zruší. Nelze tedy zaškrtnout více výběrů naráz.




Před prvním výběrem není zaškrtnuta žádná položka

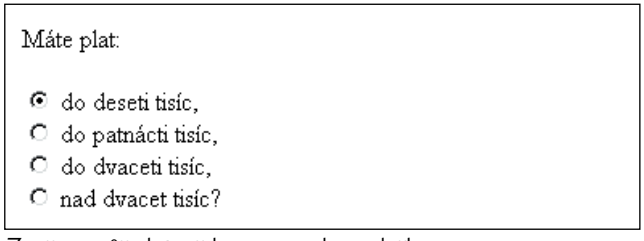
V praxi se toho dosahuje hodnotou radio parametru **TYPE** značky **INPUT**:

```
<INPUT TYPE="radio">
```

Každý jeden takový řádek vytvoří jedno „kolečko“, tedy jednu možnou volbu z celého seznamu. Pokud tedy je takových voleb více (např. u příkladu výběru platového zařazení), je nutné použít odpovídající počet značek **INPUT**. U každé značky samozřejmě nesmí chybět parametr **NAME** se stejnou hodnotou, aby prohlížeč poznal, že všechny výběry patří k sobě:

```
<BODY>
  <P>Máte plat:</P>
  <INPUT TYPE="radio" NAME="Plat"> do deseti tisíc,<BR>
  <INPUT TYPE="radio" NAME="Plat"> do patnácti tisíc,<BR>
  <INPUT TYPE="radio" NAME="Plat"> do dvaceti tisíc,<BR>
  <INPUT TYPE="radio" NAME="Plat"> nad dvacet tisíc?<BR>
</BODY>
```

Pokud bude zdrojový text vypadat takto, nebude zatrženo zpočátku žádné kolečko. Pokud chcete vytvořit nějaké implicitní nastavení, přidejte k některé značce **INPUT** parametr **CHECKED**, který má stejný význam jako u zatrhávacích políček.




Zatržena může být vždy pouze jedna položka

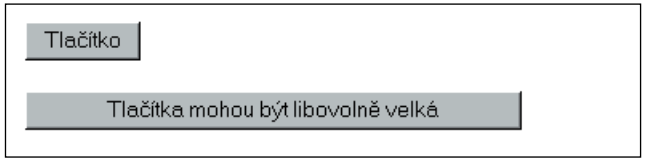
**POZNÁMKA:** *Spíše než radiobuttony je však lepší využívat klasické výběry ze seznamu (**SELECT**), šetří totiž místo ve formuláři a nabízí prakticky stejné možnosti. Radiobuttony se hodí pouze tam, kde zvýší přehlednost, a není mnoho možností na výběr.*

# 1.5 Tlačítka formulářů

Po vyplnění formuláře musí dát uživatel na vědomí prohlížeči, že může s obsahem formuláře dále pracovat. Buďto obsah odeslat serveru nebo předat nějakému řídicímu skriptu. To zpravidla provádí klepnutím na tlačítko, které bývá umístěno na konci formuláře. To se definuje opět klasickým způsobem, značkou **INPUT**:

```
<INPUT TYPE="button" VALUE="Tlačítko">
```

Toto je základní, jednoduché tlačítko, s jehož stisknutím je ve skriptu spojena nějaká další akce, např. zpracování obsahu formuláře.




Jednoduchá tlačítka. Text na spodním hovoří sám za sebe.

Velikost tlačítka se přizpůsobí velikost textu, který je uveden jako hodnota parametru **VALUE**. Pokud ji napíšete třeba takto:

```
<INPUT TYPE="button" VALUE="    Tlačítko    ">
```

zvětší se zároveň i velikost tohoto tlačítka (viz předchozí obrázek).

Další možnosti tlačítek jsou následující:

- **TYPE="reset"** – Tlačítko, které se používá při resetování (vymazávání, nastavení na původní hodnotu) všech prvků formuláře.
- **TYPE="submit"** – Tlačítko, po jehož stisknutí se obsah vyplněného formuláře odesílá směrem k serveru.

Tlačítko lze také definovat značkou **BUTTON**, jehož možnosti formátování zdaleka přesahují možnosti obyčejného tlačítka, jež je definováno přes **<INPUT TYPE="button">**. Jeho syntaxe je:

```
<BUTTON NAME="jméno" VALUE="hodnota"
TYPE="typ">obsah tlačítka</BUTTON>
```

Jednotlivé parametry mají tento význam (stručně):

- **NAME** – Jméno tlačítka, na které se odkazují skripty, které jsou s tlačítkem provázány.

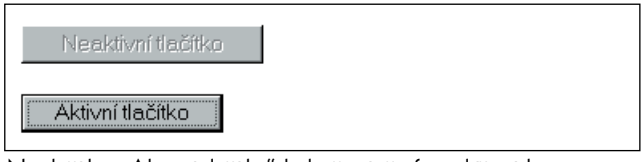
- **VALUE** – Hodnota předaná serveru, když je tlačítko stisknuto.
- **TYPE** – Tlačítko může být jedním z těchto tří typů, i když navenek vypadá stále stejně:
  - **button** – spouští zadaný skript;
  - **submit** – odesílá veškeré vyplněné hodnoty formuláře na server, např. jako elektronickou poštu;
  - **reset** – nastaví vstupní hodnoty celého formuláře, neboli resetuje všechny hodnoty, nic neodesílá ani nic neprovádí.
- **DISABLED** – Toto tlačítko nepůjde použít, bude zašedlé a nepůjde na ně klepnout myší; bude zobrazeno šedě (pracuje i předchozí definice tlačítka).


Příklad neaktivního tlačítka

Dalšími používanými parametry pak jsou (platí i u předchozí definice tlačítka):

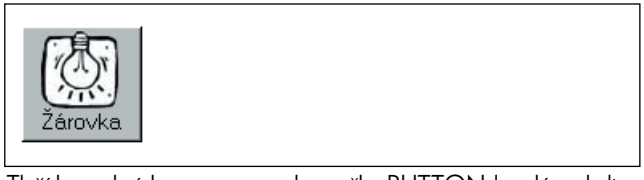
- **ONFOCUS=skript** – Skript, který se provádí, když uživatel umístí formulářový kurzor na tlačítko.
- **ONBLUR=skript** – Skript, který se provádí, když uživatel umístí formulářový kurzor mimo tlačítko.

**POZNÁMKA:** *Kurzor, který tady mám na mysli, je ten, který umožňuje vyplňovat formulář. Lze jej na tlačítko (nebo na jakýkoliv prvek formuláře) přesunout např. tabulátorem nebo klávesou* ENTER*, ale i klepnutím myši. U tlačítka je kurzor vyznačen tečkovaným obrysem.*




Na tlačítko „Aktivní tlačítko" byl přemístěn formulářový kurzor

Vlastní obsah tlačítka, tedy to, co je umístěno mezi značkami **<BUTTON></BUTTON>**, může tvořit cokoliv, formátovaný i víceřádkový text nebo třeba obrázek.



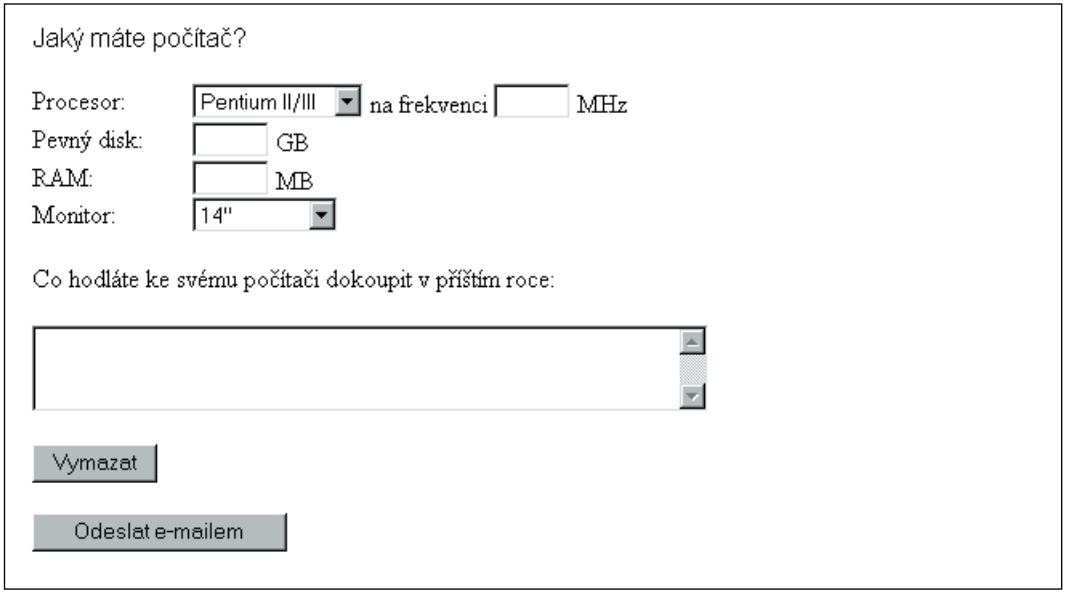

Tlačítko s obrázkem a textem; do značky BUTTON lze dát cokoliv

## 1.6 A jak je to v praxi?

To, co jste dosud četli, je vlastně jen prázdná teorie, která nedojde naplnění, pokud nebude propojena s patřičným skriptem. Příkladem takového formuláře, který opravdu něco dělá, je následující anketa. Ta po vyplnění patřičných políček (dotazů) zpracuje hodnoty, vypíše je v okně a následně je odešle e-mailem na adresu *broza@cpress.cz*. Pokud budete příklad pozorně zkoumat, můžete přijít i na základy JavaScriptu a jeho provázání s webovou stránkou. To všechno však vysvětlím podrobně až v následující kapitole.

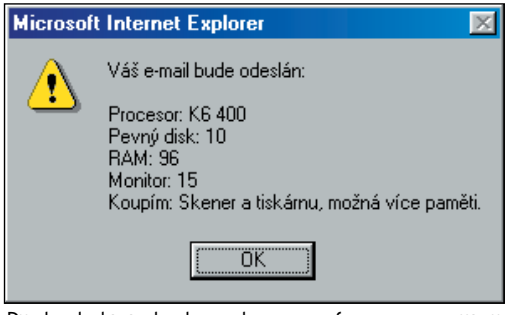
```
<HTML>
<HEAD>
  <TITLE>Anketa</TITLE>

  <SCRIPT LANGUAGE="JavaScript">
  <!--

  function mailIt(form) {
    var data = document.ANKETA;
    var info="";

    info+="Procesor: "+data.procesor.value+" ";
    info+=data.freq.value + "\n";
    info+="Pevný disk: "+data.hdd.value + "\n";
    info+="RAM: "+data.ram.value + "\n";
    info+="Monitor: "+data.monitor.value + "\n";
    info+="Koupím: "+data.chci.value + "\n";

    form.mailBody.value = info;

    alert("Váš e-mail bude odeslán:\n\n"+ info);
    return true
  }
  // -->
  </SCRIPT>
</HEAD>

<BODY>
  <FORM NAME="ANKETA">
  <P><FONT FACE="Arial" SIZE="3">Jaký máte počítač?</FONT></P>
  <TABLE WIDTH="480" BORDER="0" CELLSPACING="0" CELLPADDING="0">
  <TR>
    <TD WIDTH=100>Procesor: </TD>
    <TD><SELECT NAME="procesor" >
```

```
        <OPTION VALUE="PII/III">Pentium II/III</OPTION>
        <OPTION VALUE="P/PMMX">Pentium MMX</OPTION>
        <OPTION VALUE="C">Celeron</OPTION>
        <OPTION VALUE="A">Athlon</OPTION>
        <OPTION VALUE="K6">K6/K6-2/K6-III</OPTION>
        </SELECT>
        na frekvenci
        <INPUT TYPE="Text" NAME="freq" SIZE="5"> MHz
    </TD>
  </TR>
  <TR>
    <TD>Pevný disk: </TD>
    <TD><INPUT TYPE="Text" NAME="hdd" SIZE="5"> GB<br></TD>
  </TR>
  <TR>
    <TD>RAM: </TD>
    <TD><INPUT TYPE="Text" NAME="ram" SIZE="5"> MB<BR></TD>
  </TR>
  <TR>
    <TD>Monitor: </TD>
    <TD><SELECT NAME="monitor">
        <OPTION VALUE="14">14"</OPTION>
        <OPTION VALUE="15">15"</OPTION>
        <OPTION VALUE="17">17 a vyšší"</OPTION>
        <OPTION VALUE="Jiný">Jiný</OPTION>
        </SELECT><BR>
    </TD>
  </TR>
  </TABLE>
  <BR>
  <P>Co hodláte ke svému počítači dokoupit v příštím roce:</P>
  <TEXTAREA NAME="chci" COLS="50" ROWS="3"></TEXTAREA>
  <BR><BR>
  <INPUT TYPE="reset" VALUE="Vymazat">
  </FORM>

  <FORM ACTION="mailto:broza@cpress.cz?subject=Anketa"
        METHOD="post" ENCTYPE="text/plain" NAME="mailForm"
        ONSUBMIT="return mailIt(this)">
  <INPUT TYPE="hidden" NAME="mailBody" VALUE>
  <INPUT TYPE="submit" VALUE="Odeslat e-mailem">
  </FORM>
</BODY>
</HTML>
```




Zobrazení formuláře v prohlížeči

Klepnete-li na tlačítko *Vymazat*, budou všechna textová políčka vymazána a u jednotlivých výběrů se nastaví původní hodnoty. Tlačítkem *Odeslat e-mailem* se spustí skript, který je definován v hlavičce stránky, který z jednotlivých políček ankety sestaví řetězec, který následně formulář odešle e-mailem přes nastavený klient elektronické pošty. Podrobně bude způsob odesílání formulářů popsán v šesté kapitole třetí části.




Před odesláním budou zobrazeny informace o počítači

**Poznámka:** *V příkladu se objevil prvek formuláře typu* hidden. *V podstatě jde o prázdný prvek, který nic ve formuláři nezobrazí. Má však svoji hodnotu a využívá se ho tím, že jeho hodnotu lze modifikovat. Více v příští části v kapitole 6 věnované odesílání formulářů.*

# 2. Skripty – interaktivita webu

Jak už jste určitě pochopili z předchozí části knihy, jsou skripty hlavním nástrojem pro interaktivní webové stránky. Nebudu zde opakovat, co skripty jsou a jak fungují, to bylo popsáno v první části (1.1), koneckonců v předchozí kapitole (1.6) jsem schválně uvedl příklad formuláře i se skriptem, který jej zpracovává – z něj byla struktura stránky, myslím, docela jasná. Nyní se už zaměřím přímo na konkrétní začlenění skriptu do stránky a způsoby, jakými skripty běží a kdy a za jakých okolností jsou zpracovány.

## 2.1 Jak integrovat skript do webové stránky

Celý program napsaný v nějakém skriptu musí být ve zdrojovém kódu HTML označen párovou značkou:

```
<SCRIPT></SCRIPT>
```

Parametrem této značky je pak jazyk (VBScript, JavaScript), v němž je skript napsán, aby jej prohlížeč mohl dobře interpretovat. Tímto parametrem je

```
LANGUAGE="skript"
```

jehož hodnotou je název jazyka, v němž je vytvořen program, který bezprostředně následuje. Pokud je takovýmto jazykem JavaScript, bude značka **SCRIPT** vypadat takto:

```
<SCRIPT LANGUAGE="JavaScript">
```

bude-li jím Visual Basic Script, bude značka vypadat pro změnu takto:

```
<SCRIPT LANGUAGE="VBScript">
```

Značka **SCRIPT** může mít uvedeny i další dva parametry, kterými se však nebudu v této knize zabývat, proto je uvádím pouze na okraj:

- **SRC="url"** – Skript nemusí být nutně součástí zdrojového kódu vaší stránky, může být uložen i do externího souboru (s patřičnou příponou, např. pro JavaScript je to JS). Z něho pak bude skript přečten ve chvíli, kdy dojde na jeho provádění.
- **RUNAT="server"** – Parametr **RUNAT** s hodnotou server říká, že program bude spuštěn už na serveru a do stránky budou vloženy až výsledky jeho činnosti.

Protože však skript není přímo součástí HTML, tedy toho, co se má na stránce zobrazit, je dobré jej uzavřít do poznámky. Tento krok se provádí kvůli starším prohlížečům, které se skripty neumí pracovat. Ovšem prohlížeče nové (IE 5.0) vědí, že jde o skript, takže jej do komentářů uzavírat nemusíte – bude proveden, nikoliv zobrazen.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  Vlastní program skriptu
-->
</SCRIPT>
```

**Poznámka:** *Moderní prohlížeče už tak striktně tyto komentáře nevyžadují, používat je tedy není nezbytně nutné, ale doporučuje se to kvůli kompatibilitě s prohlížeči staršími.*

Vlastní skript ohraničený patřičnými značkami lze zapsat do zdrojového kódu stránky prakticky kamkoliv. Může být zapsán dokonce do sekce HEAD; je totiž ohraničen značkami pro komentář, takže je to prohlížeči jedno, kde bude. Záleží také na tom, kdy má být skript zpracován. Má-li být proveden ihned po načtení stránky do prohlížeče, můžete jej umístit třeba takto:

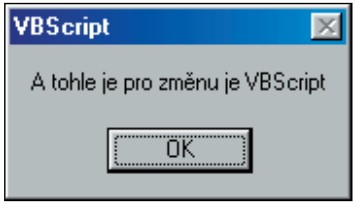
```
<HTML>
<HEAD>
  <TITLE>První program</TITLE>

  <SCRIPT LANGUAGE="JavaScript">
  <!--

    alert("Tohle je JavaScript");

  -->
  </SCRIPT>

</HEAD>

<BODY>
</BODY>
</HTML>
```

Ihned po spuštění JavaScriptu se objeví okno se zprávou „Tohle je JavaScript“, po odklepnutí myší se bude dále pokračovat v zobrazení stránky podle značek umístěných v sekci BODY. V tomto příkladě však žádné nejsou, stránka tedy zůstane prázdná.


Upozornění JavaScriptu

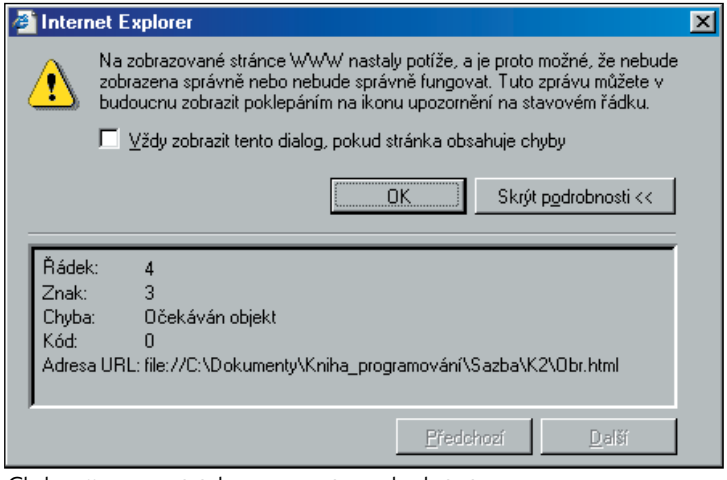
V příkladu byl použit příkaz JavaScriptu, stejného efektu lze však dosáhnout i použitím VBScriptu:

```
<SCRIPT LANGUAGE="VBScript">
<!--

  MsgBox("A tohle je pro změnu je VBScript")

-->
</SCRIPT>
```




Upozornění VBScriptu

Z tohoto je patrné, že nezáleží na použitém skriptovacím jazyku, použitý skript však musí být správně uveden v parametru **LANGUAGE** – příkaz *MsgBox* je totiž příkazem VBScriptu. Pokud však hodnotou parametru **LANGUAGE** bude „JavaScript“, vyvolá to chybu interpretace, což nám Internet Explorer oznámí takto:




Chyba při zpracování skriptu na stránce s konkrétním popisem

Skript může být uložen i v těle stránky, v sekci BODY. Tehdy čas jeho provedení bude záviset na umístění. Pro lepší pochopení opět příklad.

```
<BODY>
Ahoj poprvé
  <SCRIPT LANGUAGE="JavaScript">
  <!--
    alert("Hlášení JavaScriptu");
  // -->
  </SCRIPT>
Ahoj podruhé
</BODY>
```

Nejdříve se na stránku vypíše text „Ahoj poprvé“, poté se zobrazí v okně text „Hlášení JavaScriptu“, a teprve pak se vypíše na stránku text „Ahoj podruhé“. Z toho je patrné, že skript může být prakticky kdekoliv a může jich být ve stránce libovolné množství, a lze dokonce i kombinovat jak JavaScript, tak VBScript, podle toho, co lépe vyhovuje.

## 2.2 Události prohlížeče

Čas, kdy se skript provede, nemusí být vázán pouze na umístění daného skriptu ve zdrojovém kódu HTML. Vezměme třeba příklad ankety, kdy je skript spuštěn až ve chvíli, kdy uživatel klepne na tlačítko odeslat. Nebo když ve stránce najedete kurzorem myši na obrázek, který se rozsvítí nebo jinak reaguje a když myší uhnete stranou, opět zhasne (neboli provede se výměna obrázků, z nichž jeden je normální a druhý jasnější). Jde tedy o provedení skriptu v případě, že se splní daná podmínka, nastane určitá *událost*. A právě tyto události jsou tím, co dělá web dynamickým.

Pro lepší pochopení rozvedu právě příklad rozsvícení a zhasnutí obrázku při přejetí myší: Nejdříve vytvoříte dva obrázky, jeden bude tmavý a druhý jasný. Ten tmavý se bude jmenovat *obrazek.jpg*, ten světlý *obrazek_cool.jpg*. Tmavý bude zobrazen v případě, že myš ne-bude ukazovat na něj. Druhý, jasný, se zobrazí tehdy, přejede-li přes něj kurzor myši.

Následně vložte tento skript do sekce HEAD, nebude však vadit, umístíte-li jej kamkoliv jinam (je napsán v JavaScriptu):

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  function MakeCool(Image) {Image.src=Image.src.substring
      (0,Image.src.length-4)+"_cool"+
      Image.src.substring(Image.src.length-4,Image.src.length);}
  function MakeNormal(Image) {Image.src=Image.src.substring
      (0,Image.src.length-9)+Image.src.substring
      (Image.src.length-4,Image.src.length);}
-->
</SCRIPT>
```

A pak už jen vložte místo klasického odkazu na obrázek **<IMG SRC="Obrazek.jpg">** vložte řádek tento:

```
<IMG ID="Obrazek" ONMOUSEOVER="MakeCool(Obrazek)"
ONMOUSEOUT="MakeNormal(Obrazek)" SRC="obrazek.jpg" BORDER=0>
```

Kdykoliv nyní přejdete přes obrázek myší, jako by se rozsvítil. Tohle má význam např. u odkazů, kdy je takto aktivní obrázek velice efektní. V příkladu jsou těmito událostmi **onmouseover** a **onmouseout**, tedy událost, když myš je umístěna na objektu a když myš je umístěna mimo. Tyto události pak předají řízení uvedeným skriptům, které jsou definovány výše. Výhodou tohoto skriptu je navíc to, že obrázky mohou mít jakoukoliv příponu, tedy lze použít jakéhokoliv obrázkového formátu. Ale to jen tak na okraj.

Pro úplnost uvedu ještě tentýž skript ve VBScriptu, abyste si udělali představu, že to jde i tak:

```
<SCRIPT LANGUAGE="VBScript">
<!--
  Function MakeCool(Image)
    Image.Src = Left(Image.Src,Len(Image.Src)-4)&"_cool"
                &Right(Image.Src,4)
  End Function
  Function MakeNormal(Image)
    Image.Src = Left(Image.Src,Len(Image.Src)-9)
                &Right(Image.Src,4)
  End Function
-->
</SCRIPT>
```

Myslím, že použití událostních skriptů je jasné. Spustí pouze v případě, že je splněna daná podmínka. U této podmínky je pak také umístěno i jméno funkce, která převezme řízení její platnosti. Těchto událostí je celá řada, proto uvedu stručný přehled těch základních a nejčastěji používaných:

- **onmouseover** – Událost nastane ve chvíli, kdy se kurzor myši objeví nad objektem, není nutné stisknout tlačítko myši.
- **onmouseout** – Nastane tehdy, když je kurzor myši mimo objekt; doplňková událost k **onmouseover**.
- **onmouseover –** Událost nastává, když se kurzor pohybuje. Pokud se zastaví, je událost ukončena.
- **onclick –** Nastává v okamžiku, když uživatel klepne tlačítkem myši nad objektem.
- **ondblclick –** Totéž jako **onclick**, ovšem je nutné provést poklepání, neboli dvakrát za sebou stisknout tlačítko myši.
- **onload –** Událost nastane, když prohlížeč načte veškerá data stránky nebo rámu. Podle toho, zda se má jít o jeden rám nebo všechny, umísťuje se událost do sekce BODY nebo FRAMESET.
- **onunload** – Událost nastane, když uživatel ukončí prohlížení aktuální stránky a nechává si načíst stránku jinou. Používá se v sekcích BODY nebo FRAMESET.
- **onfocus** – Událost nastává, když je vybrán objekt formuláře – políčko, checkbox,

tlačítko apod. To nastává v okamžiku, kdy je buďto do textového políčka přesunut kurzor, kdy je klepnuto na tlačítko nebo zatržen *checkbox* nebo *radiobutton*.

- **onblur –** Přesný opak události **onfocus**.
- **onkeypress** – Událost nastává, když je nad objektem, tedy ve chvíli, když je nad ním kurzor myši nebo je právě aktivní, stisknuta klávesa. Opět lze použít u formulářových objektů.
- **onsubmit** – Událost nastává, když je formulář odeslán ke zpracování na server; je jasné, že může být proto použita pouze u značky **FORM**, která definuje formulář. S touto událostí jste se mohli setkat již u příkladu odesílání ankety e-mailem.
- **onreset** – Nastává ve chvíli, kdy je formulář „resetován", tj. když je stisknuto tlačítko typu reset (**BUTTON TYPE="reset"** nebo **INPUT TYPE="reset"**); opět samozřejmě platí jen pro formulář.
- **onselect** – Událost je aktuální, když uživatel vybere text v políčku formuláře (**INPUT**, **TEXTAREA**), je jedno, jestli myší nebo klávesou SHIFT a kurzorovými klávesami.
- **onchange** – Podobně jako **onselect**, událost je však aktivní ve chvíli, kdy uživatel ukončí editaci textu v políčku, např. stisknutím klávesy ENTER nebo přesunutím tabelátorem na další objekt.

**POZNÁMKA:** *Protože někteří uživatelé nemusí používat prohlížeč, který umí skripty, i když dnes už o tom silně pochybuji, byla vytvořena značka* **NOSCRIPT***, která uzavírá sekvenci značek pro případ, že daný prohlížeč skriptování nepodporuje:*

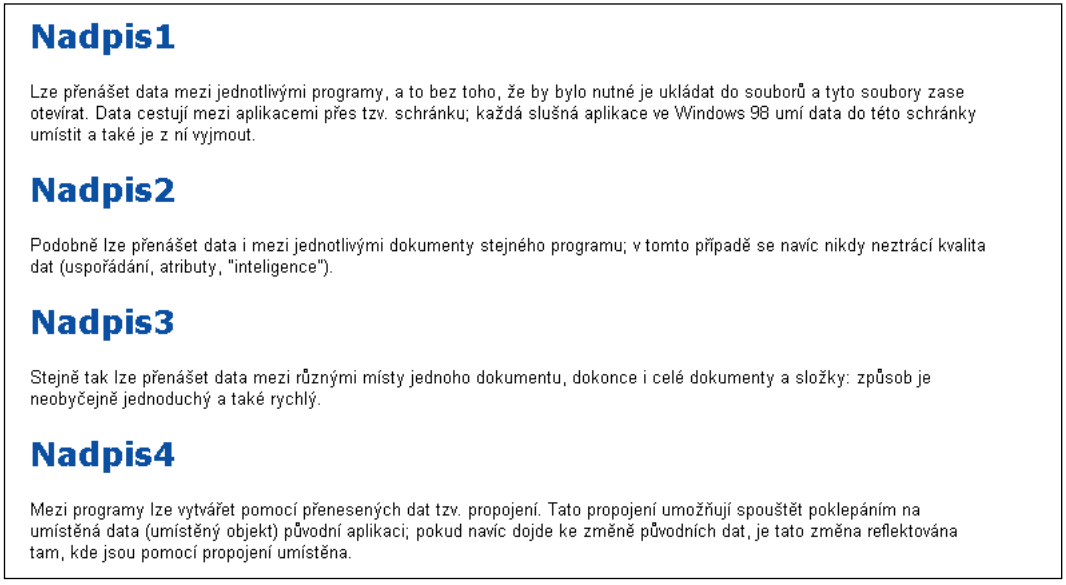
```
<SCRIPT LANGUAGE="VBScript">
<!--
  tělo skriptu
-->
</SCRIPT>

<NOSCRIPT>
  Upozornění, že prohlížeč nepodporuje skripty nebo jakákoliv
  jiná sekvence značek.
</NOSCRIPT>
```

# 3. Kaskádové styly

Jak byste se asi cítili, kdyby vám někdo pod ruku místo Wordu nebo jiného profesionálního textového editoru dal třeba jednoduchý Wordpad? Veškerá pohodlná práce by byla ta tam. Každý odstavec, každý nadpis musíte zvlášť formátovat, ve Wordpadu totiž neexistuje nic jako definice stylů, které jednou nastavíte a pak jen jednoduše používáte. Vše se musí dělat na místě s daný kouskem textu.

Tak jste se možná cítili, když jste vytvářeli svoje první stránky v HTML. Žádné vyšší možnosti formátování, u každého odstavce nutnost zadat opět parametry značky **FONT**, opakované definice stylů. Klasické HTML tohle prostě neumí a někdy to může docela dost vadit. Ostatně, posuďte sami u tohoto příkladu. Je to krátký text, kdy každý odstavec je naformátován stejně (písmo Arial CE, velikost 10 bodů, barva černá), mezi každým odstavcem je nadpis, všechny nadpisy jsou opět formátovány stejně (písmo Verdana, velikost 16 bodů, barva tmavě modrá). Protože HTML definuje velikost písma, než třeba Word, jsou zde použity standardní velikosti HTML: velikost 10 bodů reprezentuje velikost 2, velikost nadpisu 16 bodů reprezentuje značka **H2**:

```
<FONT FACE="Verdana" COLOR="blue"><H2>Nadpis1</H2></FONT>
<FONT FACE="Arial" SIZE="2"><P>Lze přenášet data mezi jednotlivý-
mi programy, a to bez toho, že by bylo nutné je ukládat do
souborů a tyto soubory zase otevírat. Data cestují mezi
aplikacemi přes tzv. schránku; každá slušná aplikace ve Windows
98 umí data do této schránky umístit a také je z ní
vyjmout.</P></FONT>
<FONT FACE="Verdana" COLOR="blue"><H2>Nadpis2</H2></FONT>
<FONT FACE="Arial" SIZE="2"><P>Podobně lze přenášet data i mezi
jednotlivými dokumenty stejného programu; v tomto případě se
navíc nikdy neztrácí kvalita dat (uspořádání, atributy,
"inteligence"). </P></FONT>
<FONT FACE="Verdana" COLOR="blue"><H2>Nadpis3</H2></FONT>
<FONT FACE="Arial" SIZE="2"><P>Stejně tak lze přenášet data mezi
různými místy jednoho dokumentu, dokonce i celé dokumenty a
složky: způsob je neobyčejně jednoduchý a také rychlý.
</P></FONT>
<FONT FACE="Verdana" COLOR="blue"><H2>Nadpis4</H2></FONT>
<FONT FACE="Arial" SIZE="2"><P>Mezi programy lze vytvářet pomocí
přenesených dat tzv. propojení. Tato propojení umožňují spouštět
poklepáním na umístěná data (umístěný objekt) původní aplikaci;
pokud navíc dojde ke změně původních dat, je tato změna reflek-
tována tam, kde jsou pomocí propojení umístěna.</P></FONT>
```




Docela pracné formátování v HTML sice dá požadovaný efekt, ale není to stále ono.

Vidíte, že zdrojový text za použití klasických metod formátování v HTML je zdrojový text velice nepřehledný a zbytečně dlouhý. Sice poskytne požadovaný efekt, představte si však, že podobných prvků bude na stránce více. Zdrojový kód stránky bude dlouhý a bude se dlouho stahovat na klientský počítač. A přitom to lze vyřešit tak elegantně a jednoduše, a sice za použití kaskádových stylů, CSS:

Do sekce HEAD se umístí tato definice dvou použitých stylů:

```
<STYLE TYPE="text/css">
<!-
H2 {font-family: Verdana;
    font-size: 16pt;
    color: Blue;}

P {font-family: Arial;
     font-size: 10pt;
     color: Black;}
->
</STYLE>
```

Vlastní zdrojový kód vypisovaného textu pak bude vypadat takto:

```
<H2>Nadpis1</H2>
<P>Lze přenášet data mezi jednotlivými programy, a to bez toho,
že by bylo nutné je ukládat do souborů a tyto soubory zase
```

```
otevírat. Data cestují mezi aplikacemi přes tzv. schránku; každá
slušná aplikace ve Windows 98 umí data do této schránky umístit
a také je z ní vyjmout.</P>
<H2>Nadpis2</H2>
<P>Podobně lze přenášet data i mezi jednotlivými dokumenty ste-
jného programu; v tomto případě se navíc nikdy neztrácí kvalita
dat (uspořádání, atributy, "inteligence").</P>
<H2>Nadpis3</H2>
<P>Stejně tak lze přenášet data mezi různými místy jednoho doku-
mentu, dokonce i celé dokumenty a složky: způsob je neobyčejně
jednoduchý a také rychlý. </P>
<H2>Nadpis4</H2></FONT>
<P>Mezi programy lze vytvářet pomocí přenesených dat tzv. propo-
jení. Tato propojení umožňují spouštět poklepáním na umístěná
data (umístěný objekt) původní aplikaci; pokud navíc dojde ke
změně původních dat, je tato změna reflektována tam, kde jsou
pomocí propojení umístěna.</P>
```

Stránka bude v prohlížeči vypadat stejně jako v případě formátování každého odstavce zvlášť.

Co se vlastně stalo? V sekci HEAD jsem předefinoval existující styly, *H2* a *P*, které má již HTML standardně definovány. Těm jsem pomocí parametrů stylů přiřadil požadované vlastnosti. Pak už stačilo text, který se má zobrazit, těmito styly zformátovat. Záměrně jsem použil pouze přidání vlastností stylům již existujícím, aby byl příklad jasnější. Samozřejmě lze definovat i styly nové, ale o tom později.

Co vlastně styly, v HTML zvané CSS, kaskádové styly, jsou? Styly nabízí tvůrcům stránek daleko širší formátovací možnosti, a to nejen textu, o jakých se ještě před několika lety nikomu ani nesnilo. Původní koncepce značek a jejich parametrů je totálně nabourána a postavena na hlavu. Stejně jako skripty i definice stylů je umístěna na začátku stránky a jednotlivé značky se na ně v průběhu zobrazení a práce se stránkou odkazují. Vlastně i definice stylů je více podobná programu v JavaScriptu než vlastnímu HTML; ale o tom později. Díky stylům tak získáváme možnosti, jaké známe třeba z již zmíněného Wordu nebo oblasti DTP. Otrocké opakování formátovacích značek, tak jak tomu bylo u prvního příkladu, je prostě minulostí. Kdo jde s dobou, používá CSS.

**POZNÁMKA:** *Webové styly se označují v anglickém originále jako kaskádové (cascade style sheets, CSS). To znamená, že jediný text může ovlivňovat několik stylů, které jej formátují. Nenastává totiž chyba, která by vyplývala z konfliktu dvou vlastností, které mají být ovlivněny dvěma různými styly, protože každý styl má jinou prioritu. U konfliktních stylů, které následují v definici stránky, má přednost ta definice, která byla uvedena jako poslední.*

## 3.1 Jak nadefinovat styl v dokumentu

Pokud si pamatujete na příklad programu v JavaScriptu, určitě zjistíte, že definice stylu v HTML má velice podobnou syntaxi. Je také umístěna v sekci HEAD, je také v poznámce, je

uzavřena do složených závorek jako funkce v JavaScriptu a jednotlivé položky definice jsou odděleny středníky. Dejme tomu, že chcete změnit vlastnosti stylu *H2*, tak jak bylo uvedeno už v příkladu využití stylu. Do sekce HEAD umístíte tuto strukturu:

```
<STYLE TYPE="text/css">
<!--
H2 {font-family: Verdana;
    font-size: 16pt;
    font-weight: bold;
    color: blue;}
-->
</STYLE>
```

Myslím, že je docela jasné, co tato definice udělá: existujícímu stylu *H2* (tedy nadpisu druhé úrovně) přiřadí modrou barvu, velikost 16 bodů, písmo Verdana a celý nadpis bude vypsán tučně. Obecně má definice stylu tato pravidla:

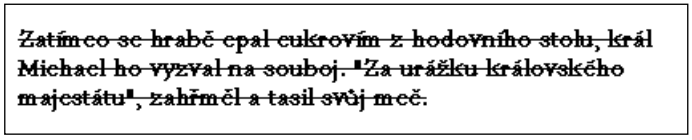
- Definice stylu je umístěna v hlavičce zdrojového kódu HTML, tedy sekci HEAD.
- Stejně jako je ohraničen skript značkami **<SCRIPT></SCRIPT>**, je definice stylu ohraničena značkami **<STYLE></STYLE>**.
- Navíc je celá definice stylu uzavřena do komentáře. Proč? Už u začlenění skriptu do stránky jsem poznamenal, že komentáře jsou nutné proto, že některé prohlížeče ještě dnes neumí dekódovat a používat styly. Komentáře samozřejmě nejsou nutné u moderních prohlížečů, jako jsou Internet Explorer 5.0 nebo Netscape Navigator 4.7. Ale i tak je dobré komentáře používat, svědčí to o určité kultuře autora stránky.

Tímto jednoduchým příkladem jsme tedy předefinovali, resp. upravili styl *H2*. Pokud tedy kdekoliv ve stránce použijete značku **<H2></H2>**, bude nadpis formátován podle definice stylu. Lze však vytvořit i nový styl, tedy takový, který definicí nebude opraven, ale přímo vytvořen. Tehdy se před jméno tohoto stylu umístí tečka:

```
<STYLE TYPE="text/css">
<!--
.novy {font-family: Gatineau;
       font-size: 12pt;
       font-weight: bold;
       text-decoration: line-through}
-->
</STYLE>
```

Pokud je definován styl jinak než úpravou stylu existujícího, nestačí už jen použít značku pro daný styl. Musí se dát značce, která pracuje s textem, jenž chceme podle stylu naformátovat, vědět, že má daný styl použít. Dejme tomu, že chceme na text použít styl *novy*. Na daný text jej můžeme aplikovat třeba takto:

```
<P CLASS="novy">Zatímco se hrabě cpal cukrovím z hodovního
stolu, král Michael ho vyzval na souboj. "Za urážku královského
majestátu," zahřměl a tasil svůj meč.</P>
```




Text formátovaný stylem *novy*

Zde vidíte příklad užití kaskádových stylů. Protože první formátovací styl, *P*, byl definován dříve, velikost písma, jeho font a řez bude přebit stylem *novy*, který byl definován později. Z příkladu je také patrné, že užít nadefinovaný styl lze kdekoliv. Není problém jej třeba umístit do značky **H2** definující nadpis:

```
<H2 CLASS="novy">Nadpis</H2>
```

nebo do značky **DIV**:

```
<DIV CLASS="novy">Text</DIV>
```

Není však vždy žádoucí používat nadefinovaný styl zároveň se stylem existujícím, jak tomu bylo doposud v uvedených příkladech. Pro tento případ má HTML značku **SPAN**:

```
<SPAN CLASS="novy">Zatímco se hrabě cpal cukrovím z hodovního
stolu, král Michael ho vyzval na souboj. "Za urážku královského
majestátu," zahřměl a tasil svůj meč.</SPAN>
```

Na text uvedený uvnitř značky **<SPAN></SPAN>** bude použit pouze styl *novy* a žádný jiný (**P**, **DIV** ani **H2**). Sama značka nemá žádný význam, nijak text neformátuje ani na něj nemá žádný vliv.

Pokud jde o parametr **CLASS**, ten definuje tzv. třídu. Třídy velice usnadňují práci, jak je patrné z těchto příkladů. Vlastnosti stylu se nadefinují v sekci HEAD a pak už je jen uveden odkaz na jméno stylu právě pomocí parametru **CLASS**. Obecné použití je tedy takovéto:

Nejprve se styl nadefinuje:

```
<STYLE TYPE="text/css">
.cerveny {font-size: 12pt;
          color=red}
</STYLE>
```

Použití ve stránce je pak následující:

```
<P CLASS="cerveny">Červený text</P>
<P>a toto je normální text</P>
```

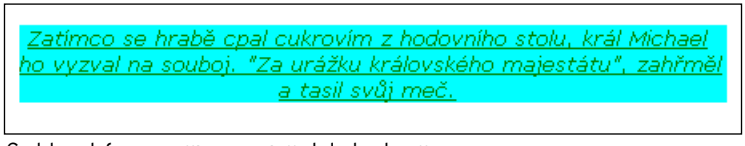
Základní užití stylu tedy znáte. Obecná syntaxe definice stylu tedy je:

```
<STYLE TYPE="text/css">
<!-
.jméno1 {parametr: hodnota;
         parametr1; hodnota1;
         parametr2; hodnota2;}

.jmeno2 {parametr: hodnota;
         parametr1: hodnota1;
         parametr2; hodnota2;}
apod.
-->
</STYLE>
```

Definice stylu je tedy započata značkou **STYLE** s parametrem **TYPE="text/css"**, za kterým následuje značka pro komentář, a pak vlastní definování stylů (úprava stylu bez tečky, nový styl s tečkou). Parametry každého stylu jsou uvedeny ve složených závorkách, každý má dvě části: jméno parametru a hodnota parametru, ukončeno středníkem. Jak jste si jistě všimli u předchozích konkrétních příkladů, není nutné řetězcové hodnoty uvádět v uvozovkách. Mezi jednotlivými parametry dokonce ani nemusejí být volné řádky, vše je odděleno středníky (stejně jako příkazy JavaScriptu). Definice stylu je ukončena uzavírající složenou závorkou, následuje uzavření komentáře a ukončení definice **</STYLE>**.

Styly mohou být definovány také přímo v textu, tj. přímo v sekci BODY může být bez jakékoli definice v sekci HEAD uvedeno:

```
<P STYLE="font-size: 10pt; font-family: Verdana; font-style:
italic; text-decoration: underline; text-align: center; color:
green; background: aqua"> Zatímco se hrabě cpal cukrovím z
hodovního stolu, král Michael ho vyzval na souboj. "Za urážku
královského majestátu," zahřměl a tasil svůj meč.</P>
```




Styl lze definovat přímo v místě, kde bude užit

Tady už nejde o žádné předdefinování stylu, ale přímo o definici stylu na místě, kde bude použit. Různých parametrů zde může být uvedeno daleko více; jistě uznáte, že provést něco

takového pomocí značek a parametrů HTML by bylo velice pracné a hlavně hodně nepřehledné.

**POZNÁMKA:** *Komentáře u CSS mají stejnou syntaxi jako v programu JavaScriptu, a sice* **`/* komentář */`**. *Příklad:*

```
.styl { font-size: 10pt } /* definice velikosti v bodech */
```

# 3.2 Práce s písmem a jeho vlastnostmi

V předchozí kapitole jste se již setkali s některými parametry definice stylu. V této kapitole nás budou především zajímat ty, které definují formát písma. mezi základní vlastnosti písma patří: font, řez a velikost.

## 3.2.1 Font písma

Font písma se definuje parametrem

```
font-family: písmo;
```

Parametr písmo může mít tyto hodnoty:

```
písmo1, písmo2, … , generické písmo
```

Za dvojtečku se umístí seznam písem, které má prohlížeč použít. Je dobré sem umísťovat fonty veskrze standardní, protože ne všichni uživatelé mají všechna písma nainstalována. Prohlížeč tedy prochází od počátku tímto seznamem písem, bere jedno po druhém a zjišťuje, zda v konkrétním počítači existuje. Pokud ano, zastaví se a nastaví dané písmo, pokud ne, jde dál. To šlo o písma konkrétní, reprezentovaná svým vlastním názvem a přesně typograficky popsaná, např.:

```
Arial, Gatineau, Times New Roman, Verdana
```

Existuje však ještě druh písma, tzv. generický, který označuje širší rodinu písem. To pro případ, že by v systému nebylo nalezeno ani jedno z konkrétních písem. Nenajde-li v systému tedy ani jedno z těchto písem, použije písmo generické, pokud je definováno. Bude-li tedy definice uvedena takto:

```
font-family: Arial, Verdana, sans-serif;
```

a nenajde-li prohlížeč ani písmo Arial, ani Verdana, pak použije základní bezpatkové písmo (sans-serif). Generických písem je definováno celkem pět a pokrývají prakticky celou oblast možných písem konkrétních:

- *serif* – standardní patkové písmo;
- *sans-serif* – standardní bezpatkové písmo;
- *cursive* – zdobné písmo v kurzivním řezu;
- *fantasy* – celkem libovolné ornamentální písmo;
- *monospace* – neproporcionální písmo.

## 3.2.2 Řez písma

Pro řez písma mají kaskádové styly určeny dva parametry. Pro kurzivu nebo sklonění písma je to

```
font-style: řez;
```

který umožňuje nastavit tyto hodnoty:

- *normal* – Klasické, stojaté písmo (toto nastavení implicitní, proto se nepoužívá).
- *italic* – Klasická kurziva vytvořená autorem písma. Např. pro písmo Times New Roman je to *Times New Roman*.
- *oblique* – Umělé sklonění písma, není totéž jako kurzíva. Aktuální font se pouze skloní, není to jiné písmo. Používá se tehdy, není-li k dispozici font definující kurzivu (většinou ale má kurzíva a sklonění stejný efekt).

```
<P STYLE="font-family: Times New Roman; font-style: normal">Times
New Roman psán normálně</P>
<P STYLE="font-family: Times New Roman; font-style: italic">
Times New Roman psán kurzívou</P>
<P STYLE="font-family: Times New Roman; font-style: oblique">
Times New Roman psán skloněným písmem</P>
```

Times New Roman psán normálně

*Times New Roman psán kurzívou*

*Times New Roman psán skloněným písmem*

U písma Times New Roman má *oblique* i *italic* stejný efekt.

Pro druhý řez písma, tučnost, má CSS parametr

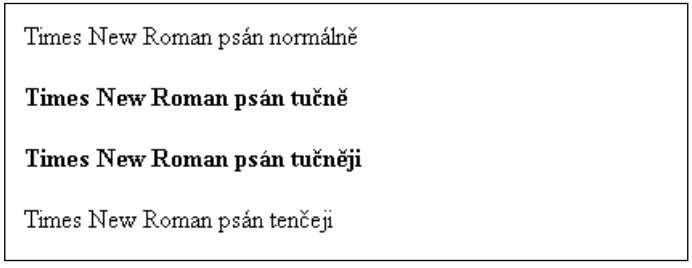
```
font-weight: řez;
```

který má tyto hodnoty:

- *normal* – Normální písmo, implicitní nastavení.
- *bold* – Tučný řez písma, je-li k dispozici od autora písma.

- *bolder* – Tučnější řez písma, tento už k dispozici u některých písem nebývá, takže je toto nastavení stejné jako bold.
- *lighter* – Naopak tenčí písmo než normální.

```
<P STYLE="font-family: Times New Roman; font-weight:
normal">Times New Roman psán normálně</P>
<P STYLE="font-family: Times New Roman; font-weight: bold"> Times
New Roman psán tučně</P>
<P STYLE="font-family: Times New Roman; font-weight: bolder">
Times New Roman psán tučněji</P>
<P STYLE="font-family: Times New Roman; font-weight: lighter">
Times New Roman psán tenčeji</P>
```




U písma Times New Roman řezy *lighter* a *bolder* chybí.

Nejčastěji vystačíme pouze s hodnotami *normal* a *bold*; několik často se vyskytujících rodin písem má i varianty *bolder* a *lighter*.

## 3.2.3 Velikost písma

Poslední definicí vlastního písma je jeho velikost. Zatímco dosud se velikost písma v HTML definovala sedmi různými velikostmi (jedna nejmenší písmo, sedm písmo největší), ve stylech se nejčastěji používá velikost v bodech (pt) a v pixelech (px). Velikost písma lze definovat i v klasických velikostech HTML, ale nepředbíhejme.

Parametr určující velikost písma je **font-size** a má tuto obecnou syntaxi:

```
font-size: velikost;
```

Velikost lze definovat několika způsoby: relativně, absolutně, v typografických jednotkách a procentuálně:

- *absolutní velikost* – Určuje velikost tak, jak je to obvyklé v HTML, od 1 do 7. Jednotlivé hodnoty jsou: *xx-small*, *x-small*, *small*, *medium*, *large*, *x-large*, *xx-large*. Hodnota *xx-small* tedy odpovídá velikost 1, *medium* velikosti 4 a *xx-large* velikosti 7.
- *relativní velikost* – Používá se tehdy, je-li styl textu odvozen z nějakého již existujícího stylu, takže pokud měníte styl *H2*, můžete nastavit velikost písma větší nebo,

menší než je ta aktuální: *larger* a *smaller*. *Larger* definuje tu vyšší hodnotu, *smaller* naopak tu menší.

- *velikost se zadanými jednotkami* – Zadává výšku písma v jednotkách, nejčastěji v bodech (pt) nebo pixelech (px).
- *procentuální velikost* – Určuje zvětšení či zmenšení oproti základní velikosti odstavce v procentech.

Pro lepší pochopení raději několik příkladů:

```
DIV {font-size: 14pt;}
   /* písmo bude mít velikost 14 bodů */
H2 {font-size: larger;}
   /* písmo nadpisu bude větší o jednotku */
P {font-size: 140%;}
   /* písmo bude o 140 % větší než původní */
CODE {font-size: 13px;}
   /* velikost písma je dána přesně 13 obrazových pixelů */
```

Jaké efekty mají jednotlivá zadání velikostí je ovšem nejlépe vyzkoušet, slovy se nedá bohužel popsat vše.

Určitě jste si všimli různých použitých definic délkových údajů: jednou body, jednou pixely, jednou procenta. Proto uvedu všechna tato zadání délkových jednotek pohromadě, abyste se mohli lépe orientovat.

Hodnota definující délku se skládá ze znaménka (plus se nemusí uvádět), za nímž bezprostředně následuje vlastní číslo a za ním ihned bez mezery označení jednotky. Jednotky mají zásadně zkratku ze dvou písmen, a jsou buď relativní (vzhledem k objektu, z něhož vychází), nebo absolutní:

- *px* – výška zadaná v pixelech - pixel sám je však bezrozměrný a jeho aktuální „fyzická“ velikost závisí na parametrech zobrazení a výsledně i tisku;
- *in* – palce (1" = 2,54 cm);
- *cm* – centimetry;
- *mm* – milimetry;
- *pt* – typografické body, 1 pt = 1/72 palce;
- *pc* – picas, 1pc = 12pt.

## 3.3 Barvy textu a pozadí

Styly umožňují také daleko lépe pracovat s barvami, neomezují se pouze na definici barvy textu. Lze nastavit zvlášť jak barvu textu, tak pozadí textu; do této skupiny parametrů patří také nastavení obrázku na pozadí, jeho umístění, opakování, rolování apod.

## 3.3.1 Barva textu

Barva textu se nastavuje podobně jako značkou **FONT**, a sice parametrem **color**:

```
color: barva;
```

Tato barva se nastavuje standardním způsobem, buďto jménem základní barvy nebo trojicí barev RGB:

```
color: blue;
color: RGB(64, 64, 128);
```

Definice barev pomocí RGB se liší od klasického HTML, kde se barva zadává hexadecimálně: #rrggbb. Tato definice je naštěstí srozumitelnější a umožňuje definovat barvu v normální desítkové soustavě. Lze také zadávat barvy procentuálně:

```
color: rgb(100%, 50%, 0%)
```

kde procenta reprezentují procentní hodnoty jednotlivých barevných složek, jsou přepočítána na nejbližší celočíselné hodnoty v rozsahu 0–255 pro každou barvu. Pro uvedený příklad je analogické toto zadání:

```
color: rgb(255, 128, 0)
```

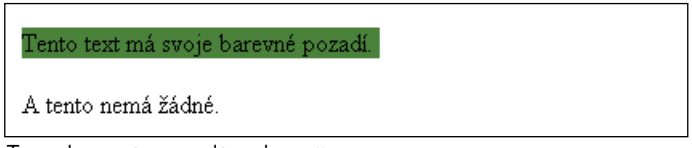
## 3.3.2 Barva pozadí textu

Text může mít i barvu svého pozadí. Je vlastně barva jakéhosi rámečku okolo textu, který je normálně „neviditelný“. Barva pozadí se nastavuje parametrem **background-color**:

```
background-color: barva;
```

Barva se zadává úplně stejně jako u textu, navíc lze zadat i průhledné pozadí, a sice hodnotou **transparent**.

```
<P STYLE="background-color: green">Tento text má svoje barevné
pozadí.</P>
<P>A tento nemá žádné.</P>
```




Text s barevným pozadím a bez něj

Na pozadí textu se dá umístit i obrázek, nejenom jednolitá barva. To se zařídí parametrem:

```
background-image: url;
```

Hodnota *url* je cestou k obrázku, který chcete na pozadí textu zobrazit. Pro práci s obrázkem na pozadí mají styly několik dalších parametrů, které upřesňují práci s tímto obrázkem. Lze jimi dosáhnout velice zajímavých efektů, ale jejich ladění není vůbec úměrné výsledku.

# 3.4 Formátování textu

Konečně se dostáváme k nejzajímavější části kapitoly o stylech: formátování textu. Styly nabízejí velice bohaté možnosti, které vám standardní HTML neumožní. Velice se blíží nastavení stylů v profesionálních textových editorech. Můžete nastavit mezery mezi písmeny, mezery mezi slovy, libovolné řádkování, zarovnání textu, podtrhávání, přeškrtnutí apod.

## 3.4.1 Mezi slovy a písmeny

Mezi písmeny i mezi slovy lze nastavit vlastní velikost mezer. Ty lze dát jak od sebe, tak více k sobě.

Mezery mezi písmeny se nastavují parametrem **letter-spacing**:

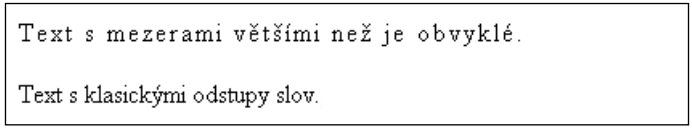
```
letter-spacing: délka;
```

Velikost mezery se nastavuje v již zmíněných jednotkách, většinou se však zadává v tzv. čtverčíku, tedy šířky písmena m:

```
letter-spacing: 1.5m;
```

Tuhle velikost si na rozdíl od velikosti v bodech nebo pixelech dokáže každý představit, a proto je její použití pravděpodobně nejlepší. Lze zadat i hodnotu normal, která nastaví standardní velikost mezery, která je definována implicitně.

```
<P STYLE="letter-spacing: 2m;">Text s mezerami většími, než je
obvyklé.</P>
<P>Text s klasickými odstupy slov.</P>
```

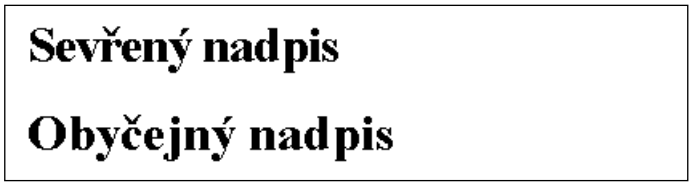



Mezera mezi písmeny může být jakkoliv velká.

Současně s mezerami mezi písmeny se zvětšují i mezery mezi slovy; i mezera je totiž znak. Lze zadat dokonce i záporné hodnoty, a tím například u nadpisů dosáhnout zajímavého

sevřeného formátování, které je obvyklé v kvalitních programech DTP.

```
<H1 STYLE="letter-spacing: -1pt">Sevřený nadpis</H1>
<H1>Obyčejný nadpis</H1>
```




Mezera mezi slovy může být i záporná; tehdy způsobí sevření slova.

## 3.4.2 Mezery mezi řádky, odřádkování

Mezery lze nastavit i mezi jednotlivými řádky, můžeme tedy nastavit řádkování. Tuto meziřádkovou mezeru (meziřádkový proklad) určuje parametr **line-height**:

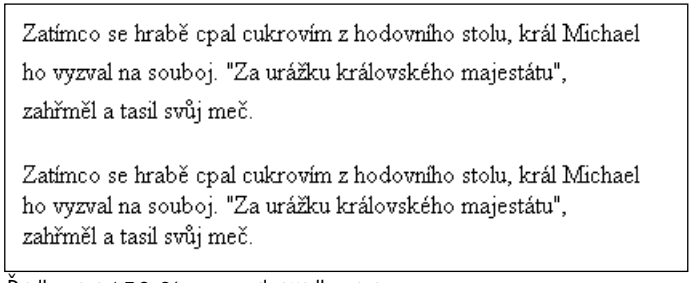
```
line-height: výška;
```

Vzdálenost mezi řádky je vlastně výška řádku, tj. pokud je mezera řádky 10 a výška písma 90, je výška řádku 100. Jako hodnota tohoto parametru se tedy udává výška řádku, a ne mezera mezi řádky.

Tato výška řádku se zadává buďto jako relativní, ta definuje odchylku od výšky standardní (ta bývá 110 – 120 % výšky písma), nebo absolutně. Relativní zadání je buďto v násobku, tj. 1.5 znamená 1,5x výška písma, nebo procentem (150 % je totéž jako 1.5). Absolutní zadání je totožné jako zadávání jakékoliv velikosti (pt, px apod.), je to však poněkud problematické vzhledem k různé velikosti písma.

Pokud tedy zadáte **line-height: 150%**, zvětšíte výšku řádku o 50 %:

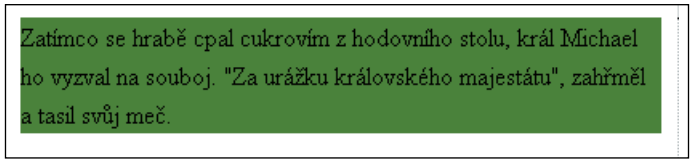
```
<P STYLE="line-height: 150%">Zatímco se hrabě cpal cukrovím z
hodovního stolu, král Michael ho vyzval na souboj. "Za urážku
královského majestátu," zahřměl a tasil svůj meč.</P>
```




Řádkování 150 % a normální řádkování.

Pokud zadáte textu barvené pozadí, uvidíte, že obdélníček, kterému se tato barva nastavuje, se zvětší podle velikosti odřádkování:

```
<P STYLE="line-height: 150%; background-color: green">
```




Barevné pozadí se přizpůsobí i větším řádkům

## 3.4.3 Další parametry pro formátování

Další řezy písma (podtržení, přeškrtnutní) se definují parametrem **text-decoration**:

```
text-decoration: řez;
```

Parametry jsou následující:

- *none* – zruší všechny řezy;
- *underline* – daný text podtrhne;
- *overline* – daný text nadtrhne;
- *line-through* – daný text přeškrtne.


Různé typy řezu (dekorace) textu

Ve stylu lze použít i tzv. verzálky, kapitálky a minusky (velká a malá písmena). Ty se zadávají parametrem názvem **text-transform**:

```
text-transform: nastavení písma;
```

Parametry mají na text tento vliv:

- *capitalize* – převede text na kapitálky (druh velkých písmen);
- *uppercase* – převede text na verzálky (velká písmena);

- *lowercase* – převede text na minusky (malá písmena),
- *none* – ruší tyto efekty.

Tyto efekty budou mít na text tento vliv:

TEXT PSANÝ KAPITÁLKAMI.
TENTO TEXT JE PSÁN VERZÁLKAMI.
tento text je psán minuskami.

Pokud si vzpomínáte na parametr **ALIGN**, kterým se zarovnává text formátovaný značkou HTML, určitě vás napadne, že něco takového musí být i ve stylech. Zarovnání textu se v horizontální podobě provádí parametrem **text-align**:

```
text-align: zarovnání;
```

Jednotlivé hodnoty jsou myslím docela jasné:

- *left* – zarovnání vlevo;
- *right* – zarovnání vpravo;
- *center* – zarovnání na střed;
- *justify* – zarovnání do bloku.

Posledním parametrem bude odsazení textu od levého okraje, které nastavuje parametr **text-indent**:

```
text-indent: vzdálenost;
```

Toto odsazení platí pro první řádek a tedy je vhodné pro tvoření odstavců. A to klasicky: buď v délkových jednotkách (body, centimetry, pixely) nebo v procentech celé šířky řádku.

Zatímco se hrabě cpal cukrovím z hodovního stolu, král Michael ho vyzval na souboj. "Za urážku královského majestátu", zahřměl a tasil svůj meč.

Odsazení prvního řádku nového odstavce

# 3.5 Důležité vlastnosti stylů

Styly mají daleko rozsáhlejší možnosti. jak už jsem však naznačil, stylům se v této knize věnuji zejména kvůli jejich využití v dynamickém HTML, které je tématem poslední části této knihy, a ne kvůli přesné definic všech vlastností. Je však ještě několik věcí, které pro lepší pochopení vlastností a užití stylů musím uvést.

## 3.5.1 Seskupování stylů

Seskupování definicí stylů je vlastnost, která umožní jedinou definicí parametru přiřadit více stylům naráz, viz příklad:

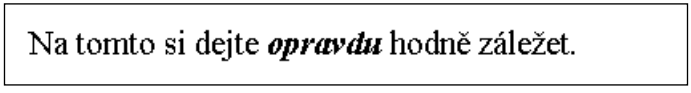
```
Styl1, Styl2, Styl3 {font-family: Arial, sans-serif};
```

Všechny tyto definované styly, kromě jiných svých vlastností, budou zobrazovány písmem Arial nebo jiným bezpatkovým písmem, nebude-li písmo Arial nalezeno. To se dá využít různými způsoby a lze si tak ušetřit hodně práce při definici samotných stylů.

## 3.5.2 Dědičnost stylů

Dědičnost definicí vyplývá již z dědičnosti platné v samotném HTML. Vnořený styl si zachovává veškeré formátovací vlastnosti stylu, do něhož je vnořen – přidává pouze to, čím se od něj odlišuje nebo co definice nadřazeného stylu neobsahuje. Například:

```
<P STYLE="font-size: 16pt">Na tomto si
dejte<STRONG><I>opravdu</I></STRONG> hodně záležet.</P>
```




Do stylu bylo přidáno zesílení a sklonění slova.

Ve značce **P** je definováno jasně, že písmo musí mít velikost 16 bodů. Není však vyloučena kurziva a silné písmo, takže je možné jeho část formátovat tak, jako je to patrné v tomto příkladu.

## 3.5.3 Priorita použití stylů

Styly mají i svoje priority, tzn. pokud použijete na jeden text více stylů, z nichž každý definuje jinou velikost, je předem určeno, jakou výslednou velikost bude text mít. Stejně jako v HTML jsou priority značek jasné: **BODY** je nadřazený všem prvkům, které obsahuje. Pokud pak ve stylech předefinujete značku **BODY**, změníte definice všech prvků, které patří do sekce BODY. Definice:

```
BODY {color: blue}
```

způsobí, že veškeré písmo ve stránce bude vykresleno modrou barvou, pokud nebude ve značkách zdrojového kódu (**P**, **H2** atd.) uvedena barva jiná nebo pokud nebude barva implicitně zadána v formátování HTML.

Nebo zadání:

```
P {line-height: 120 %}
```

znamená, že u všech textů, které budou formátovány značkou **P** nebo styly, které byly vytvořeny ze stylu *P*, bude výška řádku nastavena na 120 %.

## 3.5.4 Změna určité vlastnosti stylu

Styly mohou mít dokonce i něco jako „podstyly“, kdy jeden styl má pro každou událost jinak definovaný formát. Například značka **A**, která definuje odkaz, má definován styl pro aktivní odkaz, pro odkaz, který již byl navštíven, nebo pro odkaz, nad nímž je umístěn kurzor. Změnou vlastností těchto „podstylů“ lze docílit docela zajímavých efektů. Dejme tomu, že chcete, aby odkaz nebyl podtržený, byl vypsán žlutou barvou, tučným fontem Arial o velikosti 12 pixelů. Dosáhnete toho touto definicí stylu:

```
a:link {font-family : Arial;
        font-size : 13px;
        color : yellow;
        text-decoration : none;}
```

Pokud chcete, aby styl změnil barvu nebo třeba velikost písma, když jej přejedete myší (událost **onmouseover**), na červenou, zadejte:

```
a:hover {color : red;}
```

A takto bych mohl pokračovat. Styly lze dosáhnout opravdu zajímavých efektů, a to včetně přesného umístění objektu na obrazovce. Tematika kaskádových stylů je totiž daleko širší, než tu bylo prezentováno. Cílem této kapitoly nebylo vytvořit referenční příručku užití stylů, ale na jednoduchých příkladech ukázat základní možnosti. Tyto znalosti pak mnohokrát zúročíte při tvorbě dynamických stránek, které kombinují styly, objekty a jejich metody, události a skripty. Pokud jste tedy zvědaví na tvorbu programů v JavaScriptu, otočte na další, třetí část.

# Část třetí:
# Úvod do Javascriptu

**Proměnné a práce s nimi**

**Věstavěné objekty JavaScriptu**

**Základní příkazy JavaScriptu**

**Funkce a jejich využití**

**První programy v JavaScriptu**

V této části, která pokrývá základní problematiku programování v JavaScriptu, se naučíte tvořit jednoduché programy, zatím v podstatě bezúčelné. Půjde zatím spíše o teorii nežli o praxi; tato část je nutným zlem, které musíte zvládnou, abyste mohli sami vytvářet webové aplikace. K tomu jsou nutné další znalosti, které budou obsahem čtvrté a poslední části této knihy: provázání skriptů s webem a prohlížečem, s jejich objekty a událostmi, dynamické HTML.

Tato část však nepokrývá celou problematiku JavaScriptu. Když jsem psal tuto knihu, měl jsem při ruce osmisetstránkovou referenční příručku, a to si ještě myslím, že v ní nebylo úplně všechno; tato část si klade za cíl naučit čtenáře ten nutný základ, aby nemusel při vytváření stránek tápat, a v takovýchto tlustých knihách jen hledal přesné syntaxe příkazů, názvy objektů a událostí, s nimiž pracuje. Měl by prostě umět programovat v JavaScriptu, aniž by znal veškeré jeho podrobnosti. Při programování pak může mít v ruce stejnou příručku, jako jsem měl já, kde si tyto podrobnosti a přesné syntaxe sám vyhledá.

# 1. Krátce na úvod

Začnu docela jednoduchým programem, který vygeneruje šest náhodných čísel do loterie. Z něho je patrné, jakou základní strukturu program v JavaScriptu má a jak je integrován do těla zdrojového kódu HTML:

```
<HTML>
<HEAD>
  <TITLE>Loterie</TITLE>
</HEAD>

<BODY>
<H2>Náhodná čísla loterie:</H2>
<SCRIPT LANGUAGE="JavaScript">
<!--
numb = new Array(6);

var Different;
for (var i = 0; i < 6; i++)
  {
  Different = false;
  while (Different == false) {
    Different = true;
    numb[i] = Math.floor(Math.random() * 49) + 1;

    for (j = 0; j < 6; j++) {
      if ((numb[i] == numb[j]) && (i != j)) {
        Different = false;
      }
```

```
    }
  }
}
document.write(numb[0] + " " + numb[1] + " " + numb[2]
+ " " + numb[3] + " " + " " + numb[4] + " "
+ numb[5])
// -->
</SCRIPT>
</BODY>
</HTML>
```

Jedinou funkcí tohoto krátkého programu je vypsat na webovou stránku šest čísel, která jsou generována pomocí matematického objektu **Math.random** (co jsou objekty, se dozvíte dále). Výsledná stránka, která se zobrazí v prohlížeči, bude podobná této:

**Náhodná čísla loterie:**

19 8 46 38 9 34

Po každém načtení do prohlížeče vygeneruje loterijní program šest čísel.

Běh programu probíhá zhruba podle tohoto schématu:

1) Je vytvořeno pole šest čísel pro šest sázek loterie.
2) Vygeneruje se náhodné číslo a to se uloží do pole sázek.
3) Generování se opakuje celkem šestkrát; pokud je nějaké číslo stejné jako už existující, generuje se znovu.
4) Jakmile je určeno všech šest sázek, jsou vypsány na webovou stránku.

Přepis takovýchto jednoduchých programů do zdrojového kódu JavaScriptu je možný už se základními znalostmi programování a několika příkazů a funkcí. Všechny tyto příkazy a funkce budu postupně popisovat, abyste po přečtení celé této části byli sami schopni napsat jakýkoliv jednodušší program v JavaScriptu. Takže, jdeme na to!

**POZNÁMKA:** *JavaScript je jazyk, který rozlišuje velká a malá písmena. To znamená, že klíčová slova jazyka, proměnné, názvy funkcí a všechny další identifikátory musí být vždy psány odpovídající velikostí písma. Například klíčové slovo* for *musí být psáno* „for“*, nikoli* „For“ *nebo* „FOR“*. Podobně* help, Help *a* HELP *jsou tři různé názvy proměnných.*

*Všimněte si však, že HTML velká a malá písmena nerozlišuje, což je vzhledem k blízké asociaci s JavaScriptem poněkud matoucí. Obzvlášť názvy událostí jsou často psány v HTML různou velikostí písma (například* onClick *nebo* OnClick*), ale pokud je na ně odkazováno z JavaScriptu, musí být všechna písmena malá (*onclick*).*

# 2. Proměnné, výrazy a práce s nimi

Základem každého programovacího jazyka jsou příkazy (klíčová slova), struktury, které řídí program a říkají, co má interpret (v našem případě prohlížeč) dělat, a proměnné, podle kterých se zase řídí příkazy a chování programu v rozhodovacích okamžicích. Lze zjednodušeně říci, že na začátku programu je nějaký vstup, který program přetransformuje na výstup. Tyto vstupy a výstupy se uchovávají právě v proměnných, a právě o proměnných bude tato kapitola.

## 2.1. Co jsou proměnné

Proměnné jsou určitá místa v paměti, která mají svoji hodnotu. Abyste si nemuseli pamatovat, kde jsou v paměti uloženy, mají proměnné definován svůj název, který je podstatně snáze pamatovatelný, usnadňuje a zpřehledňuje vlastní program. Výhodou proměnných je to, že nemají konstantní hodnoty, tj. jednou zadané a pak dále neměnné. Hodnotu proměnné můžete kdykoliv během běhu programu měnit, a to v závislosti na okolnostech. V danou chvíli je hodnotou proměnné konstanta; konstantou může být konkrétní číslo, konkrétní řetězec. Příklad takových konstant může být takovýto:

***Číselné konstanty:***
3,14
-1,414
0,333333333333333333
6,02e+23
1,4738223E-32

***Řetězcové konstanty:***
„P"
„3,14"
„Jak se máte?"

JavaScript nemá předem definován typ proměnných, proto se číselné a řetězcové konstanty odlišují uvozovkami. Řetězcové proměnné lze někdy uzavírat i do apostrofů; já budu preferovat klasické uvozovky.

### 2.1.1 Jak proměnná nabývá hodnoty

Hodnotu získává proměnná přiřazením nějaké konstanty, jiné proměnné nebo výsledku nějaké operace. Pokud chcete např. zařídit, aby proměnná *a* měla hodnotu 10, provedete to tímto přiřazením:

```
a = 10;
```

Následující řádek k proměnné *a* přidává 5 a výsledek přiřazuje nové proměnné *soucet*:

```
soucet = a + 5;
```

Jistě jste si všimli středníku na konci řádku. Tento středník odděluje jednotlivé příkazy a operace JavaScriptu. Pokud používáte pro každý takovýto příkaz nový řádek, není středník nutný. Pokud by však tento kousek zdrojové kódu JavaScriptu byl na jednom řádku, je středník nezbytný:

```
a = 10; soucet = a + 5;
```

Než začnu popisovat deklarace proměnných a jednotlivých datových typů, rád bych ještě vysvětlil alespoň stručně rozdíl mezi globální a lokální proměnnou. Globální proměnná je definovaná v celém rozsahu programu. Tj. platí ve všech funkcích a procedurách, stejně tak jako v hlavním těle programu. Může být pouze jedna stejného jména; kdykoliv proměnné s tímto názvem přiřadíte nějakou hodnotu, původní hodnotu ztratí. Nebude tak deklarována žádná nová proměnná se stejným jménem.

Lokální proměnná má platnost omezenu lokálně, pouze na jednu funkci, proceduru nebo podprogram. S lokální proměnnou budete pracovat pouze v případě, bude-li definována na úrovni funkce, nikoliv na úrovni programu. Tzn. proměnná existuje pouze tehdy, je-li prováděna funkce, jakmile je její provádění u konce, proměnná zanikne. Pokud definujete lokální proměnnou se stejným názvem jako nějakou proměnnou globální, bude se uvnitř funkce pracovat s tou lokální, venku (mimo funkci) pak s tou globální; obě budou mít jiné hodnoty.

## 2.2 Definice proměnných

Víte tedy, co jsou proměnné. Na začátku každého programu v JavaScriptu je dobrým zvykem tyto proměnné deklarovat, říci prohlížeči, že tyto a tyto proměnné budou používány a mají takové a takové počáteční hodnoty. Tato deklarace však není nutná, proměnná se totiž vytvoří v okamžiku jejího prvního použití.

Pro deklaraci proměnné má JavaScript klíčové slovo

```
var
```

Pokud zůstanu u výše uvedeného příkladu, bude vypadat deklarace takto:

```
var a;
var soucet;
```

Pomocí slova var lze deklarovat i více proměnných zaráz:

```
var a, soucet;
```

Slovem **var** lze dokonce kombinovat deklaraci proměnné s úvodním přiřazením nějaké hodnoty, konstanty:

```
var a = 10;
```

**POZNÁMKA:** *Jak už jsem napsal, není deklarace proměnné vždy zapotřebí. Okamžik, kdy poprvé použijete proměnnou, která není dosud deklarována, bude automaticky její deklarací. Jediný případ, kdy skutečně potřebujete deklarovat proměnnou pomocí slova* **var***, nastane, když deklarujete lokální proměnnou uvnitř definice funkce a název proměnné se používá též jako globální proměnná mimo tuto funkci. Pokud totiž použijete proměnnou ve funkci bez její deklarace, pak by JavaScript předpokládal, že jste měli na mysli globální proměnnou deklarovanou mimo tuto funkci, a nedeklaroval by automaticky v této funkci lokální proměnnou.*

*Toto se dá obejít tím, že uvnitř funkce použijete proměnnou s jiným názvem; tehdy není nutné* **var** *použít. Pro přehlednost je dobré* **var** *používat.*

## 2.3 JavaScript – proměnné bez určeného typu

Důležitým rozdílem mezi JavaScriptem a „velkými" programovacími jazyky typu Java, Pascal nebo C je to, že JavaScript nemá typové proměnné, tedy že proměnné mohou mít hodnotu jakéhokoli datového typu (řetězec, číslo, pole apod.). V JavaScriptu tak klidně můžete přiřadit proměnné číslo a později třeba řetězec:

```
a = 10;
a = "deset";
```

Díky tomu není nutné při deklaraci proměnné specifikovat její datový typ tak, jak je to nutné v C++, Pascalu nebo Javě. V těchto jazycích se musí proměnná deklarovat také datovým typem, který nelze v průběhu programu měnit:

```
int a; // deklarace celočíselné (integer) proměnné v C, C++ nebo
       // Javě
```

V JavaScriptu není potřeba žádný konkrétní datový typ, stačí pro deklaraci proměnné pouze jedno klíčové slovo, **var**:

```
var a; // deklarace proměnné jakéhokoliv typu v JavaScriptu
```

**POZNÁMKA:** *Dvě lomítka znamenají v JavaScriptu poznámku, stejně jako v C nebo C++. Jakýkoli text mezi // a koncem řádku je považován za poznámku a je JavaScriptem ignorován. Také jakýkoli text (který může zabírat více řádků) mezi znaky /* a */ je považován za poznámku (stejně jako u definice stylů, viz kapitola 3 druhé části).*

Dalším zajímavým rysem JavaScriptu je skutečnost, že se hodnoty proměnných automaticky konvertují z jedné na druhou; nejlépe to osvětlí příklad:

```
a = "Automobil ";
b = 10;
c = a + b;
```

Zatímco v C++ nebo Javě by toto nebylo možné a kompilátor by generoval chybu, v JavaScriptu se číslo, hodnota proměnné *b*, převede na řetězec a do proměnné *c* se uloží text „Automobil 10“. To je nespornou výhodou JavaScriptu a dělá to jazyk o něco jednodušším.

# 2.4 Základní typy hodnot proměnných

Proměnné mají několik základních a obecných datových typů, které jsou společné jak pro JavaScript, tak pro VBScript. I když většině uživatelů, kteří se již s programování srazili, budou následující řádky jasné, pro jistotu to stručně přelétnu. Pečlivěji byste se však měli zaměřit na odstavec o objektech; JavaScript je totiž objektově orientovaný jazyk a objekty jsou nutné pro pochopení jeho práce.

## 2.4.1 Čísla

Čísla jsou nejjednodušším datovým typem a myslím, že nevyžadují příliš mnoho vysvětlování. Čísla mohou být celá nebo s pohyblivou (desetinnou) čárkou. Na rozdíl od programovacích jazyků nedělá rozdíl mezi celým číslem a číslem s pohyblivou čárkou – to je dáno „netypovostí“ JavaScriptu. Všechna čísla jsou tak v JavaScriptu uložena jako hodnoty s pohyblivou čárkou (8 bajtů), a tak jsou i znázorněna – za použití standardního zápisu. Čísla tedy mohou i takto:

±5,5825423813355103*10308
±3,5245504755762468*10-308,11

Je jasné, že 8 bajtů např. pro číslo 10 je zbytečně moc; programy v JavaScriptu však naštěstí nemají tolik proměnných, aby to ucpalo paměť, takže to ani tolik nevadí a tvůrcům programů to usnadňuje práci.

## 2.4.2 Řetězce – písmena, slova, věty

Co je řetězec, je také jasné. Sada libovolných písmen, číslic a další znaků spojených v jednu řadu. Řetězce se v JavaScriptu ohraničují jednoduchými (apostrofy) nebo dvojitými uvozovkami. Stejně jako čísla i řetězce lze sčítat neboli slučovat:

```
hlaska = "Ahoj, " + "Petře"; // Obsahem proměnné hlaska bude
                             // "Ahoj, Petře"
```

nebo

```
jmeno = "Petr Broža";
pozdrav = "Jmenuji se" + " " + name;
```

Samozřejmě JavaScript neumí jenom slučovat; s řetězci lze provádět mnoho dalších operací: vyhledávat jednotlivé znaky, odstraňovat slova, převracet apod. To bude předmětem kapitoly 2.6, která se zabývá základní prací s řetězci.

### 2.4.3 Logické hodnoty – *pravda*, *nepravda*

Logické hodnoty (*boolean*) definují dva stavy: *pravda* a *nepravda*. Používají se k vyhodnocení porovnání hodnot proměnných, na základě vyhodnocení této pravdy nebo nepravdy (rovná se, nerovná se, je větší nebo měnší) se pak program dále větví. Pomocí této pravdy a nepravdy lze např. zjistit, zda hodnota proměnné a se rovná deseti:

```
a == 10
```

Pokud se opravdu proměnná a rovná 10, pak je výsledkem porovnání logická hodnota *pravda* neboli *true*. Nerovná-li se, je výsledkem *nepravda* neboli *false*.

**Poznámka:** *Při porovnání se používá operátor == (dvě rovnítka za sebou), nikoliv jedno. To má v JavaScriptu význam přiřazení. Tady často dochází k chybám!*

V praxi se takový příklad zapíše pomocí příkazu **if** a **else**:

```
if (a == 10) b = b + 1;
else a = a + 1;
```

Tento velice jednoduchý prográmek zjišťuje, zda se *a* rovná 10. Jestliže (if) ano, přičte 1 k proměnné *b*, jinak (else) přičte 1 k proměnné *a*.

### 2.4.4 Pole

*Pole* je velice specifickou proměnnou, či spíše souborem proměnných jakéhokoliv datového typu. To znamená, že je to např. deset čísel poskládaných za sebou. Každé z těchto čísel má svůj index, kterým se označuje, jméno je pro celé pole společné. Dejme tomu, že se toto pole jmenuje *jablka*, které definuje koš deseti jablek. Každé toto jablko je postupně očíslováno 0–9 (indexy začínají vždy nulou). Obecný zápis tohoto pole v JavaScriptu bude:

```
jablko[n]
```

Pokud budete chtít pracovat třeba s pátým jablkem, řeknete to JavaScriptu takto:

```
jablko[5]
```

Přitom hodnota proměnné jablko[n] může být jakéhokoliv datového typu. Poli se však v této knize příliš zabývat nebudu, jejich užití už záleží na hodně konkrétních případech, které nejsou obsahem této publikace.

### 2.4.5 *Null* – speciální datový typ

Někdy je při programování potřeba definovat proměnnou, která nemá žádnou hodnotu. Té se pak přiřazuje speciální hodnota, *null*. Není to totéž jako nula; nula je číslo a hodnota *null* není ani číslo, ani pole, ani řetězec ani objekt. I když za určitých okolností bude *null* na 0 konvertována, ale 0 není ekvivalentem *null*.

### 2.4.6 Objekty

*Objekt* je dalším specifickým datovým typem, souborem pojmenovaných dat. Trošku objekt připomíná pole, ale podobnost je čistě povrchní. JavaScript je objektově orientovaný jazyk a pracuje zejména s objekty; o pojmenovaných datech objektu hovoříme jako o vlastnostech objektu. Zatímco na prvek pole se odkazujeme indexem, na vlastnost objektu se odkazujeme tečkou, která je umístěna mezi objektem a vlastností. Máme-li tedy objekt **window**, tedy objekt okna prohlížeče, na jednotlivé vlastnosti se můžeme odkazovat takto:

```
window.document
window.history
```

Vlastnosti objektů se v mnoha ohledech chovají jako proměnné JavaScriptu a mohou obsahovat jakýkoli typ dat, včetně polí, funkcí a jiných objektů. Pokud je ve vlastnosti objektu uložena funkce, nazýváme ji metodou. Máme-li tak třeba objekt **document**, který má definovánu metodu **write**, která vypíše do okna prohlížeče zadaný text, zadáme do skriptu tento řádek:

```
document.write("Text napsaný metodou write.");
```

Podrobněji se jednotlivým objektům a metodám budu věnovat v dalších částech této a příští kapitoly.

## 2.5 Číselné, porovnávací a logické operátory

Základem každého programovacího jazyka je vyhodnocování výrazů. Výraz je tedy kousek zdrojového kódu, který JavaScript může vyhodnotit a získat tak nějakou hodnotu. Většinou jsou tyto hodnoty logického typu, tedy *boolean*, které mohou nabývat buďto hodnoty pravda nebo nepravda (viz předchozí kapitola).

Základní výrazy jsou jednoduché, pouhé konstanty. Hodnotou této konstanty je konstanta sama. Ovšem zde se nechci zabývat těmito jednoduchými výrazy; ty nejsou nijak zvlášť zají-

mavé. Zajímavější a daleko použitelnější jsou výrazy složitější, výrazy vytvořené kombinací těchto jednoduchých výrazů:

```
i + 2 // hodnotou výrazu je součet proměnné i a konstanty 2
(i + 2) - j // hodnotou výrazu je součet proměnné i a konstanty
            // 2, od kterého se odečte proměnná j
```

Pro vyjádření těchto výrazů se používá operátorů, zde jsou to *plus* a *minus*. JavaScript jich zná daleko víc a všechny jsou svým způsobem důležité: jednou pro zápis operací, ať už matematických nebo řetězcových, jiné zase pro vyhodnocení pravdivosti nebo nepravdivosti podmínek (využívá se zejména v příkazech **if**, **while** a **for**, viz dále).

Jednotlivé používané operátory popíšu, bez nich si totiž nedokážu další výklad představit; operátory jsou totiž základním prvkem všech programovacích jazyků.

**POZNÁMKA:** *Protože se operátory týkají především porovnávání proměnných, zařadil jsem kapitolu o operátorech do části o proměnných, abych nemusel tvořit samostatnou, velkou kapitolu. Podle mého názoru sem tato tématika přesně zapadá.*

## 2.5.1 Aritmetické (matematické) operátory

Co jsou aritmetické operátory, je myslím jasné. Definují základní matematické operace, na kterých jsou postaveny všechny další složitější matematické a i jiné funkce.

**POZNÁMKA:** *V následujícím textu je používáno slovo operand. To je výraz, s nímž operátor pracuje. V našem případě to jsou většinou čísla nebo řetězce, popř. objekty.*

***Přehled aritmetických operací:***

- *sčítání* (+) – Pokud jsou operandy čísla, budou sečteny; pokud oba řetězce, složí je dohromady. Pokud je jeden z operandů řetězec, pak je ten druhý konvertován také na řetězec a tyto dva řetězce jsou spojeny.
- *odečítání* (-) – Odčítá druhé číslo od prvního.
- *násobení* (*) – Násobí dvě čísla.
- *dělení* (/) – Dělí první číslo druhým číslem. Výsledkem je vždy číslo s pohyblivou čárkou.
- *modulo* (%) – Vrací zbytek po celočíselném dělení prvního čísla druhým číslem; např. 5%2 dává 1, 4,5%2,2 dává 0,1.
- *unární negace* (-) – Z kladného čísla dělá záporné a naopak.
- *inkrement* (++) – Zvyšuje hodnotu proměnné o jednotku. Využívá se zejména v cyklech (kapitola 4).
- *dekrement* (--) – Odečítá od proměnné jednotku.

## 2.5.2 Porovnávací (relační ) operátory

V tomto oddíle jsou popisovány porovnávací operátory JavaScriptu. Ty srovnávají různé hodnoty a v závislosti na výsledku vrací logické hodnoty pravda nebo nepravda. Používají se zejména v příkaze **if** a smyčkách **while** a **for** (kapitola 4).

***Přehled porovnávacích operátorů:***

- *rovnost* (==) – Vrací hodnotu *true*, jsou-li si jeho dva operandy rovny, a hodnotu *false* v případě, že si rovny nejsou. Tyto operandy mohou být jakéhokoli typu a definice rovnosti závisí na daném typu: dvě proměnné jsou si rovné, pouze pokud obsahují stejnou hodnotu; dva řetězce si jsou rovny, obsahují-li oba přesně stejné znaky.
- *nerovnost* (!=) – Porovnává přesný opak operátor rovnosti. Vrací hodnotu *true* tehdy, pokud si operandy nejsou rovny, a *false*, pokud si rovny jsou.
- *menší než* (<) – Pokud je první operand menší než druhý operand, vrací hodnotu true, jinak dává false.
- *větší než* (>) – Opak operátoru menší než. Vrací *true,* pokud je první operand větší než druhý operand; jinak vrací *false*.
- *menší než nebo rovno* (<=) – Vrací hodnotu *true*, pokud je první operand menší než nebo roven jeho druhému operandu; jinak vrací *false*.
- *větší než nebo rovno* (>=) – Vrací hodnotu *true*, pokud je první operand větší než nebo roven druhému operandu; jinak vrací *false*.

## 2.5.3 Logické operátory

Logické operátory vyhodnocují splnění několika podmínek naráz, provádí na nich tzv. logickou algebru, např.:

Je x>y a zároveň je a==b?
Platí z+1=8 nebo z+1=7? Nebo obojí?

V JavaScriptu máme tři základní logické operátory:

- *and* (&&) – Vrací *true* pouze tehdy, mají-li oba operandy hodnotu true, tj. platí ten i ten. Jinak vrací hodnotu *false*.
- *or* (||) – Vrací hodnotu *true*, pokud alespoň jeden z operandů má hodnotu *true*. Pokud ani jeden operand nemá hodnotu *true*, vrací funkce *or* hodnotu *false*.
- *not* (!) – Vrací hodnotu *true* tehdy, má-li operand hodnotu *false,* a pokud má operand hodnotu *true*, vrací *not* hodnotu *false*.

Operátorů má JavaScript daleko více. Tyto základní však v devadesáti procentech případů postačí, většina už je od těchto operátorů nějakým způsobem odvozena.

Následující část, která popisuje základy práce s řetězci, patří spíše do nové kapitoly, nikoliv do kapitoly popisující práci s proměnnými. Svým způsobem však práce s řetězci je práce

s proměnnými, proto jsem ji zařadil ještě sem, abych pak logicky navázal na další tématiku v kapitole třetí.

# 2.6 Práce s texty

Jak už bylo řečeno, řetězec je jeden ze základních datových typů JavaScriptu a programovacích jazyků vůbec. Pro práci s řetězci má JavaScript k dispozici vestavěný objekt **string**, který má definovánu několik užitečných metod a jednu vlastnost, které dokáží hodně usnadnit práci. Můžete tak například vyjmout z řetězce znak nebo podřetězec, vyhledat znak nebo podřetězec nebo třeba převést všechna písmena řetězce na kapitálky apod. Obecný zápis pak vypadá takto:

```
string.metoda()
string.vlastnost
```

kde metoda může nabývat některé z těchto nejpoužívanějších hodnot:

- **substring()** – Vrací část řetězce definovanou číslem prvního a posledního požadovaného znaku.
- **charAt()** – Vrací požadované znak řetězce.
- **toLowerCase()** – Převede řetězec na malá písmena.
- **toUpperCase()** – Převede řetězec na velká písmena.

a vlastnost jediné hodnoty:

- **length** – Zjistí délku řetězce, hodnotou vlastnosti je číslo.

Poslední řetězcovou operací, která se neprovádí přes objekt **string**, je slučování řetězců, které už bylo popsáno v kapitolce o datových typech proměnných. Jak je patrné z předchozí kapitoly, není na tom nic složitého.

## 2.6.1 Jak zobrazit hodnotu řetězce

Než se podíváme na jednotlivé metody podrobněji, zabrousím ještě kousek dále. Abyste si mohli sami zjistit, jakou hodnotu řetězec má po operaci s určitou metodou, musíte si jej vypsat na obrazovku. Existují dva základní způsoby, jak něco JavaScriptem zobrazit:

1) vypsat požadovaný řetězec (číslo, proměnnou) přímo do okna prohlížeče;
2) vypsat řetězec do oznamovacího okénka.

První způsob jsem už vlastně zmínil, a sice

```
document.write();
```

Jde o objekt **document**, tedy vlastní obsah stránky, a metodu **write**, funkci, která do něj vypíše data obsažená v závorkách. Metoda write však nevypisuje jen holý text nebo obsah proměnné, ale obsahuje přímo zdrojový kód HTML, který je prohlížečem zobrazen podle standardních pravidel. Tzn. pokud použijete např. značku **<B></B>**, bude výsledný text vypsán tučně.

Budete-li chtít tak vypsat obsah proměnné *soucet*, zadáte řádek JavaScriptu:

```
document.write(soucet);
```

Chcete-li napsat hodnotu proměnné *soucet* tučně, použijte tento řádek:

```
document.write("<B>",soucet,"</B>");
```

Pro lepší pochopení ještě uvedu jednoduchý příklad:

```
document.write("Ema má maso.<BR>");
document.write("<B>Ema má maso.</B><BR>");
document.write("<I>Ema má maso.</I><BR>");
document.write("<H2>Ema má maso.</H2>");
```

Ema má maso.
**Ema má maso.**
*Ema má maso.*

**Ema má maso.**

Metodě **write** můžete předat jakýkoliv zdrojový text HTML.

Druhým způsobem je pak zobrazení textu v okně metodou **alert** objektu **window**:

```
alert(soucet);
```

Prohlížeč zobrazí okno s hlavičkou Microsoft Internet Explorer, žlutým vykřičníkem (alert znamená výstraha), obsahem proměnné soucet a tlačítkem *OK*. Okno zmizí klepnutím na toto tlačítko.

## 2.6.2 Zjištění délky řetězce

Délku řetězce lze zjistit velice snadno. Máme-li např. řetězec „Babička“ a přiřadíme jej do proměnné *Kdo*:

```
Kdo = "Babička";
```

zjistíme jeho délku použitím vlastnosti **length**:

```
document.write(Kdo.length);
```

V okně prohlížeče se vypíše celé číslo, které udává počet znaků v řetězci *Kdo*, tedy 7.

## 2.6.3 Získání části řetězce

Někdy je nutné získat z nějakého řetězce pouze jeho část. Pokud si vzpomínáte na efekt s blikajícími obrázky ve druhé části, vzpomenete si také jistě na odřezávání přípony ze jména obrázku. To bylo důležité proto, aby skript byl nezávislý na použití obrazového formátu JPG a GIF – každý má jinou příponu.

K vyřezání části řetězce má objekt **string** definovánu metodu **substring**, jejímiž parametry jsou počátek a konec podřetězce, který chceme získat:

```
string.substring(od, do);
```

Na našem příkladě babičky, z níž bychom chtěli jenom prostředek, bychom to mohli aplikovat takto:

```
document.write(Kdo.substring(2,5));
```

Výsledkem vypsaným na stránce bude „bič“. První písmeno řetězce totiž není označeno indexem 1, ale indexem 0, tj. část řětězce 2 až 5 začíná na *třetí* pozici (0, 1, 2). Zákonům logiky by navíc odpovídalo, že vypsán bude řetězec „bičk“, nikoli „bič“. Podřetězec definovaný metodou **substring** totiž končí předposledním znakem, tj. na pozici 4, a tedy je vypsán pátý (ale s indexem 4). Platí, že počet znaků, který bude vyjmut, je *do* minus *od*, a tedy bez posledního znaku. Tato skutečnost je poněkud matoucí a je potřeba si na ni zvyknout.

Na rozdíl od VBScriptu nemá JavaScript funkci pro ořezání zprava. Tu supluje také metoda **substring**, příklad zjistí tři znaky zprava:

```
document.write(Kdo.substring(Kdo.length-3));
```

Prohlížeč vypíše řetězec „čka“. Z příkladu je také patrné, že není nutné zadávat parametr *do*, ten je automaticky nahrazen koncem řetězce.

Chcete-li naopak získat tři znaky zleva, uděláte to takto:

```
document.write(Kdo.substring(0,3));
```

Prohlížeč vypíše řetězec „Bab“.

## 2.6.4 Získání jednoho písmena řetězce

Abyste nemuseli při potřebě získat z řetězce pouze jedno písmeno používat metodu substring, má objekt string také metodu **charAT**. Její hodnotou je jedno písmeno definovaného řetězce. Chcete-li např. z babičky získat prostřední, tedy čtvrté písmeno, zadejte:

```
document.write(Kdo.charAt(4));
```

Prohlížeč vypíše písmeno „č“.

## 2.6.5 Převod řetězce na velká nebo malá písmena

Někdy je potřeba převést znaky řetězce na velká nebo malá písmena. I proto má objekt **string** metody:

- **toLowerCase()** – konvertuje řetězec na malá písmena;
- **toUpperCase()** – konvertuje řetězec na velká písmena.

Protože ani jedna z funkcí nepotřebuje pro svoji práci žádné parametry, zůstanou závorky prázdné. Úplně vynechat je ovšem není možné, prohlížeč by generoval chybu.

Dejme tomu, že potřebujeme program, který nejprve proměnou Kdo, jejímž obsahem je řetězec "Babička", převede na velká písmena, vypíše tento řetězec a následně jej převede na písmena malá a opět jej vypíše. V praxi to vypadá takto:

```
document.write(Kdo);
Velka=Kdo.toUpperCase();
document.write(Velka);
Mala=Velka.toLowerCase();
document.write(Mala);
```

Prográmek nejprve vypíše výchozí hodnotu proměnné *Kdo*, pak ji převede na velká písmena a vzniknuvší řetězec uloží do proměnné *Velka* a vypíše jej. Potom převede velká písmena na malá, uloží nový řetězec do proměnné *Mala* a obsah této proměnné vypíše. Na stránce tedy bude sled těchto babiček: „Babička“, „BABIČKA“ a „babička“.

**POZNÁMKA:** *Určitě jste si už všimli, jakým způsobem píšu jednotlivá označení metod, objektů, proměnných a jejich hodnot, příkazů JavaScriptu nebo značek HTML. Pro jistotu však uvedu rekapitulaci: Značky HTML jsou v této knize psány neproporcionálním tučným písmem, objekty, metody, vlastnosti a příkazy JavaScriptu tučným proporcionálním písmem, stejným jako normální text, a konečně proměnné a jejich hodnoty kurzivou. Kurzivou však nejsou psány hodnoty typu řetězec nebo číslo, pouze hodnoty slovní: null, true, false apod. Programový kód a kód HTML je vždy vypsán tučným neproporcionálním písmem pro zvýšení přehlednosti a lepší možnosti formátování (pevné mezery apod.).*

# 3. Vestavěné objekty JavaScriptu

Základy práce s proměnnými jste tedy úspěšně překousli, a když už jsem začal psát o vestavěných objektech JavaScriptu (**string**), nerad bych vynechal další dva, neméně důležité: **Math** a **Date**. **Math** se používá pro matematické operace a **Date** pro práci s datem a časem. JavaScript má tedy tři základní vestavěné objekty, každý má svoje vlastnosti a metody.

**POZNÁMKA:** *Vestavěný objekt jazyka JavaScript znamená, že tyto objekty způsobem nesouvisí s prohlížečem či jazykem HTML a jejich objekty (např.* **document**, **window***). Byly přidány tvůrci jazyka JavaScript pro rozšíření jeho možností. Je nutné je znát, protože se bez metod a vlastností, které nabízí, prakticky neobejdete.*
*Pro úplnost ještě dodám, že rozlišujeme celkem tři typy objektů:*

1) *Objekty jazyka HTML – velmi úzce souvisí s prvky HTML a webovou stránkou. Mohou to být linky, seznamy, každý z možných prvků formuláře.*
2) *Objekty prohlížeče – objekty umožňující pracovat přímo s prohlížečem a jeho prostředím. Sem patří objekty a jejich metody pro otevírání nového okna, práce se seznamem navštívených webových stránek apod.*
3) *Vestavěné objekty*

## 3.1 Objekt *string* – práce s řetězci

Když jsem psal o metodách objektu **string**, uvedl jsem pouze ty, které pracovaly s řetězcem samotným. Objekt string má však několik dalších metod, které však v praxi příliš nevyužijete. Tyto metody formátují daný řetězec pro zobrazení na webové stránce. Řetězec vypisovaný pomocí document.write lze formátovat značkami HTML, např. aby byl text vypsán tučně, bude uzavřen značkami **<B></B>**. Pro lepší pochopení příklad:

Dejme tomu, že chceme vypsat tučně proměnnou *Kdo*, která obsahuje řetězec „Babička“. Za použití současných znalostí byste to udělali asi takto:

```
document.write("<B>",Kdo,"</B>");
```

Pokud však budete obsah proměnné *Kdo* vypisovat častěji, budou se značky **<B></B>** opakovat, což vede k určité zdlouhavosti zápisu. Pomocí metody **bold** objektu **string** lze však značky pro tučný text přidat přímo do řetězce a v budoucnu pouze vypisovat nový řetězec:

```
Kdo2 = Kdo.bold();
document.write(Kdo2);
```

Oba příklady vypíšou tučně text „Babička“.

Pokud jste dobře pochopili předchozí příklad, nebude vám činit problémy užít další podobné metody objektu **string**.

**Tip:** *Pokud vás přesná syntaxe s příklady zajímá, můžete si jednotlivé metody nalistovat např. v referenční příručce* JavaScript – kompletní průvodce *Davida Flanagana.*

***Přehled dosud nepopsaných metod objektu string, abecedně řazeno:***

- **anchor()** – Převádí text na hypertextový odkaz, který je parametrem metody.
- **big()** – Převádí text na tučný, přidává značky **<BIG></BIG>**;
- **bold()** – Převádí text na tučný, přidává značky **<B></B>**.
- **fixed()** – Přidává k textu značky **<TT></TT>**.
- **fontcolor()** – Přidává značku **<FONT COLOR>** pro barevné zvýraznění textu. Barva je uvedena jako parametr metody ve formátu RGB v hexadecimálním zápisu.
- **fontsize()** – Nastavuje velikost textu značkou **<FONT SIZE>**. Velikost je uvedena jako parametr metody a může se pohybovat v rozmezí od 1 do 7.
- **indexOf()** – Metoda vyhledává znak (uvedený jako její parametr) v příslušném řetězci. Pokud je zadaný znak nalezen, metoda vrací index takového znaku. Pokud nalezen není, metoda vrátí číslo -1. Jako druhý parametr této metody může být uveden index znaku, od něhož se začne s prohledáváním.
- **italics()** – Převádí text na kurzivu, přidává značky **<I></I>**.
- **lastIndexOf()** – Významově je stejná jako metoda **indexOf()**, jen s vyhledáváním začne od konce řetězce.
- **link()** – Podobně jako metoda **anchor()** i tato metoda převádí text na hypertextový odkaz, tentokrát ovšem s parametrem **HREF**, který bude shodný s parametrem této metody.
- **small()** – Přidává k textu značky **<SMALL></SMALL>**.
- **strike()** – Přidává k textu značky **<STRIKE></STRIKE>**.
- **sub()** – Přidává k textu značky **<SUB></SUB>**.
- **sup()** – Přidává k textu značky **<SUP></SUP>**.

**Tip:** *Jednotlivé značky, které jsou zde uvedeny a které formátují text, jsem stručně vysvětlil v první kapitole. Podrobněji se jim věnuje kapitola 3 knihy* Tvorba WWW stránek pro úplné začátečníky, *Computer Press.*

## 3.2 Objekt *Math* – výpočet matematických funkcí

Objekt **Math**, jak už jsem zmínil, se používá pro počítaní různých matematických operacích. Kromě metod, které představují běžné matematické funkce, obsahuje také některé standardní konstanty. Na rozdíl od objektu **Date** (viz dále) je neměnný a vždy budete odkazovat přímo na něj, zatímco u objektu **string** jste ukazovali přímo na proměnnou, která obsahovala objekt typu **string**.

Dejme tomu, že chceme určit sinus 180 stupňů, a tedy PI:

```
var x=Math.PI;    // v proměnné x se zobrazí hodnota konstanty PI
document.write(Math.sin(x));
```

Podobně získáme i kosinus stejného úhlu:

```
document.write(Math.cos(x));
```

Lze tak pomocí *PI* třeba spočítat obvod kruhu:

```
var x=2;
document.write(Math.PI * 2 * x);
```

A tak dál a tak dál. Pomocí metod (matematické funkce) a vlastností (standardní konstanty) lze provádět prakticky jakékoliv matematické operace. To je celkem jasné a není potřeba delšího výkladu. Nyní tedy uvedu přehled základních matematických konstant a funkcí se stručným popisem, opět abecedně řazeno.

***Vlastnosti (konstanty) objektu Math:***

- **E** – matematická konstanta e, základ přirozených logaritmů LN (= 2,71828),
- **LN10** – logaritmus 10 o základě e (= 2,30259),
- **LN2** – logaritmus 2 o základě e (= 0,69315),
- **PI** – 3,141592653,
- **SQRT1_2** – druhá odmocnina z 1/2 (= 0,7071),
- **SQRT2** – druhá odmocnina ze dvou (= 1,4142).

***Metody (funkce) objektu Math:***

- **abs(x)** – Absolutní hodnota čísla *x*.
- **acos(x)** – Arckosinus čísla *x* (*x* je z množiny od -1,0 do 1,0).
- **asin(x)** – Arcsinus čísla *x* (*x* je z množiny od -1,0 do 1,0).
- **atan(x)** – Arctangens čísla *x*.
- **ceil(x)** – Zaokrouhlí desetinné číslo směrem nahoru.
- **cos(x)** – Kosinus čísla *x* (radiány);
- **exp(x)** – Exponent *e* na *x*.
- **floor(x)** – Zaokrouhlí desetinné číslo směrem dolů (odsekne desetinnou část).
- **log(x)** – Přirozený logaritmus čísla *x*.
- **min(x,y)** – Určí, která ze dvou hodnot *x* a *y* je menší, a tu vrátí.
- **max(x,y)** – Určí, která ze dvou hodnot *x* a *y* je větší, a tu vrátí.
- **pow(x,y)** – Umocní *x* na *y*.
- **random()** – Náhodné číslo v rozsahu od 0 do 1.
- **round(x)** – Zaokrouhlení podle klasických pravidel.
- **sin(x)** – Sinus čísla *x* (v radiány);
- **sqrt(x)** – Druhá odmocnina čísla *x*.
- **tan(x)** – Tangens čísla *x*; kotangens získáte obrácenou hodnotou tangentu.

Jednotlivé funkce nepotřebují, myslím, bližšího vysvětlení.

# 3.3 Objekt *Date* – práce s datem a časem

Posledním vestavěným objektem je **Date**, který umožňuje pracovat s datem a časem. Počítá čas od roku 1970 a ukazuje jej v podobě srozumitelné člověku. Na rozdíl od objektů **string** a **Math** nemá **Data** žádné vlastnosti, pouze metody. Zkuste si hodnotu objektu vypsat:

```
document.write(Date());
```

Uvidíte nepříliš srozumitelný řetězec, který vypadá nějak takto:

```
Mon Dec 27 13:24:36 1999
```

Pro přežvýkání tohoto řetězce (den, měsíc, čas a rok) má **Date** několik metod, které z něj vybírají jen to, co chceme. Bohužel, s objektem **Date** nelze pracovat tak, že bychom třeba napsali:

```
Date.metoda();
```

Je nutné nejprve přiřadit objekt **Date** nějaké proměnné pomocí klíčového slova (konstruktoru) **new**:

```
Datum = new Date();
```

Tímto byla vytvořena proměnná *Datum*, která je, zjednodušeně řečeno, novým objektem. Tento objekt je však prázdný, nemá žádné definované vlastnosti. Spíše je takovým zástupcem, ukazatelem na objekt **Date**. V praxi se konstruktoru mnoho nepoužívá, v případě objektu **Date** je to však nezbytné.

Využití objektu **Date** na různých příkladech uvidíte později, teď se omezím na pouhé vyjmenování základních metod a stručného popisu (abecedně řazeno).

***Metody objektu Date:***

- **getDate()** – Vrací číslo dne v tomto měsíci.
- **getDay()** – Vrací pořadové číslo dne v aktuálním týdnu (0-6, kde 0 je neděle).
- **getHours()** – Vrací aktuální hodinu od 0 (půlnoc) do 23.
- **getMinutes()** – Vrací aktuální minutu 0 až 59.
- **getMonth()** – Vrací číslo, které reprezentuje měsíc v roce; 0=leden, 11=prosinec.
- **getSeconds()** – Vrací aktuální sekundu mezi 0 a 59.
- **getFullYear()** – Vrací rok ve tvaru RRRR (1999, 2000).
- **setDate(x)** – Nastaví den v měsíci (1 až 31).
- **setHours(x)** – Nastaví hodinu (0 až 23).
- **setMinutes(x)** – Nastaví minutu (0 až 59).
- **setFullYear(x)** – Nastaví aktuální rok (1999, 2000).

# 4. Základní příkazy JavaScriptu

Výkonnými jednotkami každého programovacího jazyka jsou příkazy. Program v JavaScriptu, stejně jako v každém jiném programovacím jazyku, je jednoduše souborem příkazů. Jednotlivé příkazy pracují s různými parametry, které zpracovávají. V programu jsou odděleny středníky, stejně jako všechny výrazy a operandy JavaScriptu. V této kapitole se budu věnovat pouze výkonným příkazům, příkazy pracující s funkcemi (**function**, **return**) si nechám až na samostatnou kapitolu o funkcích.

## 4.1 Podmínka – příkaz *if*

Podmínka je základní řídicí strukturou programu. Na základě splnění nebo nesplnění určité podmínky program provede určitou operaci. Pokud vzpomenete na první příklad generování náhodných čísel do loterie, i tam byla podmínka: pokud se vygenerované číslo rovná nějakému již taženému, generuj znovu. Pokud ne, můžeš pokračovat. Bez podmínky bychom se v tomto případě neobešli.

Podmínku v JavaScriptu, stejně jako ve většině programovacích jazyků definuje příkaz **if**. Ta rozhodne, zda je podmínka uvedená v parametru příkazu pravdivá nebo ne a podle toho buďto provede nebo neprovede příkaz, který následuje. Struktura příkazu **if** je následující:

```
if (výraz) příkaz;
```

Znamená to, že pokud je výraz uvedený v závorce platný (a tedy má hodnotu *true* – viz kapitola o proměnných a datových typech), bude proveden daný příkaz. Pokud if vyhodnotí výraz jako *false*, tedy nepravdivý, příkaz se neprovede. Příklad:

```
if (jmeno == "Petr") alert ("Ahoj, Petře!")
```

Prohlížeč vypíše pozdrav „Ahoj, Petře!“ pouze tehdy, má-li proměnná jmeno hodnotu „Petr“.

**Poznámka:** *Závorky okolo výrazu jsou nutnou součástí příkazu* **if**. *V tomto případě sice vypadají zbytečně, pokud je však definovaná složená podmínka, nemohla by být správně vyhodnocena.*

Málokdy však stačí po splnění podmínky provést pouze jeden příkaz; většinou jich je nutné provést více. To naštěstí příkaz if umožňuje:

```
if ((jmeno == null) || (jmeno == "")) {
      jmeno = "Jméno není zadáno";
      alert("Prosím, zadejte svoje jméno.");}
```

V tomto případě se nejen provede více příkazů, je-li podmínka platná, podmínku však také definují dva výrazy spojené logickým operátorem *or*, nebo (viz kapitola 2.5). Tyto dva výrazy

jsou každý ohraničeny závorkami, přičemž celý složený výraz je ohraničen ještě jednou, aby bylo jasné, kde je jeho konec a kde začátek.

Jak je patrné z tohoto příkladu, je celý blok příkazů, který se má provést v případě, že jméno nebylo zadáno uživatelem, uzavřen do složených závorek. Veškeré blokové příkazy, funkce apod. se v JavaScriptu uzavírají do složených závorek.

Co však dělat, když chceme provést nějakou operaci, nějaký příkaz i v případě, že podmínka splněna není? Při použití současných znalostí bychom to udělali takto:

```
if ((jmeno == null) || (jmeno == "")) {
      jmeno = "Jméno není zadáno";
      alert("Prosím, zadejte svoje jméno.");}

if ((jmeno != null) && (jmeno != ""))
       alert("Jmenujete se ", jmeno);
```

Tento program provede následující:

1) Pokud je zadáno jméno, a tedy proměnná má nenulovou velikost, první podmínka se ignoruje a druhá pak vypíše hlášení, jak se jmenujete.
2) Pokud není zadáno jméno (to nastane v případě, že proměnná *jmeno* nemá žádnou hodnotu nebo je hodnotou prázdný řetězec), je nastavena proměnná *jmeno* na řetězec „Jméno není zadáno“. Při vyhodnocování pravdivosti druhé podmínky však zjistí, že se jmeno nerovná ani *null*, ani prázdnému řetězci a vypíše, že se jmenujete „Jméno není zadáno.“ A to je špatně, tato podmínka měla být logicky ignorována, protože žádné jméno zadáno nebylo. V tomto případě bychom do druhé podmínky museli přidat ještě jednu podmínku, a sice:

```
&& (jmeno !="Jméno není zadáno.")
```

Jedině tehdy program proběhne správně.

Sami vidíte, že vyřešit případ, že jméno není zadáno, je při současných znalostech problematické. Proto má příkaz **if** ještě jeden tvar, který umožňuje řešit případ, že podmínka není splněna v jednom jediném průchodu a není nutné definovat žádné další podmínky:

```
if (výraz) příkaz1
else příkaz2
```

V této formě příkazu je výraz vyhodnocen, a pokud je pravdivý, je proveden *příkaz1*; jinak je proveden *příkaz2*. Vrátím-li se k předchozímu příkladu, vypadalo by to takto:

```
if ((jmeno == null) || (jmeno == "")) {
      jmeno = "Jméno není zadáno";
      alert("Prosím, zadejte svoje jméno.");}
else alert("Jmenujete se ", jmeno);
```

Myslím, že je docela jasné, jak příklad funguje. Pokud není jméno zadáno, přiřadí se do proměnné *jmeno* hodnota „Jméno není zadáno.“ a vypíše se hlášení, abyste je zadali. V opačném případě se vypíše hlášení, že se jmenujete třeba Albert.

Aby bylo úplně jasné, jak podmínková struktura pracuje, uvedu ještě jeden příklad, z něhož to možná bude patrnější: Máme dvě proměnné *a*, *b* a chceme zjistit, zda se *a* rovná *b* nebo ne. Můžeme použít tento jednoduchý program:

```
if (a == b)
     document.write("a se rovná b");
else
     document.write("a se nerovná b");
```

Příkaz **if...else** může být i složený, to znamená, že uvnitř jedné podmínky **if** může být definovaná i další podmínka **if** a tak dál. Tehdy je nutné dát si pozor na to, že příkaz **else** je součástí nejbližšího příkazu **if**. Pokud se příkaz **else** omylem připlete do cesty jinému příkazu **if**, budou příkazy a podmínky vyhodnocovány a prováděny úplně jinak.

Obecný zápis složeného příkazu **if...else**:

```
if (výraz1)
     if (výraz2) příkaz1
     else příkaz2
else příkaz3
```

nebo

```
if (výraz1)
     if (výraz2) příkaz1
     else příkaz2
else if (výraz3) příkaz3
     else příkaz4
```

Možných kombinací je nespočetně mnoho a závisejí na konkrétních případech. Příkaz **if...else** je však pro začátečníky ideálním místem programu, kde se dá dokonale ztratit, a mohou tápat hodně dlouho, než přijdou na to, že tento příkaz **else** vůbec k tomuto příkazu **if** vlastně vůbec nepatří.

## 4.2 Základní cyklus – příkaz *while*

Stejně jako je příkaz **if** základním řídicím a rozhodovacím příkazem, který JavaScriptu umožňuje rozhodovat, a na základě tohoto rozhodnutí provést nebo neprovést daný příkaz, je příkaz **while** základním příkazem, kterým JavaScriptu dovoluje provádět opakující se akce. Používá se např. tehdy, když chcete provádět určitou akci tak dlouho, dokud je nějaká podmínka splněna. Jakmile není, je cyklus přerušen a JavaScript pokračuje dále.

Tento cyklus má velice podobnou syntaxi příkazu **if**:

```
while (výraz) příkaz;
```

Většinou však místo jednoho příkazu následuje blok příkazů, který provádí danou akci a zároveň nějakým způsobem mění vyhodnocovanou podmínku.

A jak to pracuje? Nejdříve je vyhodnocen výraz; je-li nepravdivý, je příkaz přeskočen a JavaScript pokračuje dalším příkazem programu. Pokud je vyhodnocen jako pravdivý, tedy *true*, je proveden zadaný příkaz a poté je opět vyhodnocena podmínka zadaná ve výrazu. A opět, pokud je hodnota výrazu nepravdivá, přesune se JavaScript na další příkaz v programu; jinak provede příkaz a cyklus pokračuje. A to až do doby, kdy je výraz vyhodnocen jako nepravdivý, kdy se cyklus ukončí.

Lze tak (většinou omylem nebo chybou programátora) vytvořit nekonečnou smyčku. Většinou se totiž v těle smyčky jedna proměnná, ta, která ovlivňuje hodnotu výrazu, mění s každým opakováním smyčky. Protože se tato proměnná mění, může se (ale nemusí) měnit akce provedené vykonáním příkazů v těle smyčky, pokud jsou nějak touto měnící se proměnou ovlivněny, jak je to vidět na jednoduchém příkladu smyčky **while**:

```
citac = 0;
while (citac < 10) {
      document.write(citac + "<BR>");
      citac++;}
```

Tento program vypíše do sloupečku čísla od 0 do 9. Proměnná *citac* začíná na hodnotě 0 a je vždy zvýšena o jednotku (*citac*++) vždy, když proběhne jedno tělo smyčky. Jakmile je smyčka provedena desetkrát, výraz se stane nepravdivým (proměnná *citac* už není menší ale rovna deseti) a příkaz **while** skončí.

**POZNÁMKA:** *Většina smyček cyklu* **while** *používá nějakou takovou čítací proměnnou, která určuje počet provedených cyklů (např. výpočet faktoriálu). Pozor však na zacyklení!*

## 4.3 Rozšířený cyklus – příkaz for

JavaScript má k dispozici ještě jeden příkaz pro smyčku, který bývá častokrát vhodnější než příkaz **while**. Příkaz for využívá model společný pro většinu smyček (včetně výše uvedené smyčky **while**). Jak pracují takovéto smyčky?

Většina smyček má nějakou čítačovou proměnnou, která je inicializována *před začátkem smyčky*. Pak je testována jako součást výrazu, který se vyhodnocuje před každým opakováním smyčky. Na konci smyčky je tato čítačová proměnná zvýšena o jednotku nebo jinak změněna předtím, než je výraz znovu vyhodnocen (podívejte se na příklad smyčky **while** o kousek výše).

Tohle všechno, inicializaci čítače, test nějakého výrazu, jehož součástí čítač bývá, a aktualizace čítače jsou tři rozhodující manipulace smyčkové proměnné. Příkaz **for**, na rozdíl od **while**, tohle všechno kombinuje v sobě:

```
for (inicializace čítače ; výraz ; změna čítače)
      příkaz
```

Nejjednodušším způsobem jak vysvětlit předchozí zápis, je ukázat totéž na smyčce **while**:

```
inicializace čítače;
while(výraz) {
      příkaz
      změna čítače;}
```

Tímto se příkaz **for** liší od podobného příkazu ve VBScriptu, který má předem určený počet opakování a žádný výraz nevyhodnocuje. Příkaz for je tak v JavaScriptu daleko univerzálnější. Abyste lépe pochopili jak příkaz **for** pracuje, uvedu jednoduchý příklad, který má stejnou funkci jako smyčka vytvořená příkaze **while** výše:

```
for (citac = 0; citac< 10 ; citac++)
      document.write(citac + "<BR>");
```

Všimněte si, že tato syntaxe umísťuje všechny důležité informace o řídicí proměnné smyčky na zvláštní řádek, což vyjasňuje provádění smyčky. Rovněž si povšimněte, že umístění inkrementačního výrazu v příkazu **for** zjednodušuje tělo smyčky na jednoduchý příkaz a my dokonce nemusíme používat složené závorky, abychom vytvořili příkazový blok.

**Příklad:** *Výpočet faktoriálu čísel od 1 do 10*

```
document.write
      ("<B><FONT SIZE=4>Tabulka faktoriálů</FONT></B><P>");
for (var i = 1, f = 1; i <= 10; i++, f *= i)
document.write(i + "! = " + f + "<BR>");
```

Výsledkem tohoto programu bude tato jednoduchá tabulka. V levém sloupečku je číslo, ke kterému je počítán faktoriál, a v druhém pak samotný faktoriál:

```
Tabulka faktoriálů
1 ! = 1
2 ! = 2
3 ! = 6
4 ! = 24
5 ! = 120
6 ! = 720
7 ! = 5040
8 ! = 40320
9 ! = 362880
10 ! = 3628800
```

Smyčky mohou být samozřejmě mnohem složitější, než jsou tyto jednoduché příklady. Dokonce se může s každým opakováním smyčky měnit více než jedna proměnná. Jednotlivé proměnné, které se mají měnit, jsou zde odděleny čárkou:

```
for(i = 0, j = 10 ; i < 10 ; i++, j-)
      sum += i * j;
```

## 4.4 Přerušení cyklu – příkazy *break* a *continue*

Cykly lze i předčasně ukončit, pokud je to potřeba, „nestandardním“ způsobem, tj. jiným způsobem, než by bylo nesplnění podmínky definované v těle řídicího příkazu. K tomu slouží příkaz **break**, která má naprosto triviální syntaxi:

```
break;
```

Tento příkaz je určen pro cykly **while** a **for**, nelze jej použít nikde jinde. Jinde by ani neměl žádné využití.

Základní funkcí příkazu break je ukončení právě běžícího cyklu – tehdy JavaScript předá řízení hned následujícímu příkazu, který je umístěn za tímto cyklem.

Jeho využití je patrné třeba z následujícího příkladu. Ten prohledává řetězec a hledá v něm konkrétní znak. Jakmile jej najde, je cyklus ukončen. Je to taková obdoba metody **string.charAt()**:

```
var kde = "Desatero hor a desatero řek"
var hledam = "h"
for(i = 0; i < kde.length; i++)
  {
      if (kde.charAt(i) == hledam)
            break;
  }
```

Program bere jednotlivé znaky řetězce zleva doprava a porovnává je s proměnnou *hledam*. Jakmile narazí v řetězci na písmeno „h“, skončí příkazem **break**. Pokud nenajde nic, ukončí se korektně cyklus podmínkou *i<kde.length*, tzn. v řetězci už není žádné další písmeno.

Existuje však i jiné přerušení provádění cyklu příkazem **continue**. To však neukončí běh celé smyčky a nepředá řízení dál, ale ukončí pouze a jenom provádění aktuálního cyklu a jde znovu na příkaz **while** nebo **for**, podle toho, kde jej použijete. Má stejně jednoduchou syntaxi:

```
continue;
```

Co se přesně při provedení příkazu **continue** děje? Aktuální opakování uzavřeného cyklu je ukončeno a začíná nové. Pokud jde o smyčku **while**, je výraz, který ovlivňuje běh smyčky,

znovu testován a je-li pravdivý, je tělo smyčky opět provedeno. Trošku jinak je tomu u smyčky **for**. Zde je nejdříve změněna čítačová proměnná podle výrazu definovaného v těle příkazu **for**, a teprve pak je výraz testován. Další opakování se provede tehdy, je-li pravdivý, v případě opačném se cyklus ukončí.

Tento příklad ukazuje použití příkazu **continue** k přerušení aktuálního opakování cyklu v případě, že nastane chyba (daný výraz má hodnotu *null*):

```
for (i = 0; i < lide.length; i++)
  {
      if (lide[i] == null)
            continue;
      celkem += lide[i];
  }
```

Příklad vychází ze skutečnosti, že přičtení hodnoty *null* k proměnné není možné. Proto je tomuto příkazem **continue** zabráněno a cyklus jde pracovat s další proměnnou v pořadí; nejdříve však zvýší proměnnou i o jednotku.

Pro ještě lepší pochopení lze použít pro ten samý efekt i poněkud upravený program:

```
for (i = 0; i < lide.length; i++)
  {
      if (lide[i] != null)
            celkem += lide[i];
  }
```

Přičtení proměnné *lide[i]* se provede pouze v případě, že se tato proměnná nerovná *null*. Funkce příkazu **continue** je teď snad už docela jasná.

## 4.5 Jednoduchá práce s objekty – příkaz *with*

Poslední příkaz, o němž chci psát, je příkaz **with**. Ten slouží k zjednodušení zápisu příkazů a operací pro práci s objekty. Říká, s jakým objektem se mají dělat následující operace. Pro lepší pochopení příklad. Dejme tomu, že chcete pracovat s tímto objektem:

```
frames[1].document.forms[0].tlacitko.value
```

Už jen zápis je zdlouhavý, to, že je nepřehledný, je docela patrné. Příkaz **with** má tuto syntaxi:

```
with (objekt)
       příkaz;
```

Objekt specifikovaný v závorce příkazu **with** se pro další operace v těle příkazu **with** stává implicitním, tzn. všechny operace, které budou provedeny, se budou týkat tohoto objektu.

Může tu být jak jeden příkaz, tak i blok příkazů uzavřených do složených závorek. Příkaz with lze použít ke zjednodušení např. tohoto příkladu:

```
x = Math.sin(i * Math.PI);
y = Math.cos(i * Math.PI);
```

V tomto příkladu se dvakrát vyskytuje objekt **Math**. Pokud použijete příkaz **with** s tímto objektem, lze zápis podstatně zjednodušit a zpřehlednit:

```
with(Math) {
      x = sin(i * PI);
      y = cos(i * PI);}
```

Kdykoliv se v bloku příkazů příkazu **with** objeví nějaký identifikátor (proměnná apod.), zjišťuje JavaScript, zda není tento identifikátor metodou nebo vlastností objektu uvedeného v příkazu **with**. Pokud ano, pracuje se s tímto objektem; pokud ne, pracuje JavaScript s tímto identifikátorem tak, jak by pracoval normálně.

Lze tak např. zjednodušit i často používanou metodu objektu **document**, která se používá pro zápis na webové stránky:

```
with(document) {
      write(x);
      write(y);
      write(z);}
```

Použití příkazu **with** je poněkud složitější, než je tady prezentováno. Způsob vyhledávání indentifikátorů není tak přímočarý, pro podrobné vysvětlení bych však musel zbytečně zabíhat do podrobností. Opět proto odkazuji na příručku Davida Flanagana *Kompletní průvodce JavaScriptu*.

# 5. Funkce a jejich využití

Každý program má nějakou strukturu. Kdy bude jaký příkaz proveden, záleží na jeho umístění v programu. Program může vypadat třeba takto:

```
datum = new Date();
hodiny = datum.getHours();
minuty = datum.getMinutes();
sekundy = datum.getSeconds();
cas = hodiny;
cas += ((minuty < 10) ? ":0" : ":") + minuty;
cas += ((sekundy < 10) ? ":0" : ":") + sekundy;
document.write(cas);
```

Funkcí tohoto příkladu je vypsat na stránku aktuální čas ve tvaru 11:05:12, kde 11 jsou hodiny, 05 jsou minuty a 11 jsou sekundy. Je to posloupnost činností, které se provádějí za sebou a které mají na konci nějaký výsledek a tím je vypsání času.

Tuto část kódu lze do zdrojového kódu zařadit na začátek, aby při načtení zobrazil na stránce aktuální čas. Ale co když tento čas budete chtít každou sekundu nebo minutu aktualizovat? V tom případě bude lepší tento program nadefinovat jako funkci, která bude volána jednou za určitý časový úsek. V programu bude tento kousek kódu umístěn jenom jednou, ale proveden může být libovolněkrát.

Nebo jiný příklad. Víte, že se na stránku můžete cokoliv zapsat metodou **write** objektu **document**. Ale co když chcete při každém výpisu třeba zapsat konec řádku (**<BR>**) nebo napsat daný text tučně? Přece nebudete stále používat

```
document.write(text, "<BR>");
```

nebo

```
document.write("<B>", text, "</B>");
```

V tomto případě je také lepší použít funkci, které jako parametr předáte požadovaný text, který se má napsat, a tato funkce už potřebné parametry přidá automaticky sama. Zápis se zjednoduší a zprůhlední a vám to ušetří práci při psaní zdrojového kódu programu.

S funkcemi jste se v této kapitole už setkali, a to dokonce na několika místech. Vzpomeňte si např. na kapitolu o vestavěných objektech JavaScriptu: tam jsem popisoval objekty **string**, **Math** a **Date** a práci s jejich metodami. Metody je vlastně jej jiný název pro funkci; objekt je soubor pojmenovaných hodnot – vlastností a funkcí, které umožňují uživateli s těmito objekty pracovat. Např. pokud chcete získat hodnotu kosinus úhlu 1,8, napsali byste tuto matematickou funkci:

x=cos 1,8

V JavaScriptu napíšete vlastně také funkci, která se bude pouze lišit zápisem, efekt však bude mít stejný:

```
x=Math.cos(1.8);  /* v JavaScriptu se používá desetinná tečka */
```

Objekt **Math** a jeho metody jsou předdefinované funkce, které má JavaScript vestavěny, stejně jako řadu dalších (např. metody pro práci s řetězci, datem apod.). JavaScript vám však umožňuje naprogramovat funkce vlastní, které pak můžete ve svém hlavním programu libovolně vyvolávat a používat. Funkce se tedy používají zejména pro opakované činnosti a obsluhu různých událostí, které mohou při běhu nastat (viz kapitola 2).

Pokud vezmu za příklad třeba funkci pro výpis tučného textu, bude zápis programu vypadat takto:

```
function napis(text) {
document.write("<B>", text, "</B>");}

napis("Ahoj!");
napis("Jak se máte?");
napis("Docela dobře.");
```

Z předchozího příkladu je patrné, že funkcím lze předávat i data, která mají zpracovat. Ta jsou předávána jako argumenty volané funkce, tzn. naší funkci byl předán jako argument nějaký text, který je následně ve vlastní funkci uložen do proměnné *text*, s kterou se dále pracuje. Jako argument může posloužit jakákoliv proměnná, objekt, pole apod.

# 5.1 Zápis, definice a volání funkce

Obecný zápis pro funkci je tedy takovýto:

```
function jmenofunkce ([a_1 [,a_2 […, a_n]]])
{
     tělo funkce
}
```

kde jednotlivé části mají tento význam:

- *jmenofunkce* – Název funkce, která je tímto definována.
- *a_1 … a_n* – Seznam libovolného počtu argumentů funkce, které jsou odděleny čárkami. Když bude funkce volána, budou těmto argumentům přiřazeny hodnoty specifikované ve výrazu volání funkce.
- *tělo funkce* – Blok příkazů, které mají být vykonány při volání funkce.

Všimněte si, že tělo funkce, seznam příkazů, je uzavřen do složených závorek, podobně jako smyčky **while** a **for** v případě, že vykonávají blok příkazů. Rozdílem však je, že zatímco v případě jediného příkazu v těle smyček se tyto závorky použít nemusí, u funkce jsou složené závorky součástí syntaxe a tudíž jsou povinné; a to i v případě, že je obsažen v těle funkce pouze jediný příkaz.

**POZNÁMKA:** *Příkaz* **function** *se liší od ostatních příkazů, které byly popsány v kapitole 4, tím, že příkazy, které tvoří tělo funkce, nejsou provedeny hned při spuštění programu, ale až v případě, že jsou volány buďto z hlavního programu nebo v případě zpracování nějaké události (viz kapitola 2).*

Až dosud jsem psal o funkcích, které provádí nějakou operaci, např. že jim předáte text, který má být zobrazen. Druhou variantou je však funkce, které předáte nějaké argumenty, funkce je zpracuje a vrátí nějaký výsledek, ať už je jakéholiv typu. Takovou funkcí je např. již zmíněný kosinus:

```
x=Math.cos(1.8);
```

Návratovou hodnotou funkce **Math.cos** je kosinus úhlu 1,8, který bude přiřazen do proměnné *x*. Příkazem, který zajistí, aby funkce vrátila nějaký výsledek, který bude dále zpracován (vypsán na obrazovku, přiřazen do nějaké proměnné), je **return**, neboli česky návrat. Jeho použití ukážu na jednoduchém příkladu, který porovná dvě hodnoty a řekne, která z těchto hodnot je menší: pokud první, vrátí funkce číslo 1, pokud druhá, vrátí 2, pokud jsou si rovny, vrátí nulu:

```
function porovnej(a,b) {
     if (a < b) return 1;
     if (a > b) return 2;
     else return 0;
}
```

Pokud byla funkce jednou definována, může být kdykoliv vyvolána a provedena. Funkci z toho příkladu vyvoláme nejjednodušeji takto:

```
porovnej(x,y);
```

Protože však chceme s výsledkem porovnání dále pracovat, musíme toto volání funkce přiřadit nějaké proměnné nebo třeba vypsat na obrazovku:

```
vysledek=porovnej(2,5);
```

nebo

```
document.write(porovnej(2,5));
```

Kdykoliv zavoláme funkci porovnej a předáme jí požadované argumenty, provede jejich vyhodnocení a vrátí buďto 0, 1 nebo 2 podle výsledku. Abyste pochopili přesně, co příkaz **return** dělá, popíšu přesně krok za krokem, jak pracuje:

1) Přečte argumenty *a*, *b*, které se mají porovnat.
2) Porovnává, zda je *a* menší než *b*; pokud ano, zastaví běh funkce a předá řízení zpět programu, který funkci volal a zároveň vrátí hodnotu 1.
3) Porovnává, zda je *a* větší než *b*; pokud ano, zastaví běh funkce a předá řízení zpět programu, který funkci volal, a zároveň vrátí hodnotu 2.
4) Poslední netestovanou možností je případ, kdy je *a* rovno *b*, automaticky tak ukončí běh funkce a vrátí hodnotu 0.

Z příkladu je patrné, že příkaz return nejen že vrací nějakou hodnotou, ale vždy také přeruší běh funkce, a to i tehdy, když v těle funkce ještě zůstávají další příkazy.

**POZNÁMKA:** *Když vyvoláte funkci, každý z výrazů, které specifikujete v závorkách, je vyhodnocen a výsledná hodnota je použita jako argument nebo parametr dané funkce. Tyto hodnoty jsou přiřazeny proměnným, pojmenovaným v definic funkce, a funkce operuje se svými parametry tak, že na ně odkazuje názvem. Tyto parametrické proměnné jsou však definovány pouze po dobu provádění dané funkce a jsou zrušeny při ukončení zpracování funkce.*

Obecný zápis příkazu **return** tedy je:

```
return [výraz];
```

Příkaz return nemusí vždy tak jednoduchou syntaxi, jako bylo uvedeno v příkladě porovnání dvou čísel. Může vypadat např. takto:

```
function mocnina(x) {return x*x;}
```

tzn. že přímo ve svém těle spočítá nějaký (i složitější) matematický výraz.

Příkaz **return** může být rovněž použit bez výrazu k jednoduchému ukončení provádění funkce bez vrácení hodnoty. Např.:

```
function zobraz_objekt(obj) {
     // Nejdříve funkce se zjistí, zda je objekt platný
     // a pokud ne, vynechá zbytek funkce a vrátí se zpět.
     if (obj == null) return;
     // zbytek funkce přijde sem
}
```

Jestliže funkce provede prázdný příkaz **return** nebo pokud nikdy neprovede příkaz **return** (to znamená, že jednoduše provede všechny příkazy ve svém těle a vrátí se standardní cestou), pak bude hodnota výrazu, který volá funkci, nedefinovaná.

Funkce **return** může obsahovat prakticky jakýkoliv výraz, který lze nějak vyhodnotit tak, aby byla určena nějaká návratová hodnota. To je patrné např. z příkladu funkce, která počítá a vrací vzdálenost mezi dvěma body:

```
function vzdalenost(x1, y1, x2, y2)
{
      var dx = (x2 - x1);
      var dy = (y2 - y1);
      return Math.sqrt(dx*dx + dy*dy);
}
```

Kdyby mohl příkaz **return** obsahovat pouze jednoduchý výraz, museli bychom přidat ještě jeden řádek zdrojového kódu:

```
delka = Math.sqrt(dx*dx + dy*dy);
return delka;
```

To je docela zbytečné, proto má příkaz **return** možnost pracovat i se složitějšími výrazy, aby takovéto řádky odpadly.

## 5.2 Opakované (rekurzivní) volání funkce

Funkce může být volána i rekurzivně, tj. volat sama sebe. Zní to složitě, ale je to docela prosté; pro příklad uvedu výpočet faktoriálu čísla zadaného čísla:

```
function faktorial(x)
{
      if (x <= 1)
            return 1;
      else
            return x* faktorial(x-1);
}
```

Jak tato funkce pracuje:

1) Funkci je z hlavního programu předána hodnota, z níž se má spočítat faktoriál. Faktoriál je číslo, které vznikne vynásobením všech celých kladných čísel, které jsou menší nebo rovny číslu, ke kterému hledáme faktoriál počínaje jedničkou: např. 5!=1*2*3*4*5=120.
2) Pokud je hodnota argumentu 1, je faktoriálem číslo 1, funkce vrátí hodnotu 1 a skončí.
3) Pokud je argumentem číslo větší, je vynásobeno hodnotou funkce *faktorial* s novým argumentem, který je o jedničku menší. Tady funkce volá sama sebe, provádí rekurzivní volání. Toto rekurzivní volání se provede tolikrát, jakou hod-

notu má původní argument snížený o jednotku, má-li např. hodnotu 5, provede se toto volání 4krát.

A teď konkrétně: Dejme tomu, že původní argument *x* má hodnotu 3. Bude volána funkce *faktorial* s argumentem 3. Protože 3 je větší než 1, provede se výraz *x*faktorial(3-1)*. Zavolá se tedy opět funkce *faktorial* s argumentem 2 (1. rekurzivní volání). Opět je 2 větší než jedna, provede se tedy výraz *x*faktorial(2-1)*. Tzn. bude opět zavolána (2. rekurzivní volání) funkce *faktorial* s argumentem 1. Ta vrátí hodnotu jedna (1 se rovná 1) 1. rekurzivnímu volání a provede se tedy výpočet *2*1*, jehož výsledek, 2 se předá původnímu volání funkce. Ta provede výpočet *3*2*, výsledek je tedy 6 a tato hodnota se předá jako výsledek původního volání funkce. Výsledkem funkce *faktorial(3)* je tedy číslo 6.

**POZNÁMKA:** *Všimněte si, že funkce* napis *(viz výše) neobsahuje příkaz* **return***, a tudíž nevrací hodnotu. Proto nemůže být použita jako součást jiného výrazu, na rozdíl od funkcí* vzdalenost *a* faktorial*, které hodnoty vrací. To je vidět právě u funkce* faktorial*, která součástí jednoho takového výrazu je.*

**UPOZORNĚNÍ:** *Protože JavaScript je netypový jazyk, neočekává funkce ani specifikace datových typů pro argumenty funkcí. Navíc JavaScript při provádění funkce ani nekontroluje, zda jste předali typ dat, který funkce očekává. JavaScript dokonce nekontroluje ani to, zda jste předali správný počet argumentů; pokud jich je více, hodnoty navíc budou jednoduše ignorovány, pokud méně, pak některé z parametrů dostanou nedefinovanou hodnotu – to většinou ale způsobí, že se funkce nebude chovat tak, jak by měla.*

Nyní máte dostatek znalostí k tomu, abyste mohli začít vytvářet vlastní programy v JavaScriptu. Tomuto se tedy bude věnovat další kapitola, která na několika příkladech ukáže základní možnosti, jaké JavaScript při vytváření webových stránek nabízí.

# 6. První programy v JavaScriptu

Tato kapitola na konkrétních příkladech ukazuje nejen využití toho, co jsem až dosud v předchozích kapitolách popisoval. Výpisy těchto programů vám umožní lépe pochopit, jak JavaScript vlastně funguje; pokud to nebude ze zdroje příliš jasné, bude připojen i krátký komentář funkce programu. To vše vám pomůže získat nové informace a seznámíte se s některými používanými objekty a metodami prohlížeče, které budou podrobně vysvětleny v poslední části této knihy.

## 6.1 Práce s formuláři

Základní funkcí JavaScriptu, která je asi nejvíce využívána při tvorbě stránek, je zpracování formulářových dat, které pak následně budou odeslány na zadanou e-mailovou adresu. Lze tak zkontrolovat, zda jsou vyplněna všechna pole, nebo přidat do e-mailové zprávy další informace.

### 6.1.1 Jednoduché zaslání formuláře e-mailem

Dejme tomu, že byste chtěli zaslat e-mailem data z formuláře, který můžete najít na začátku druhé kapitoly. Tento formulář v podobě, v jaké tam byl prezentován, byl neaktivní, sice jej šlo vyplnit, ale jinak nic nedělal. Pokud jste klepnuli na tlačítko *Odešli*, prohlížeč nijak nereagoval a nikam nic neposílal.

Aby takovýto formulář nebyl mrtvý a aby opravdu data odesílal na zadanou e-mailovou adresu, je nutné doplnit určité parametry do značky **FORM**, která ohraničuje formulář a řídí jeho akce. Kromě parametru **NAME**, který pojmenovává tento formulář, zná značka **FORM** ještě další, které zařídí, aby byl formulář opravdu odeslán. Vysvětlím přímo na příkladu z kapitoly 2:

```
<HTML>
<HEAD>
  <TITLE>Formulář</TITLE>
</HEAD>
<BODY>
<FORM NAME="Formular"
  ACTION="mailto:broza@cpress.cz?Subject=Formulář" METHOD="POST"
  ENCTYPE="text/plain">
<TABLE>
  <TR>
    <TD>Vaše jméno</TD>
    <TD><INPUT TYPE="text" NAME="Jmeno"></TD>
  </TR>
  <TR>
    <TD>Co jíte k snídani?</TD>
```

```
    <TD><INPUT TYPE="text" NAME="Snidane"></TD>
  </TR>
  <TR>
    <TD>Váš oblíbený nápoj</TD>
    <TD><INPUT TYPE="text" NAME="Napoj"></TD>
  </TR>
  <TR>
    <TD>Chcete mi něco vzkázat?</TD>
    <TD><TEXTAREA NAME="Vzkaz" COLS="50" ROWS="3"></TEXTAREA>
  </TR>
</TABLE>
<INPUT TYPE="submit" VALUE="Odešli">
<INPUT TYPE="reset" VALUE="Vymaž">
</FORM>
</BODY>
</HTML>
```

Vidíte, že celý formulář zůstal stejný, až na to, že přibyly parametry ve značce **FORM**. Parametr

```
METHOD="post"
```

definuje, že formulářová data budou odeslána e-mailem. Parametr

```
ACTION="mailto:broza@cpress.cz?Subject=Formulář"
```

určuje e-mailovou adresu, na kterou budou data zaslána (broza@cpress.cz) a zároveň jméno e-mailu („Formulář“), a konečně

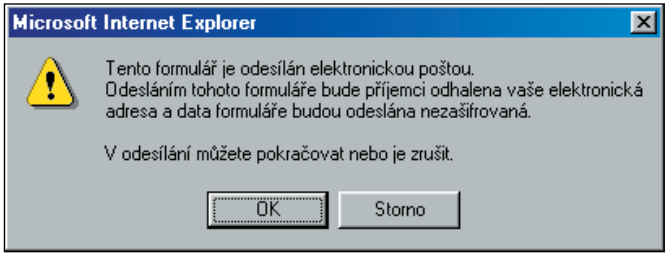
```
ENCTYPE="text/plain"
```

definuje způsob formátování e-mailové zprávy, v tomto případě prostý text bez jakýchkoliv formátovacích znaků.

Pokud tedy bude tento formulář odeslán, přijde příjemci určenému v parametru **ACTION** (broza@cpress.cz) e-mail s názvem „Formulář“ a s tímto obsahem:

```
Jmeno=Petr Broža
Snidane=Sendvič a káva
Napoj=Pivo nebo džus
Vzkaz=Ať se ti daří v novém roce!
```

Pro zaslání se použije váš implicitní klient elektronické pošty, standardně to bývá Outlook Express, který je dodáván k prohlížeči Internet Explorer. Než však bude e-mail zaslán, vypíše prohlížeč upozornění, že data budou zaslána e-mailem a že příjemce bude znát vaši e-mailovou adresu. Pokud souhlasíte, bude e-mail poslán.




Internet Explorer oznamuje, že bude odesílat data formuláře e-mailem.

## 6.1.2 Zaslání předformátovaných dat z formuláře e-mailem

Zatímco si v předchozím příkladě zasílání dat formuláře poštou JavaScript ani neškrtl, byl nutný k vysvětlení základní metody zasílání formulářů e-mailem a popisu klíčových parametrů značky **FORM**. Jdeme tedy dále. Dejme tomu, že chcete s odeslanými daty z našeho formuláře dostat i další informace o uživateli, např. jaký používá prohlížeč nebo z jaké stránky byl e-mail zaslán. K tomu už JavaScript potřebujeme.

```
<HTML>
<HEAD>
<TITLE>Formulář</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--

function Zpracuj(form) {
  var poslat = document.Formular;
  var info = "";
  var data = "";

  info = "\n\n" // přechod na nový řádek
  info += "Název stránky:  " + document.title + "\n";
  info += "Posláno ze stránky: " + document.location + "\n";
  info += "Použitý prohlížeč: " + navigator.appName;
  info += navigator.appVersion + "\n\n";
  data += "Jméno: "+poslat.Jmeno.value + "\n";
  data += "Snídaně: "+poslat.Snidane.value + "\n";
  data += "Nápoj: "+poslat.Napoj.value + "\n";
  data += "Vzkaz: "+poslat.Vzkaz.value + "\n";

  form.Anketa.value = info + data;

  alert("E-mailem budou poslány údaje:\n\n" + form.Anketa.value);
}
```

```
// -->
</SCRIPT>

</HEAD>

<BODY>
<FORM NAME="Formular">
<TABLE>
  <TR>
    <TD>Vaše jméno</TD>
    <TD><INPUT TYPE="text" NAME="Jmeno"></TD>
  </TR>
  <TR>
    <TD>Co jíte k snídani?</TD>
    <TD><INPUT TYPE="text" NAME="Snidane"></TD>
  </TR>
  <TR>
    <TD>Váš oblíbený nápoj</TD>
    <TD><INPUT TYPE="text" NAME="Napoj"></TD>
  </TR>
  <TR>
    <TD>Chcete mi něco vzkázat?</TD>
    <TD><TEXTAREA NAME="Vzkaz" COLS="50" ROWS="3"></TEXTAREA>
  </TR>
</TABLE>
<INPUT TYPE="reset" VALUE="Vymaž">
</FORM>

<FORM ACTION="mailto:Petr.Broza@cpress.cz?subject=Formulář"
  METHOD="post" ENCTYPE="text/plain" NAME="Druhy"
  ONSUBMIT="Zpracuj(this)">
<INPUT TYPE="hidden" NAME="Anketa" VALUE>
<INPUT TYPE="submit" VALUE="Odešli">
</FORM>

</BODY>
</HTML>
```

**Poznámka:** *Znak „\n“ znamená konec řádku a používá se pro odřádkování v řetězci.*

Tento zdrojový kód odešle na rozdíl od předchozího příkladu kromě formulářových dat také název stránky, v níž běží formulář, webovou adresu této stránky a použitý webový prohlížeč. Výsledný e-mail pak bude vypadat nějak takto:

Anketa=

Název stránky: Formulář
Posláno ze stránky: http://www.neznam.cz/Pokusy/Form2.html
Použitý prohlížeč: Microsoft Internet Explorer4.0 (compatible; MSIE 5.0; Windows 98; DigExt)

Jméno: Petr Broža
Snídaně: Sendviče a kávu
Nápoj: Pivo a džus
Vzkaz: Seš docela fajn kluk :)

Jak jste si jistě všimli, přibyl do zdrojového kódu ještě jeden formulář vedle toho původního. Zatímco ten první (*Formular*), do něhož zadáváte data, je doopravdy neaktivní a kromě tlačítka *Vymaž*, které nastaví původní hodnoty formuláře, vlastně nic nedělá. Aktivní je až druhý formulář, který provede vlastní odeslání pomocí nám známých parametrů **ACTION**, **POST** a **METHOD**. Paradoxní je, že tento formulář neobsahuje žádná data, jen jedno tlačítko typu *submit*, které data odešle, a jeden skrytý prvek typu *hidden*.

**Poznámka:** *Prvek typu* hidden *není na stránce vidět, je však normálním objektem formuláře. Jeho výhodou je, že lze ovlivňovat jeho vlastnosti a tudíž i vlastnosti formuláře.*

Jak tedy tato stránka pracuje?

1) Jsou vyplněny hodnoty formuláře; v případě potřeby je lze tlačítkem *Vymaž* smazat a začít znovu.
2) Je-li první formulář vyplněn, bude klepnuto na tlačítko *Odeslat*, které je však prvkem formuláře druhého.
3) V definici tohoto formuláře je uveden parametr **ONSUBMIT**, což je událost, která nastane při pokusu odeslat formulář. Klepnete-li na tlačítko typu *submit*, bude spuštěn skript, který je přiřazen parametru **ONSUBMIT**, a tedy funkce *Zpracuj*. Předávaným argumentem je formulář, v tomto případě formulář aktuální (*this*). Lze také předat přímo jméno formuláře, v tomto případě *Druhy*.
4) Je vytvořen objekt poslat, do něhož je umístěn objekt formuláře *Formular*. Každý formulář je totiž reprezentován objektem se jménem, které odpovídá tomu, jež bylo definováno parametrem **NAME**.
5) Z objektu **document** a **navigator** jsou postupně přečteny již zmíněné údaje o pro-hlížeči a stránce a umístěny do proměnné *info*.
6) Následně jsou z objektu poslat postupně čteny jednotlivé vlastnosti, které odpovídají jednotlivým políčkům formuláře, a tudíž mají shodná jména. Všechny vlastnosti jsou přiřazeny do proměnné data.
7) Tohle všechno je pak zobrazeno metodou **alert** v dialogovém okně prohlížeče pro případnou kontrolu uživatelem.
8) Protože ve formuláři *Druhy* byl vytvořen objekt typu *hidden*, který má jméno *Anketa*, byla zároveň vytvořena vlastnost *Anketa* formulářového objektu *Druhy*.

Formulář *Druhy* má tedy jedinou vlastnost, a této vlastnosti je přiřazena hodnota řetězce *info* + *data*.

9) Když je funkce ukončena, je obsah formuláře *Druhy* (a tedy jeho vlastnost *Anketa*) odeslán na danou e-mailovou adresu. Proto také začíná „Anketa=“, protože to, co je v těle e-mailu, je vlastně obsah vlastnosti *Anketa*.

Pokud se vám popis práce programu zdál příliš složitý, je to proto, že se tu pracuje s objekty, o nichž jsem zatím nikde nepsal. Pominu-li objekty **document** a **navigator**, kterým a mnoha dalším se budu věnovat v další části, mluví se tu o objektu formuláře. Jak už jsem psal, každý formulář při své definici v kódu HTML vytvoří nový objekt, který bude nést jméno tohoto formuláře a tedy to, co bylo definováno parametrem **NAME** v těle značky **FORM**. Každé pole, každé tlačítko, které ve formuláři vytvoříte, bude mít svoji analogii jako vlastnost objektu formuláře. S každou touto vlastností lze pracovat i jinak, než jen zadávat z klávesnice nebo klepáním myší, a to klasickým způsobem z JavaScriptu. Ostatně myslím, že to bylo dost názorně předvedeno právě na tomto příkladu.

## 6.1.3 Odeslání formuláře a přesměrování na zadanou stránku

Po odeslání formuláře metodami, jaké jsem ukázal v předchozích případech, zůstává načtena v prohlížeči původní stránka s formulářem. Tzn. pokud chcete přejít na jinou stránku, musíte buďto zadat adresu do prohlížeče nebo se vrátit tlačítkem *Zpět* na předchozí stránku. Jde však velice jednoduše zařídit, aby se po odeslání formuláře načetla do aktuálního okna stránka jiná, kde může uživatel pokračovat dále.

```
<HTML>
<HEAD>
<TITLE>Odeslání formuláře a přesměrování</TITLE
</HEAD>

<BODY>
<FORM NAME="Test" ENCTYPE="text/plain"
  ACTION="mailto:broza@cpress.cz" METHOD="post"
  ONSUBMIT=window.setTimeout("location='index.html'",1)>
...
vlastní tělo formuláře
...
<INPUT TYPE="submit" NAME="Submit" VALUE="Odešli">
</FORM>
</BODY>
</HTML>
```

Ve značce **FORM** je použito stejně jako v předchozím příkladu události **onsubmit** (parametr **ONSUBMIT**), která zpracuje skript v případě, že je formulář odeslán směrem na server. Jak je patrné, pokud má skript pouze jeden řádek, lze jej umístit přímo jako parametr události a není

nutné vytvářet funkci, na kterou událost odkáže. Prohlížeč dokonce sám pozná, v jakém jazyce je tento kousek skriptu napsán.

V tomto případě se využívá objektu **location**, který je definován jako vlastnost objektu **window**, která určuje, jakou stránku prohlížeč zobrazí. Používá se např. při klepnutí na odkaz definovaný značkou **<A HREF>**. Tato stránka může být libovolná, umístěná jak lokálně, tak na webovém serveru; sem se píše celá cesta.

Nové určení objektu **location** je však umístěno v parametru metody **setTimeout**, která je metodou objektu **window**. Tato metoda spouští nějakou funkci za daný časový interval, který je zde jedna milisekunda. Tento řádek kódu

```
window.setTimeout("location='index.html'",1);
```

tedy načte do aktuální stránky soubor *index.html*.

**POZNÁMKA:** *Objekt* **window** *reprezentuje okno prohlížeče, jeho vlastnosti a metody, objekt* **document** *pak vlastní webovou stránku. Podrobně budou vysvětleny všechny objekty a jejich metody ve čtvrté části této knihy, nebudu tedy zatím zabíhat do podrobností.*

# 6.2 Kalkulačky

JavaScript se dá využít i k naprogramování jednoduché (jdou ale i daleko složitější) webové kalkulačky, které může na vlastní stránce sloužit k mnoha různým účelům. Referenční příručka o JavaScriptu Davida Flanagana jde dokonce ještě dál; autor zde ukazuje zdrojový kód finančního kalkulátoru, který vypočítá vaše daně; vy už jen vyplníte daňové přiznání. Z toho je patrné, že JavaScript je plnohodnotným programovacím jazykem a jeho omezení vyplývají z omezení webu.

Následující dva příklady ukazují dvě rozdílné kalkulačky. Jednu úplně jednoduchou, do níž zadáte z klávesnice výraz a program spočítá výsledek, druhá už má dokonce tlačítka (prvky formuláře) a trochu připomíná program *Kalkulačka* z Windows. Navíc jsou obě kalkulačky jedním z možných využití formulářů, i když jde o ty jednodušší.

## 6.2.1 Jednoduchá kalkulačka

Tato opravdu primitivní kalkulačka je formulářem, do jehož jednoho políčka s názvem „Výraz“ zadáte to, co chcete spočítat např.: „52*33“, a po klepnutí na tlačítko *Vypočítat* se v políčku „Výsledek“ objeví výsledek operace.

```
<HTML>
<HEAD>
<TITLE>Jednoduchá kalkulačka</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
```

```
  function pocitej(form) {
     form.vysledek.value=eval(form.expr.value); }
-->
</SCRIPT>
<FORM>
<TABLE BORDER="3" WITDH="350">
<TR>
  <TH ALIGN="right">Výraz:</TH>
  <TD ALIGN="center">
  <INPUT TYPE="text" SIZE="15" NAME="expr"></TD>
  <TD ALIGN="center" ROWSPAN="2">
  <INPUT TYPE="button"
      VALUE=" Vypočítat " ONCLICK="pocitej(this.form)"></TD>
</TR>
<TR>
  <TH ALIGN="right">Výsledek:</TH>
  <TD ALIGN="center">
  <INPUT TYPE="text" SIZE="15" NAME="vysledek"></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```


Jednoduchá kalkulačka

Jak tato kalkulačka funguje? Jakmile zadáte výraz, který chcete spočítat a klepnete na tlačítko Vypočítat, bude spuštěna funkce pocitej (událost **onclick**). Té je předán jako argument objekt aktuálního formuláře (*this.form*). Z políčka, kde umístěn výraz, který se má spočítat (políčko se jmenuje „expr“, jeho hodnota je tedy *obj.expr.value*), se přečte jeho hodnota a ta se jako argument předá metodě **eval** objektu **object**. Ta má jediný úkol: provést příkaz, který je jí předán. V tomto případě je to tedy výraz, metoda **eval** jej tedy spočítá. Výsledek pak bude umístěn jako nová hodnota objektu *obj.vysledek.value*, a tedy políčka s názvem „vysledek“. A to je celé. Prosté, že?

**Poznámka:** *Metoda* **eval** *je metodou objektu* **Object***, lze ji tedy vyvolat prostřednictvím jakéhokoli typu objektu JavaScriptu. Zde je objekt zastoupen objektem formuláře.* **eval(kód)** *provádí kód JavaScriptu zadaný ve svém argumentu* kód. *Kód může obsahovat jeden nebo více příkazů JavaScriptu. Pokud je příkazů více, musí být navzájem odděleny středníky.*

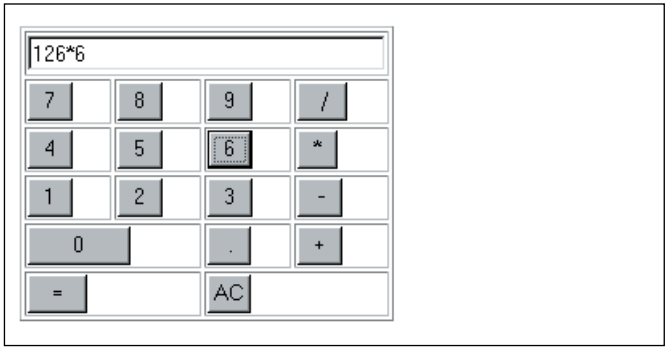
*Výrazy JavaScriptu jsou samy jednoduchým typem příkazu, takže* **eval** *lze použít namísto vykonání příkazů také k vyhodnocení výrazu JavaScriptu – to jste ostatně viděli v tomto příkladu.* **eval** *vrací hodnotu posledního vyhodnoceného příkazu v* kód. *Pokud* kód *obsahuje pouze příkazy, které nemají žádnou hodnotu, nevrací* **eval** *nic.*

## 6.2.2 Tlačítková kalkulačka

Vylepšením předchozí kalkulačky je přidání tlačítkového obalu; jednotlivá tlačítka reprezentují prvky formuláře. Díky formuláři lze také „vyčarovat" skutečnou podobu běžné kalkulačky.

```
<HTML>
<HEAD>
<TITLE>Tlačítková kalkulačka</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
  var jedna = '1'; var dva = '2';
  var tri = '3'; var ctyri = '4';
  var pet = '5'; var sest = '6';
  var sedm = '7'; var osm = '8';
  var devet = '9'; var nula = '0';
  var plus = '+'; var minus = '-';
  var krat = '*';  var lomitko = '/';
  var tecka = '.';
  function pocitej(obj)
     { obj.expr.value = eval(obj.expr.value); }
  function zapis(obj, string)
     { obj.expr.value += string; }
  function vymaz(obj)
     { obj.expr.value = ""; }
-->
</SCRIPT>
</HEAD>
<BODY>
  <FORM NAME="calculator">
  <TABLE BORDER="1">
  <TR>
    <TD COLSPAN="4"><INPUT TYPE="text" NAME="expr" SIZE="30"
                          ACTION="pocitej(this.form)"></TD>
  </TR>
  <TR>
    <TD><INPUT TYPE="button" VALUE="  7  "
            ONCLICK="zapis(this.form, sedm)"></TD>
    <TD><INPUT TYPE="button" VALUE="  8  "
```

```
                ONCLICK="zapis(this.form, osm)"></TD>
    <TD><INPUT TYPE="button" VALUE="  9  "
                ONCLICK="zapis(this.form, devet)"></TD>
    <TD><INPUT TYPE="button" VALUE="  /  "
                ONCLICK="zapis(this.form, lomitko)"></TD>
  </TR>
  <TR>
    <TD><INPUT TYPE="button" VALUE="  4  "
                ONCLICK="zapis(this.form, ctyri)"></TD>
    <TD><INPUT TYPE="button" value="  5  "
                ONCLICK="zapis(this.form, pet)"></TD>
    <TD><INPUT TYPE="button" VALUE="  6  "
                ONCLICK="zapis(this.form, sest)"></TD>
    <TD><INPUT TYPE="button" VALUE="  *  "
                ONCLICK="zapis(this.form, krat)"></TD>
  </TR>
  <TR>
    <TD><INPUT TYPE="button" VALUE="  1  "
                ONCLICK="zapis(this.form, jedna)"></TD>
    <TD><INPUT TYPE="button" VALUE="  2  "
                ONCLICK="zapis(this.form, dva)"></TD>
    <TD><INPUT TYPE="button" VALUE="  3  "
                ONCLICK="zapis(this.form, tri)"></TD>
    <TD><INPUT TYPE="button" VALUE="  -  "
                ONCLICK="zapis(this.form, minus)"></TD>
  </TR>
  <TR>
    <TD COLSPAN="2"><INPUT TYPE="button" VALUE="      0      "
                           ONCLICK="zapis(this.form, nula)"></TD>
    <TD><INPUT TYPE="button" VALUE="  .  "
                ONCLICK="zapis(this.form, tecka)"></TD>
    <TD><INPUT TYPE="button" VALUE="  +  "
                ONCLICK="zapis(this.form, plus)"></TD>
  </TR>
  <TR>
    <TD COLSPAN="2"><INPUT TYPE="button" VALUE="  =  "
                           ONCLICK="pocitej(this.form)"></TD>
    <TD COLSPAN="2"><INPUT TYPE="button" VALUE="AC" SIZE="3"
                           ONCLICK="vymaz(this.form)"></TD>
  </TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```




Kalkulačka s formulářovými tlačítky

Tato kalkulačka pracuje na stejném principu jako ta předchozí, pouze jsou použita přímo tlačítka formuláře, takže kalkulačka vypadá jako opravdová. Výraz, který se má spočítat, zadáváte těmito tlačítky; zobrazuje se v okénku formuláře. Jakmile je zadán, klepněte na tlačítko „=“, výraz bude nahrazen jeho hodnotou, výsledkem. Vše funguje opět přes objekt formuláře. Pokud klepnete na tlačítko s výrazem, ten je přičten k aktuální hodnotě políčka: *obj.expr.value* += *string*. Pokud je stisknuto tlačítko „AC“, je obsah políčka vymazán: *obj.expr.value* = *""*. A konečně, pokud je stisknuto tlačítko „=“, bude výraz spočten, stejně jako v předchozím případě, jen se výsledek neumístí do jiného políčka, ale do toho samého: *obj.expr.value* = *eval(obj.expr.value)*. A to je vše. Opět prosté, že? Kousky v JavaScriptu jsou opravdu krátké, největší kus kódu totiž zabírá vykresleni kalkulačky.

# 6.3 Datum a čas

Zobrazení data a času na webové stránce, zvláště, pokud se automaticky aktualizuje, je velice efektní a říká všem, co na stránku zavítají, že autor prostě umí. Tyto tři malé programy vám ukáží, co všechno lze s datem a časem dělat; možností je však daleko více. To už bych se ale musel ponořit hlouběji do práce s objekty a událostmi, které však budou předmětem poslední části této knihy.

## 6.3.1 Čas běžící ve formulářovém políčku

Tento příklad je spíše taková legrácka, slouží k pobavení a k důkazu, že v políčku formuláře lze zobrazit prakticky cokoliv. Tento program vypisuje v textovém políčku aktuální čas, který je neustále aktualizován, takže stále vidíte, kolik je hodin. K aktualizaci se zde využívá metody **setTimeout** objektu **window**, o níž sem psal u příkladu načtení nové stránky do prohlížeče po odeslání formuláře. Zde se tedy díky této metodě spouští funkce generující čas v políčku formuláře každých tisíc milisekund, tedy každou sekundu.

```
<HTML>
<HEAD>
<TITLE>Hodiny ve formuláři</TITLE>
```

```
<SCRIPT>
<!--
function runClock() {
  today   = new Date();
  hours   = today.getHours();
  minutes = today.getMinutes();
  seconds = today.getSeconds();
  timeValue = hours;
  timeValue += ((minutes < 10) ? ":0" : ":") + minutes;
  timeValue += ((seconds < 10) ? ":0" : ":") + seconds;
  document.clock.time.value = timeValue;
  timerID = window.setTimeout("runClock()",1000);
  timerRunning = true;
}
// -->
</SCRIPT>
</HEAD>
<BODY ONLOAD="runClock()">
<FORM NAME="clock" ONSUBMIT="0">
<INPUT TYPE="text" NAME="time" SIZE="10" VALUE="">
</FORM>
</BODY>
</HTML>
```

## 6.3.2 Čas běžící ve stavové řádku prohlížeče

Tento program pracuje velice jednoduše. Při spuštění (událost **onload**) se spustí funkce *show_date_time*, která zjistí aktuální datum a čas přečtěním z objektu **Date** a přiřadí je vlastnosti **status** objektu **window**, a tedy stavovému řádku. Metoda **setTimeout** opět zařídí, aby byla funkce spuštěna každou sekundu, čímž dojde k aktualizaci času. Čas je pro stručnost programu vypisován pouze tak, jak je přečten z objektu **Date**, lze ho samozřejmě libovolným způsobem formátovat, než se zapíše do stavového řádku. Třeba tak, jak jste mohli vidět v předchozím příkladu.

```
<HTML>
<HEAD>
<TITLE>Čas ve stavovém řádku</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
  function show_date_time()
  {
  var now=new Date();
  window.status=now;
  window.setTimeout("show_date_time();",1000);
```

```
  }
// -->
</SCRIPT>
</HEAD>
<BODY ONLOAD="show_date_time();return true;">
</BODY>
</HTML>
```

### 6.3.3 Pozdrav v závislosti na denní době

Tento program vypíše na webové stránce v závislosti na denní době předem určený pozdrav. Samozřejmě intervaly definující daný pozdrav mohou být libovolné, stejně jako text, který se zobrazí. To už záleží na vás a na vaší modifikaci zdrojového kódu programu. Navíc tento program demonstruje vícenásobnou podmínku a využití příkazu **else**. První příkaz **if** až po vypsání „Dobrou noc" je vlastně jeden jediný příkaz, který má ve svém těle další podmínku **if**, která se provede pouze tehdy, pokud ta předtím nebyla pravdivá.

```
<HTML>
<HEAD>
<HEAD>Pozdrav</HEAD>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
  dnes = new Date();
  hodina = dnes.getHours();
  minuta = dnes.getMinutes();
  if (minuta > 30) ++hodina;
  if((hodina >6) && (hodina <=8))document.write("Dobré ráno")
  else if((hodina >8) && (hodina <=11))
            document.write("Dobré dopoledne")
  else if((hodina >11) && (hodina <=13))
            document.write("Dobré poledne")
  else if((hodina >13) && (hodina <=17))
            document.write("Dobré odpoledne")
  else if((hodina >17) && (hodina <=22))
            document.write("Dobrý večer")
  else document.write("Dobrou noc");
// -->
</SCRIPT>
</BODY>
</HTML>
```

Když už teď umíte vytvářet jednoduché programy v JavaScriptu, můžete přejít na nejzajímavější část této knihy, alespoň z mého pohledu: dynamické HTML a tvorbu opravdu živých stránek. Seznámíte se důležitými objekty, jejich vlastnostmi a metodami a obsluhou nejrůznějších událostí. S kousky této kuchyně jste se již naučili vařit v této kapitole, aniž jste věděli, co vlastně používáte a k čemu všemu se to dá využití. Znáte některé objekty a umíte reagovat na některé události jednoduchými funkcemi. Teď se konečně dozvíte, proč to tak vlastně všechno je a jak to funguje.

**Tip:** *Pokud se chcete opravdu dobře naučit programovat v JavaScriptu, nesmíte si ujít publikaci Davida Flanagana,* JavaScript – kompletní průvodce*, která vyšla v nakladatelství Computer Press a web* www.javascript.sk*, na němž najdete desítky řešených situací a zajímavých programů a her v JavaScriptu. Ostatně všechny příklady z této kapitoly mají svůj původ právě tam.*

# Část čtvrtá: DHTML a objektové programování

**Objekty prohlížeče a jejich využití**

**Události a jejich obsluha**

**Příklady využití DHTML**

Dynamické HTML je velice zajímavá úroveň tvorby webových stránek. Umožňuje autorovi tvořit dynamický obsah, který není statický, jako kdyby jej vytvořil pomocí standardních značek. Může totiž svůj obsah a svoji formu měnit, a to v závislosti na tom, co uživatel dělá. Dá se tak dosáhnout toho, že se mění barvy písma, že objekty na obrazovce mění svoji polohu že obrázky nejsou statickými prvky, ale mění se při přejetí kurzorem myši, dokonce i text může měnit obsah. Stránka se tedy stává opravdu živou, protože dynamické HTML umožňuje ovládat (prostřednictvím skriptů) v podstatě každý viditelný prvek na webové stránce.

V předchozích třech částech této knihy jsem se zabýval vlastně pouze statickou tváří webové stránky, pominu-li několik příkladů na konci třetí části (některé z nich již dynamického HTML využívaly). Vždy jsme mohli ovlivnit pouze stránku do doby, než se načetla do prohlížeče. Jedinou takovou oživovací metodou byly styly, které jsem však vzal trošku zkrátka; nejsou hlavním obsahem této knihy. Dynamické HTML na druhou stranu umožňuje provádět změny ve stránce i poté, co již byla zobrazena v prohlížeči; a to je ta největší a hlavní výhoda DHTML.

Ač jsem to už několikrát napsal v předchozích částech, velice rád zopakuji: Základní prvky, kterými lze chování stránky ovlivňovat, je objektový model prohlížeče, události prohlížeče a styly. A těm se budu věnovat v následujících kapitolách.

# 1. Objekty prohlížeče a jejich využití

JavaScript, kterému jsem se dosud věnoval jen jaksi bez udání účelu, má jednu výbornou vlastnost (ostatně jako i další skriptovací jazyk, VBScript). Umožňuje totiž ovládat samotný prohlížeč, a to prostřednictvím jeho objektů. Tyto objekty vkládají tvůrci (a tedy vám) do ruky velice silný vývojový nástroj. Pomocí nich lze totiž dosáhnout prakticky čehokoliv, na co si vzpomenete.

S některými objekty jsem vás už seznámil v minulých částech, zejména v té o JavaScriptu. Používal jsem se však bez ladu a skladu, pouze jsem řekl, že jsou a že je lze využít určitým způsobem. V této kapitole je však setřídím a popíšu daleko lépe; ukážu jejich konkrétní využití a nastíním jejich možnosti. Postupně se tímto objektovým modelem dostanu k některým velice zajímavým efektům.

Co vlastně objekt je? Krátce jsem se o tom zmínil už v části o JavaScriptu v kapitole týkající se proměnných. Objekt je soubor pojmenovaných dat a funkcí, na které je odkazováno jako na vlastnosti tohoto objektu. Na vlastnost objektu se odkazuje obecně takto:

```
objekt.vlastnost
```

Příkladem může být třeba šířka a výška obrázku:

```
image.width
image.height
```

Vlastnosti objektů se v mnoha ohledech chovají jako proměnné JavaScriptu a mohou obsahovat jakýkoli typ dat, včetně polí, funkcí a jiných objektů; ostatně uvidíte sami dále.

Kromě vlastnosti má objekt také definovány určité funkce, které s tímto objektem nějakým způsobem pracují a mohou k tomu využívat i jeho vlastností. Tato funkce se nazývá *metoda* a má svoje jméno a argumenty, které jsou jí předávány. Metoda se používá obecně takto:

```
objekt.metoda()
```

Pokud chcete např. použít již známou metodu **write** objektu **document**, použijete tento zápis:

```
document.write("Text vypsaný metodou write.");
```

Objekty prohlížeče jsou propojeny navzájem do objektového modelu. Objektový model je něco jako strom, který popisuje, jakým způsobem jsou tyto objekty spojeny dohromady. Začíná kmenem, kde stojí objekt **windows**, a odkazuje na svoje jednotlivé větve (**frames**, **do-cument** apod. U objektového modelu jsou navíc objekty programovatelné – můžete nastavovat jejich vlastnosti, volat konkrétní metody nebo ovládat je událostními funkcemi.

V každém objektovém modelu obvykle bývá jeden objekt na nejvyšší úrovni. V prohlížeči je tímto objektem okno prohlížeče, objekt **window**. Proto se na objektový model prohlížeče lze dívat také jako na objekt okna prohlížeče, v němž jsou obsaženy všechny ostatní objekty prohlížeče. Každý z těchto objektů má svoje další objekty, vlastnosti a metody, některé vlastnosti jsou dokonce jen odkazy na jiné objekty, takže výsledná hierarchie je složitá a propletená.

Objekt **window** zastřešuje tyto hlavní objekty:

- **history** – historie navštívených stránek;
- **navigator** – uchovává informace o prohlížeči;
- **location** – uchovává aktuální adresu webové stránky;
- **frames** – rámy, které jsou obsaženy v aktuálním okně;
- **document** – vlastní obsah stránky aktuálního okna;
- **event** – obsahuje informace o aktuální události.

O jednotlivých součástech objektového modelu, které jsou zde uvedeny, bude tato kapitola.

# 1.1 Objekt *window* – okno prohlížeče

Jak už bylo řečeno, je objekt **window** hlavním a nadřazeným objektem všech objektů ostatních; reprezentuje okno prohlížeče. S některými jeho metodami a vlastnostmi jste se už seznámili, např. s metodou **alert**.

Díky tomu, že je objekt **window** objektem hlavním, není třeba jej explicitně uvádět. Prohlížeč totiž ví, že všechny objekty, vlastnosti a metody jsou jeho součástí. Můžete tak klidně volat metodu **alert**, aniž byste psali

```
window.alert("Text vyspaný metodou alert.")
```

Stejně tak budete-li odkazovat na vlastnost **history**, která reprezentuje objekt **history** v rámci objektu **window**, je prohlížeči jasné, že odkazujete na vlastnost **window.history**. Objekt **window** je totiž ústřední, takže většina výrazů JavaScriptu je vyhodnocena v kontextu tohoto objektu. Stejně tak jste se již setkali s metodou **write** objektu **document**; pokud chcete cokoliv vypsat na stránku, použijete této metody:

```
document.write("Text vypsaný metodou write.");
```

ve skutečnosti je celá syntaxe takováto:

```
window.document.write("Text vypsaný metodou write.");
```

Objekt **window** má totiž definovánu vlastnost **document**, která na objekt **document** ukazuje. Konečně objekt **window** v sobě má jako vlastnosti umístěny odkazy na všechny ostatní zajímavé objekty, které jsou uvedeny na počátku této kapitoly. Těchto vlastností, objektů a různých metod je však tolik, že by jejich popis vydal na několikasetstránkovou knihu; budu se tedy zabývat těmi nejzajímavějšími a nejdůležitějšími.

***Základní vlastnosti objektu window:***

- **document** – Odkaz na objekt **document**, který je součástí aktuálního okna.
- **frames[]** – Pole rámů, které obsahuje aktuální stránka.
- **history** – Odkaz na objekt **history** pro toto okno, a tedy seznam navštívených stránek z tohoto okna.
- **length** – Počet prvků v poli **frames[]**, totéž jako **frames.length**, uchovává počet rámů, které jsou obsaženy v aktuálním okně.
- **location** – Odkaz na objekt **location** pro toto okno, který uchovává informace o aktuálním URL stránky.
- **Math** – Odkaz na objekt, který uchovává konstanty a počítá matematické funkce.
- **name** – Název aktuálního okna.
- **navigator** – Odkaz na objekt **navigator**, který uchovává informace o prohlížeči.
- **status** – Řetězec, v němž je obsažen aktuální obsah stavového řádku.
- **window** – Odkaz na okno samotné.

***Důležité metody objekt window:***

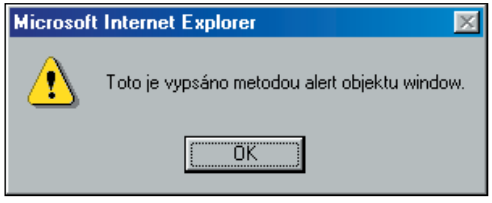
- **alert() –** Zobrazí v dialogovém okénku jednoduchou zprávu.
- **clearTimeout()** – Ruší nevyřízené operace časovače (timeout).
- **close()** – Zavírá aktuální okno.
- **confirm()** – V dialogovém okénku pokládá dotaz typu *ano/ne*.
- **open()** – Vytváří a otevírá nové okno prohlížeče.
- **prompt()** – V dialogovém okénku očekává nějaký vstup.
- **setTimeout()** – Poté, co uplyne určený čas, provede zadaný kód.

## 1.1.1 Zpráva v dialogovém okně – metoda *alert*

Metoda alert slouží pro vypsání uživatelem definované zprávy v dialogové okénku. Toto dialogové okénko obsahuje jediné tlačítko, *OK*, po jehož odklepnutí se dialogové okénko zavře. Okénko je výstražného charakteru, který reprezentuje vykřičník, takže se hodí zejména k zobrazení chybových zpráv, když je např. uživatelský vstup do některého prvku formuláře nějakým způsobem neplatný. Dialog *alert* může uživatele informovat o problému a vysvětlit, co má být opraveno, aby se do budoucna zabránilo problémům. Vzhled dialogového okénka *alert* závisí na platformě, ale zpravidla obsahuje grafiku, která indikuje, že zpráva upozorňuje na nějakou chybu nebo dává výstrahu.

Obecná syntaxe je

```
window.alert("Toto je vypsáno metodou alert objektu window.");
```




Dialogové okno metody *alert*

Zpráva (text) zobrazená v dialogovém okénku je řetězec jednoduchého textu, nikoli formátovaného HTML, jako je tomu u metody **document.write**. Lze však použít nového řádku, znaku „\n“, pro rozdělení zprávy na několik řádků. Určité zkladní formátování lze také provádět pomocí mezer, výsledky však budou záviset na fontu, který bude v dialogu použit, a tudíž na systému i prohlížeči samotném.

Samozřejmě používat u specifikace metody **alert** také odkazu na objekt **window** je zbytečné, stejně jako u všech vlastností a metod tohoto objektu se odkaz na **window** uvádět nemusí. Lze tak klidně psát:

```
alert("Toto je vypsáno metodou alert objektu window.");
```

a výsledek bude naprosto totožný.

## 1.1.2 Očekávání vstupu od uživatele – metoda *prompt*

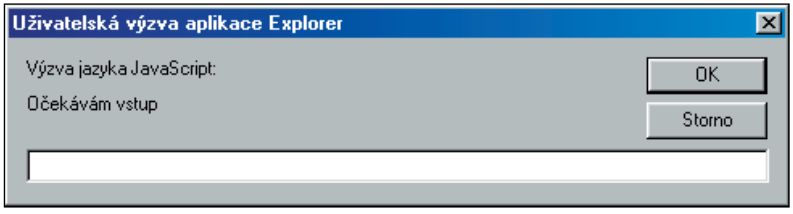
Metoda **prompt** se používá k získání nějakého údaje od uživatele pomocí dialogového okna. Zatímco **alert** pouze zobrazí nějaký text, **prompt** zobrazí text, co po uživateli chce, a textové políčko, kam uživatel zadá odpověď. Zobrazen bude dialog s tlačítky *OK* a *Storno*, kde tlačítko *OK* vstup potvrdí a *Storno* zruší.

Syntaxe metody je

```
window.prompt(text)
window.prompt(text, retezec)
```

kde *text* je jednoduchý text, který se zobrazí v tomto dialogu jako požadavek na vstup, a tedy žádá uživatele, aby zadal určitou informaci, *retezec* pak specifikuje implicitní text, který bude zobrazen ještě předtím, než uživatel cokoliv zadá. Jestliže tento parametr není nijak specifikován, pak metoda **prompt** zobrazí jako implicitní hodnotu řetězec „<undefined>“. Proto používejte prázdný řetězec, aby **prompt** zobrazil toto okénko prázdné:

```
window.prompt("Očekávám vstup", "");
```




Vstupní okno metody *prompt*

Výstupem této metody je pak řetězec, který specifikuje text, který uživatel do dialogu zadal, popř. *null*, pokud uživatel klepnul na tlačítko *Storno*. Tento výstup se tak přiřadí do nějaké proměnné:

```
vstup = window.prompt("Očekávám vstup", "");
```

Na rozdíl od zprávy obsažené v dialogu metody **alert** může být zpráva metody **prompt** pouze jednořádková, což někdy může komplikovat situaci. Dialog je navíc zbytečně dlouhý a nepřizpůsobuje se aktuální velikosti zprávy požadující vstup, nelze ani jinak ovlivnit jeho velikost.

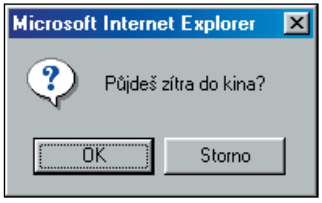
## 1.1.3 Jednoduchý dotaz – metoda *confirm*

Prohlížeč umožňuje kromě dialogu **prompt**, kde je očekáván nějaký text, ptát se uživatele pouze na otázku, na níž je odpověď typu ano/ne. V tomto případě je takovýto dialog zobrazen s tlačítky *OK* a *Storno*, kdy *OK* znamená ano a *Storno* ne. Syntaxe metody **confirm** je podobná metodě **alert**:

```
window.confirm("Půjdeš zítra do kina?");
```

nebo jen

```
confirm("Půjdeš zítra do kina?");
```




Otázka metody *confirm*

Jestliže uživatel klepne na tlačítko *OK*, pak **confirm** vrátí hodnotu *true*, pokud klepne na tlačítko *Cancel*, pak **confirm** vrátí *false*. Pro přečtení této hodnoty je dobré opět použít nějaké proměnné a na základě její hodnoty pak odpověď vyhodnotit, např. podmínkou.

Stejně jako u **alert** a **prompt** je jako argument metody předáván jednoduchý textový řetězec. Lze použít odřádkování jako u metody **alert**, a sice znakem „\n“. Lze také provádět určité základní formátování za pomoci mezer, ale stejně jako u **alert** budou výsledky záviset na použitém systému.

**Poznámka:** *Neexistuje způsob, jak změnit popisky odpovídajících tlačítek. (třeba „Ano/Ne“). Proto byste měli otázku koncipovat tak, aby „OK“ nebo „Storno“ bylo vhodnou odpovědí.*

## 1.1.4 Otevírání a zavírání okna prohlížeče – metody *open* a *close*

V JavaScriptu můžete pomocí objektu **window** otevřít nové okno a načíst do něj danou stránku. To se provádí pomocí metody **open**, jehož syntaxe je takováto:

```
window.open(adresa, jméno, [vlastnosti]);
```

Adresa zde specifikuje URL, na němž leží dokument, který chcete v novém okně otevřít. Může jít jak o lokální soubor, tak soubor na jakémkoliv světové serveru. Pokud je tato adresa prázdný řetězec, otevře se pouze prázdné okno.

Jméno je název okna, na který pak lze odkazovat pomocí parametru **TARGET** značky **A** nebo **FORM**. Pokud zadáte u parametru **TARGET** jako hodnotu jméno tohoto okna, bude toto okno načteno právě sem.

Parametr vlastnosti určuje, jaké vlastnosti bude nově otevřené okno mít. Tento parametr je volitelný a reprezentuje jej řetězec, který obsahuje čárkami oddělené jednotlivé vlastnosti okna. Např.:

```
window.open("index.html","Jmeno","width=795,height=540,toolbar=no,
directories=no,status=no,scrollbars=no,resize=no,menubar=no,");
```

tento řádek JavaScriptu otevře okno o velikosti 795x540 bodů, do něhož načte soubor „index.html“. Zároveň zakáže použití panelu nástrojů (toolbar), stavového řádku (status), odkazů (directories), posuvníků (scrollbars), hlavního panelu (menubar) a konečně i možnost zvětšit či zmenšit okno.

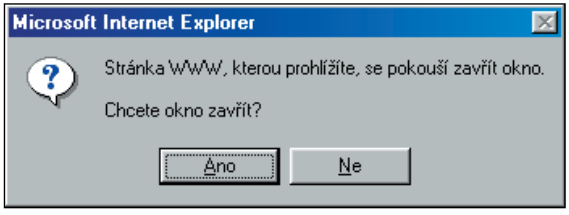
Z výčtu těchto parametrů je jasné, jak může takový řetězec vlastností vypadat. Je zřejmé, že pokud chcete nějakou vlastnost zapnout, stačí *no* nahradit *yes* nebo daný parametr vypustit.


Okno prohlížeče bez prvků, které bylu odstraněny při otvírání metodou *open*.

Takovéto okno, které jste otevřeli metodou **open**, lze i zavřít, a to metodou **close**. Má jednoduchou syntaxi:

```
window.close();
```

a zavírá aktuální okno. Pokud je toto okno jediné otevřené, zavře se i sám prohlížeč. Pokud však ukončujete okno, zobrazí prohlížeč zprávu, že se stránka pokouší toto okno zavřít; vy můžete buďto zavření potvrdit nebo stornovat.




Při pokusu o zavření okna se Internet Explorer zeptá.

## 1.1.5 Časování operací – metody *setTimeout*, *clearTimeout*

Velice užitečnou metodou objektu **window** je tzv. časovač událostí, metoda **setTimeout**. Její využití jste již viděli v minulé části u příkladů jednoduchých časových programů:

```
function runClock() {
  today   = new Date();
  hours   = today.getHours();
  minutes = today.getMinutes();
  seconds = today.getSeconds();
  timeValue = hours;
  timeValue += ((minutes < 10) ? ":0" : ":") + minutes;
  timeValue += ((seconds < 10) ? ":0" : ":") + seconds;
  document.clock.time.value = timeValue;
  window.setTimeout("runClock()",1000);
}
```

Tato funkce, pokud se jedou spustí, zjistí aktuální čas, zformátuje jej do čitelné podoby a umístí do políčka formuláře *clock*, které se jmenuje *time*. Potom se metodou **setTimeout** nastaví její další spuštění za jednu sekundu a tak pořád dokola. Obecná syntaxe zní:

```
window.setTimeout(kód, čas);
```

Obecně metoda **setTimeout** odkládá provedení příkazů JavaScriptu, které jsou specifikovány v řetězci kód, v našem případě spuštění funkce *runClock*, o zpoždění dané argumentem *čas* v milisekundách. Když specifikovaný počet milisekund uplyne, jsou tyto příkazy provedeny, ale pouze jednou. Pro opakované provádění musí řetězec kód obsahovat opětovné volání metody **setTimeout**, aby se zase nastavil čas, za který se opakovaný běh příkazů uskuteční; ostatně to je patrné také z našeho příkladu.

Metoda **setTimeout** má i svoji návratovou hodnotu, která se používá pro zrušení časového provedení aktuální operace metodou **clearTimeout**. Pro lepší pochopení příklad:

Pokud kdekoliv v programu uvedeme tento řádek:

```
provest = window.setTimeout(kód, čas);
```

získá proměnná *provest* hodnotu právě nastaveného opožděného provedení určitých příkazů. Pokud kdykoliv před provedením těchto příkazů narazí JavaScript na řádek

```
window.clearTimeout(provest);
```

bude toto opožděné provedení zrušeno. Obecně tedy metoda **clearTimeout** ruší provádění kódu, který byl odložen metodou **setTimeout**. Argument *provest* je hodnota vrácená voláním **setTimeout** a identifikuje, který z bloků odloženého kódu bude zrušen.

## 1.2 Objekt *document*

Objekt **document** je po objektu **window** základním objektem, který nesmíme opomenout. Jeho obsahem je celá stránka zobrazená v okně; objekt **document** získává nové vlastnosti

a jejich hodnoty ze zdrojového kódu HTML. Kdykoliv se např. ve stránce vyskytne formulář, zobrazí se v objektu **document** nový objekt typu **form**. Objeví-li se zde odkaz, v objektu **document** přibude nový objekt typu **link** apod.

Díky tomu lze bez problému ovlivňovat všechno, co se v dokumentu nachází pomocí těchto vlastností, které ukazují na prvky (objekty) tohoto dokumentu. Ty poskytují podrobnosti o mnoha aspektech dokumentu od barev textu, pozadí, odkazů až k datu, kdy byl dokument naposledy upraven. Objekt **document** obsahuje také několik polí, popisujících obsah dokumentu, např. pole odkazů (**links**), v němž každý objekt reprezentuje jedno hypertextové spojení v dokumentu. Existuje třeba pole **forms[]**, kdy každý prvek pole obsahuje jeden objekt typu **form** pro každý formulář obsažený v dokumentu.

Na objekt **document** odkazujeme

```
window.document
```

nebo jen (pokud se na **document** odvoláváte z aktivního okna)

```
document
```

Mnoho vlastností objektu **document** vzniká dynamicky při zobrazení stránky; vždy jsou však určitého typu. Např. definicí formuláře vznikne objekt s určitým názvem (totožným s názvem formuláře), který je typu **form**, a tedy typu formulář. Objekt **document** má však i další vlastnosti, které se využívají přímo, nikoliv jako odkazy na zástupce, jako je tomu třeba v případě formulářů nebo proměnných typu řetězec, na které odkazujeme pomocí objektu **string**.

***Vlastnosti objektu document:***

- **alinkColor** – Barva odkazu, na který jste právě klepnuli. Původně se nastavuje pomocí parametru **ALINK** značky **BODY**.
- **all** – Sbírka všech objektů webového dokumentu. Zjednodušeně řečeno, jakýkoliv prvek na stránce se zde zobrazí jako objekt.
- **bgColor** – Barva pozadí dokumentu, původně se nastavuje parametrem **BGCOLOR** ve značce **BODY**.
- **fgColor** – Barva textu dokumentu. Původně se nastavuje parametrem **TEXT** ve značce **BODY**.
- **linkColor** – Barva nenavštívených odkazů v dokumentu. Původně se nastavuje parametrem **LINK** ve značce **BODY**.
- **location** – Obsahuje aktuální adresu (URL) dokumentu, tuto vlastnost již použil skript zpracovávající formulář v příkladu v kapitole 6.1.2.
- **referrer** – Tato vlastnost udává adresu (URL), odkud byl aktuální dokument načten, dá se tak zjistit, z jaké stránky se na tuto stránku uživatel dostal. Lze využít ke zjištění, odkud nejčastěji na tuto stránku uživatelé chodí.
- **title** – Vlastnost zjišťující jméno dokumentu, které je definováno značkou **TITLE** daného dokumentu. Tuto vlastnost již použil skript zpracovávající formulář v příkladu v kapitole 6.1.2.

- **vlinkColor** – Barva již navštívených odkazů. Původně se nastavuje parametrem **VLINK** ve značce **BODY**.

***Základní metoda objektu document:***

- **write() –** Vkládá určený řetězec nebo řetězce do právě zobrazovaného dokumentu.

O metodě **write** jsem již psal několikrát a mnohokrát byla také použita v příkladech. Do aktuálního dokumentu vepisuje dynamicky generovaný text, který může obsahovat jakékoliv značky HTML.

Co se barev dokumentu týče, ani zde si nemyslím, že je nutný další delší výklad. Změna barvy z JavaScriptu změní barvu všech prvků, na něž má vliv, a to kdykoliv během zobrazení stránky. Samotné nastavení těchto barev bylo popsáno v první části věnující se základům HTML.

Zajímavým objektem definovaným v objektu **document** je **all**. Tento objekt sdružuje všechny prvky ve stránce, a tedy jsou podmnožinou jakékoliv formuláře, nadpisy, odstavce, obrázky, rámy apod. Tzn. definujete-li políčko formuláře s názvem „Tlacitko“, definujete nejen novou vlastnost aktuálního objektu formuláře (obecně **form**), ale zároveň také vlastnost objektu **all**, a tedy **all.Tlacitko**.

# 1.3 Formulář – objekt *form*

Objekt form je součástí objektu dokument a je v něm uložen formulář, který uživatel vytvoří pomocí značky **FORM**. Tento objekt vzniká v okamžiku definice formuláře a je pojmenován jménem, které definuje uživatel. Pokud je vytvořen formulář

```
<FORM NAME="Anketa">
```

bude zároveň vytvořen objekt typu **form**, který se bude jmenovat *Anketa*. Tohoto jsme již využili v příkladu zpracování dat z odesílaného formuláře v kapitole 6 předchozí části. Samozřejmě objektu **form** se nevyužívá jen pro formátování těchto dat, ale také pro ověření, zda je formulář vyplněn správně, zda v některých polích nejsou hodnoty, které tam nemají co dělat. Podle toho pak povolí odeslání formuláře nebo požádá uživatele o doplnění chybějících nebo špatných údajů.

Na jednotlivé prvky formuláře se odkazujeme takto:

```
document.form.prvek
```

kde **form** reprezentuje jméno formuláře. V našem příkladě z kapitoly 6.1.2 byl uveden tento kód:

```
var poslat = document.Formular;
data += "Jméno: "+poslat.Jmeno.value + "\n";
```

což umožní získat z formuláře, který se jednoduše jmenuje *Formular*, hodnotu políčka s názvem *Jmeno*. Na hodnotu políčka se odkazuje pomocí vlastnosti **value**.

***Důležité vlastnosti objektu form:***

- **action** – Řetězec, který je určen parametrem **ACTION**, definuje URL nebo e-mailovou adresu, na kterou, má být formulář odeslán. Lze jej libovolně JavaScriptem měnit, stejně jako každou vlastnost formuláře.
- **encoding** – Řetězec, který udává kódovací metodu, která je určena parametrem **ENCTYPE**. Pro odeslání e-mailem bývá tato hodnota *text/plain*.
- **method** – Specifikuje techniku odeslání formuláře. Tato vlastnost je definovaná parametrem **METHOD** a nabývá hodnot *get* nebo *post*.

***Základní metody objektu form:***

- **reset()** – Resetuje všechny vstupní prvky formuláře na jejich počáteční hodnoty, analogie tlačítka typu reset, lze však provádět přímo ze skriptu.
- **submit()** – Odešle daný formulář, analogie tlačítka typu *submit*.

Příklad:

```
<FORM NAME="Anketa"
ACTION="mailto:broza@cpress.cz?subject=Formulář" METHOD="post"
ENCTYPE="text/plain" NAME="Odeslat" ONSUBMIT="Zpracuj(this)">
```

bude vytvořen objekt *Anketa* typu **form** a bude mít tyto vlastnosti:

```
action = "mailto:broza@cpress.cz?subject=Formulář"
method="post"
encoding="text/plain"
```

Pokud v tomto formuláři definujete nějaké prvky (tlačítka, pole apod), např. takto:

```
<INPUT TYPE="text" NAME="Jmeno">
<TEXTAREA NAME="Vzkaz" COLS="50" ROWS="3"></TEXTAREA>
<INPUT TYPE="reset" NAME="Vymaz" VALUE="Vymaž">
```

vytvoříte zároveň vlastnosti objektu *Anketa* s názvy *Jmeno*, *Vzkaz* a *Vymaz*. Na jejich hodnoty odkazujete takto:

```
var formular=document.Anketa;
hodnota1 = formular.Jmeno.value;
hodnota2 = formular.Vzkaz.value;
hodnota3 = formular.Vymaz.value;
```

Samozřejmě lze tyto hodnoty kdekoliv ve skriptu měnit, a tím i ovlivňovat formulář samotný.

# 1.4 Další zajímavé a užitečné objekty

Kromě objektů **window**, **document** a **form** bych chtěl ještě zmínit neméně zajímavé a užitečné objekty, které spadají pod objekt **window**: **navigator**, **location** a **history**.

## 1.4.1 Popis prohlížeče – objekt *navigator*

Objekt navigator obsahuje několik vlastností, které popisují právě používaný webový prohlížeč. Užití jste viděli v příkladu v kapitole 6.1.2, kdy skript zjišťoval, jakou verzí prohlížeče si uživatel stránku prohlíží. To lze využít třeba při statistikách, kdy se zjišťuje, s jakými prohlížeči lidé na stránky chodí, nebo při použití různých verzí stránky zvlášť pro každý prohlížeč (např. jedna verze pro Internet Explorer a druhá pro Netscape Navigator).

Chcete-li např. zjistit název a verzi prohlížeče, použijete tento řádek kódu:

```
info = "Prohlížeč: "+ navigator.appName + navigator.appVersion;
```

Odkaz na objekt **window** není nutné použít, jak je z příkladu patrné.

Kompletní obecná syntaxe je tato:

```
window.navigator.vlastnost
```

***Základní vlastnosti objektu navigator:***

- **appCodeName** – kódový název prohlížeče;
- **appName** – název prohlížeče;
- **appVersion** – informace o verzi prohlížeče.

Jaké hodnoty mají tyto vlastnosti v případě, že je použitý prohlížeč Internet Explorer 5.0 a operační systém Windows 98?

```
document.write("appCodeName = ", navigator.appCodeName, "<BR>");
document.write("appName = ", navigator.appName, "<BR>");
document.write("appVersion = ", navigator.appVersion);
```

Tento programový kód vypíše následující řádky:

```
appCodeName = Mozilla
appName = Microsoft Internet Explorer
appVersion = 4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
```

Je tedy patrné, že bohatě stačí číst hodnotu vlastnosti **appVersion**, z níž vyplyne jak verze prohlížeče, tak verze operačního systému.

## 1.4.2 Adresa stránky – objekt *location*

Objekt location je uložen ve vlastnosti location objektu **window** a reprezentuje webovou adresu („polohu“) dokumentu, který je právě zobrazovan v daném okně. Vlastnost **href** obsahuje kompletní URL tohoto dokumentu a ostatní vlastnosti objektu **location** popisují každá část tohoto URL. Objekt **location** je ve skutečnosti zvláštní formou objektu **URL** – má tytéž vlastnosti jako tento objekt. Na rozdíl od něj je však lze měnit.

Obecná sytaxe je tato:

```
location
window.location
```

Pokud chcete tedy zjistit URL aktuálního dokumentu, stačí, když přečtete přímo obsah objektu **location**; automaticky tak čtete hodnotu vlastnosti **href** (vlastnosti objektu se však příliš nepoužívají, nebudu se tedy jimi zabývat).

Chcete-li tedy vypsat obsah objektu **location**, a tedy adresu aktuální stránky, zadejte:

```
document.write("Aktuální URL je: ", location);
```

a prohlížeč vypíše tuto adresu (samozřejmě záleží na situaci):

Aktuální URL je: http://www.zive.cz

Objekt **location** se však hodí nejen k zjišťování adresy aktuálního dokumentu. Pokud objektu **location** přiřadíte novou adresu, nové URL, automaticky dojde k nahrání webové stránky z této adresy a jejímu zobrazení v prohlížeči. Určitě si pamatujete na příklad přesměrování stránky na jinou adresu po odeslání formuláře (6.1.3). Pro přechod na další stránku byla použita metoda **setTimeout**, v jejímž parametru byla umístěna nová lokace objektu **location**:

```
window.setTimeout("location='index.html'",1);
```

Kdykoliv tedy budete potřebovat přejít na jinou stránku v aktuálním okně, stačí zapsat do zdrojového kódu JavaScriptu:

```
location="index.html"
```

nebo

```
window.location="index.html"
```

kde **window** reprezentuje objekt aktuálního okna prohlížeč. Z toho je patrné, že pokud znáte jméno okna, lze změnit i jeho obsah; **window** zde nahradíte jeho jménem.

## 1.4.3 Navštívené stránky – objekt *history*

Objekt **history** slouží pro pohyb po navštívených stránkách. Pro pohyb po těchto navštívených stránkách slouží tlačítka *Zpět* a *Vpřed* v prohlížeči, a díky metodám, které objekt history má, lze jejich použití simulovat. Objekt **history** je součástí objektu **window**, ale stejně jako obvykle, při použití aktuálního okna není nutné objekt **window** zmiňovat.

Pro pohyb zpět slouží metoda **back**, která se vrátí přesně o jeden krok. Tento řádek provede totéž co jedno klepnutí na tlačítko **Zpět**:

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

Pro pohyb o jednu stránku, ale dopředu, má objekt **history** metodu **forward**:

```
history.forward();
```

Poslední metoda, která provede požadovaný pohyb obousměrně o libovolný počet kroků, se nazývá **go**. Pokud se jedná o pohyb zpět, je jako parametr uvedena hodnota záporná, pro pohyb vpřed hodnota kladná. Tento řádek provede totéž co dvojnásobné klepnutí na tlačítko *Zpět*:

```
history.go(-2);
```

Pro pohyb dopředu o dva kroky je to pak:

```
history.go(2);
```

**Tip:** *Zajímavých objektů, jejich vlastností a metod je celá řada. K jejich popisu nejlépe slouží různé referenční příručky, jedna z možných je* JavaScript – Kompletní průvodce*, autora Davida Flanagana. Některé z těchto objektů využijete v praxi více, některé méně, ty, co jsem popsal, však byly základní a nešlo je opomenout.*

# 2. Události a jejich obsluha

Dalším pilířem, na kterém dynamický HTML stojí, je bohatá podpora různých událostí, které lze prostřednictvím skriptů obsluhovat. Tyto události nastávají na popud uživatele – např. pokud uživatel klepne myší na nadpis, bude vyvolána událost **onclick** na tomto konkrétním nadpisu. Nebo klepne-li myší kdekoliv do dokumentu, bude vyvolána událost **onclick** na aktuálním dokumentu. To lze demonstrovat na následujícím příkladu: pokud klepnete myší kdekoliv v aktuálním dokument, vypíše se zpráva metodou **alert**:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Zprava () {
    alert("Klepnul jsi myší do dokumentu");}
</SCRIPT>
</HEAD>
<BODY ONCLICK="Zprava()">
</BODY>
</HTML>
```

Tento příklad je velice jednoduchý a ukazuje obecné využití událostí. Chcete-li přiřadit událost nějakému prvku na stránce, jednoduše ovladač události (zde parametr **ONCLICK**) vložte do značky, která tento prvek definuje.

Událostí je také přejetí myší prvek dokumentu, tu reprezentuje událost **onmouseover**:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Zprava () {
    alert("Přejel jsi myší obrázek.");}
</SCRIPT>
</HEAD>
<BODY>
<IMG SRC="obrazek.jpg" ONMOUSEOVER="Zprava()">
</BODY>
</HTML>
```

## 2.1 Události a jejich stručný popis

Co jsou to události, byste měli zhruba vědět; podrobný popis najdete v druhé části, v kapitole 2 věnované skriptům, najdete tam i jejich stručné vyjmenování a popis. Na ty nejdůležitější se podíváme trošku podrobněji.

### 2.1.1 Události *onclick* a *ondblclick*

Událost **onclick** je vyvolána prohlížečem tehdy, když uživatel klepne na daný prvek, v němž je uveden jako parametr ovladač události **ONCLICK**. Tj. pokud je tento parametr umístěn ve značce **IMG**, která definuje obrázek, nastane událost onclick v případě, že uživatel klepne na tento obrázek. Událost **onclick** může nastat u každého prvku stránky, dokonce i u objektu **document**, jak jste mohli vidět na příkladu o kousek výše, a to tehdy, když uživatel klepne kamkoliv do dokumentu.

Hodnotou parametru **ONCLICK** je řetězec, který může obsahovat libovolný počet příkazů JavaScriptu, oddělených středníky, stejně jako každý jiný parametr definující ovladač události. Většinou však bývá v parametru **ONCLICK** uvedena funkce, která událost obsluhuje. Příklad:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Zprava () {
    alert("Klepnul jsi myší obrázek.");}
</SCRIPT>
</HEAD>
<BODY>
  <IMG SRC="obrazek.jpg" ONCLICK="Zprava()">
</BODY>
</HTML>
```

Událost **ondblclick** je té předchozí velice podobná, je však vyvolána pouze tehdy, když uživatel poklepal na tlačítko myši, tedy stisknul jej dvakrát rychle za sebou. Většinou se však používá událost **onclick**, tedy jednoduchého poklepání.

### 2.1.2 Události onmouseover, onmouseout a onmousemove

Událost **onmouseover** je jedna z nejdůležitějších událostí vůbec. Je vyvolaná tehdy, když uživatel přesune myš nad prvek zobrazený na stránce. Pokud se myš nad objektem hýbe, událost **onmouseover** nenastává; aktivní je pouze v okamžiku, kdy se myš nad objekt přesune. Ovladač události se definuje parametrem **ONMOUSOVER** v jakékoliv značce, která definuje nějaký objekt na stránce.

Událost **onmouseout** je naopak vyvolána ve chvíli, když se uživatel přesune myší mimo objekt. Jakmile je už mimo něj, událost vyvolána není; aktivní je pouze v okamžiku, kdy myš z objektu sjede pryč. Ovladač události je definován parametrem **ONMOUSEOUT** jakékoliv značky, kde chceme událost využít.

Příkladem budiž tento program. Pokud přesunete kurzor myši nad text, bude vypsán kurzivou. Pokud myš přemístíte mimo, bude opět vypsán normálně.

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Zmena (pismo) {
      test.style.fontStyle=pismo;}
</SCRIPT>
</HEAD>
<BODY>
  <P ID="test" ONMOUSEOVER=Zmena("italic")
     ONMOUSEOUT=Zmena("normal")> Přejeď přes mne myší.</P>
</BODY>
</HTML>
```

A konečně poslední událostí, která řeší pohyb kurzoru myši, je **onmousemove**. Ta nastane tehdy, když uživatel pohne myší nad daným prvkem. Tento příklad kdykoliv, kdy se myš pohne nad textovým polem **TEXTAREA**, zjistí aktuální souřadnice myši vzhledem k oknu prohlížeče a v tomto poli je zobrazí. Jakmile myší odjedete pryč z okna, souřadnice se přestanou zobrazovat.

```
<HTML>
<HEAD>
<TITLE>Souřadnice v textovém poli</TITLE>
<SCRIPT>
<!--
function souradnice() {
  var sou;
  sou = "X: " + event.clientX + " ";
  sou += "Y: " + event.clientY;
  document.all.souxy.value = sou;}
-->
</SCRIPT>
</HEAD>
<BODY>
<TEXTAREA NAME="souxy" COLS="30"
 ROWS="5" ONMOUSEMOVE="souradnice()">
</TEXTAREA>
</BODY>
</HTML>
```

### 2.1.3 Události *onload* a *onunload*

Tyto události jsou definovány na objektu **window**. **Onload** nastává, když je dokument nebo sada rámů plně načtena do prohlížeče. Bývá definován jako parametr **ONLOAD** značek **BODY** nebo **FRAMESET**.

Když je událostní ovladač **onload** vyvolán, je dokument vždy kompletně načten, a tudíž jsou všechny skripty provedeny, všechny funkce ve skriptech definovány a všechny prvky jsou jako objekty dostupné prostřednictvím objektu **document**. Událost **onload** se používá při akcích, které jsou spojené s načtením dokumentu, např spuštění nějakých časovacích operací (počítaní času apod.).

Přesným opakem je událost **onunload**. I ta je součástí objektu **window** a nastane tehdy, kdy prohlížeč ukončí práci s aktuálním dokumentem nebo sadou rámců (tzn. okno prohlížeče se zavře nebo je načítán nový dokument). Ovladač události **onunload** je definován parametrem **ONUNLOAD** značek **BODY** nebo **FRAMESET**, stejně jako u události **onload**.

Příkladem využití události onload může být zobrazení aktuálního času v textovém políčku formuláře. Poprvé se funkce *runClock* spustí při načtení dokumentu a pak cyklicky každou sekundu.

```
<HTML>
<HEAD>
<TITLE>Hodiny ve formuláři</TITLE>
<SCRIPT>
<!--
function runClock() {
  today   = new Date();
  document.clock.time.value = today;
  window.setTimeout("runClock()",1000);}
-->
</SCRIPT>
</HEAD>
<BODY ONLOAD="runClock()">
<FORM NAME="clock" ONSUBMIT="0">
<INPUT TYPE="text" NAME="time" SIZE="10" VALUE="">
</FORM>
</BODY>
</HTML>
```

## 2.2 Objekt *event* – určení původce události

Některé události nejsou bez některých dodatečných informací samy o sobě zajímavé. Například událost **onkeypress** není příliš užitečná, pokud nevíte, která klávesa byla stisknuta. To vše řeší objekt **event**, který je vestavěným objektem objektu **window** – je určen práci s událostmi. Říká vlastně, co všechno se stalo v době, kdy nastala určitá událost, u jakého prvku na stránce, kde v té době byla myš, jaká klávesa byla stisknuta apod.

Ze skupiny vlastností a objektů, které **event** sdružuje, je nejdůležitějším **srcElemen**t, který určuje, jaký prvek v dokumentu událost vyvolal:

```
window.event.srcElement
```

Jeho obsahem je objekt, který sdružuje snad všechny vlastnosti aktuální události. Celkem jich je přes dvacet, popíšu proto jen ty nejdůležitější. Už jen proto, že tyto vlastnosti nejsou vlastně vlastnostmi objektu **event.scrElement**, ale vlastnostmi objektů ve stránce; objekt **event.srcElement** ukazuje pouze na aktuální prvek, které se ho týká událost. Proto těchto vlastností lze využít i u konkrétních objektů reprezentujících prvky webové stránky.

## 2.2.1 ID, identifikátor prvku

Na rozdíl od jména prvku, které je definováno parametrem **NAME** a např. ve formuláři určuje objekt nebo vlastnost, se kterou se bude pracovat, je identifikátor (ID) jedinečné pojmenování prvku ve stránce. Jméno určené parametrem **NAME** se totiž může opakovat u různých prvků a to plodí určité nejednoznačnosti. Identifikátor přidáte k prvku jeho umístěním ve značce. Např. pokud chcete přidělit ID odstavci, uděláte to takto:

```
<P ID="Odstavec">Toto je odstavec definovaný identifikátorem.</P>
```

Pokud se pak budete chtít odkazovat právě na tento takto definovaný odstavec, učiníte tak přes identifikátor. Ten zjistíte z objektu **srcElement**:

```
identifikator = window.event.srcElement.id
```

Identifikátoru se používá zejména tam, kde není možnost použít parametru **NAME**. Pokud tedy pracujete např. s formulářem, můžete použít pro zjištění jména tohoto řádku:

```
jmeno = window.event.srcElement.name
```

## 2.2.2 Obsah textových prvků

Pokud je vyvolána událost, lze z objektu scrElement přečíst i text, který je součástí např. značek **P**, **DIV** nebo **H1**, a to tehdy, patří-li událost právě k těmto prvkům. K tomu slouží vlastnost **innerText**, která uchovává hodnotu tohoto textu. Pro výpis této vlastnosti může sloužit třeba tato funkce:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Vypis() {
    alert(event.srcElement.innerText);}
</SCRIPT>
</HEAD>
<BODY>
  <P ONCLICK="Vypis()">Text, který je vlastností innerText.</P>
</BODY>
</HTML>
```

Text odstavce nebo nadpisu však lze i dynamicky měnit, např. při přejetí myší:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Zmena (text) {
      event.srcElement.innerText=text;
  }
</SCRIPT>
</HEAD>
<BODY>
  <P ONMOUSEOUT=Zmena("onmouseout")
     ONMOUSEOVER=Zmena("onmouseover")>onmouseout</P>
</BODY>
</HTML>
```

Kdykoliv přejedete přes text „onmouseout“, změní se na „onmouseover“.

## 2.2.3 Styl textu

Stejným způsobem, jako lze zjistit text odstavce nebo nadpisu, lze zjistit a změnit i jeho aktuální styl. Můžete tak efektně měnit styl odstavce kdykoliv, když jej přejede kurzor myši nebo když na něj klepnete. Styl textového objektu, k němuž se váže aktuální událost, je uložen ve vlastnosti **style** objektu **srcElement**. Sám nemá žádný obsah, odkazuje pouze na jednotlivé vlastnosti stylu. Např.:

**srcElement.style.fontFamily** – zrcadlí parametr *font-family* (písmo stylu);
**srcElement.style.fontWeight** – zrcadlí parametr *font-weight* (tloušťku písma);
**srcElement.style.fontSize** – zrcadlí parametr *font-size* (velikost písma stylu) atd.

Příklad jednoduchého skriptu, který mění font písma odstavce, když přes něj přejede myší:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Zmena (pismo) {
      event.srcElement.style.fontFamily=pismo;}
</SCRIPT>
</HEAD>
<BODY>
  <P ONMOUSEOVER=Zmena("Times") ONMOUSEOUT=Zmena("Arial")
     STYLE="font-family: Arial"> Přejeď přes mne myší.</P>
</BODY>
</HTML>
```

Další vlastnosti už patří přímo objektu **event**, a nejsou tedy vlastnostmi objektu **event.scrElement**, který jen odkazuje na prvek, jehož se událost týká.

## Určení typu události

Typ události, který byl vyvolán nějakou akcí, se ukládá do vlastnosti **type** objektu **event**. Ta ukládá název události psaný malými písmeny a bez předpony on. Např. nastane-li událost **onmousedown**, bude hodnota vlastnosti type „mousedown“, pro **onclick** pak „click“ apod. Výhodou toho, že známe typ události, je to, že jediná rutina události může rozlišovat mezi více událostmi a může je provést:

```
function Zpracuj()  {
      // Spustí společnou funkci pro různé události
      typ = event.type;
      if (typ == "click")
            // Zpracuje událost onclick
      if (typ == "mouseover")
            // Zpracuje událost onmouseover}
```

## Pozice myši a stisknuté tlačítko

Jakmile nastane událost, můžeme přečíst z objektu **event** i aktuální pozici myši na obrazovce a přečíst i typ stisknutého tlačítka myši. To lze provést např. takto:

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function Souradnice() {
  alert("X:" + event.clientX);
  alert("Y:" + event.clientY);
  alert("Tlačítko:" + event.button);
  alert("Zdrojový prvek:" + event.srcElement.tagName);}
</SCRIPT>
</HEAD>
<BODY ONCLICK="Souradnice()">
</BODY>
```

Příklad po každém stisku tlačítka se vyvolá funkce *Souradnice*, která vypíše aktuální souřadnice vzhledem k aktuálnímu oknu, stisknuté tlačítko a prvek (značku), na nějž bylo takto klepnuto.

## Informace o stisknuté klávese nebo tlačítku

Objekt **event** rovněž zpřístupňuje vlastnosti, které představují aktuální klávesy a tlačítka myši, která jsou v čase události stisknuta. To umožní událostní funkci reagovat podle toho, co vlast-

ně bylo stisknuto. Jednotlivé vlastnosti objektu **event** jsou tyto:

- **button** – Aktuální nastavení stisknutých tlačítek myši:
  0 – není stisknuto žádné tlačítko,
  1 – je stisknuto levé tlačítko myši,
  2 – je stisknuto pravé tlačítko myši,
  4 – je stisknuto prostřední tlačítko myši.
  Pokud je např. stisknuto pravé a levé tlačítko myši, **button** vrátí hodnotu 3.
- **ctrlKey** – Logická hodnota, která udává, zda je stisknuta klávesa CTRL.
- **altKey** – Logická hodnota, která udává, zda je stisknuta klávesa ALT.
- **shiftKey** – Logická hodnota, která udává, zda je stisknuta klávesa SHIFT.
- **keyCode** – hodnota ASCII stisknuté klávesy.

Jednoduchý program, který vypíše ASCII kód stisknuté klávesy:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function Klavesa () {
      alert(event.keyCode);}
</SCRIPT>
</HEAD>
<BODY ONKEYPRESS="Klavesa()">
</BODY>
</HTML>
```

# 3. Příklady využití DHTML

Tato kapitola rozvíjí veškeré předchozí znalosti, a to zejména na konkrétních příkladech. Každá část této kapitoly bude vycházet z nějakého obecně využitelného programu, na němž popíšu nové prvky, s nimiž jste se dosud nesetkali. Nechci zde zbytečně zabíhat do podrobností a bezhlavě popisovat všechny možné metody, vlastnosti nebo objekty. Tato kapitola je spíše takový malý návod, kuchařka, z níž se dají vařit daleko lepší a chutnější pokrmy.

## 3.1 Hodiny běžící přímo ve stránce

Začneme běžícími hodinami ve stránce. Určitě si vzpomínáte na program, který generoval hodiny v textovém políčku formuláře. Se současnými znalostmi tento program můžeme přepsat tak, aby vepisoval běžící čas přímo do stránky, nikoliv jen do formuláře. Využijeme k tomu znalosti vlastnosti **innerText**, která umožňuje měnit text prvku, který má textový charakter a vlastnosti **id**, která je jedinečná pro každý prvek v dokumentu.

```
<HTML>
<HEAD>
<TITLE>Hodiny ve stránce</TITLE>
<SCRIPT>
<!--
function runClock() {
  today   = new Date(); // aktualizace času
  hours   = today.getHours();
  minutes = today.getMinutes();
  seconds = today.getSeconds();
  timeValue = hours;
  timeValue += ((minutes < 10) ? ":0" : ":") + minutes;
  timeValue += ((seconds < 10) ? ":0" : ":") + seconds;
  document.all.hodiny.innerText = timeValue; // zobrazení času
  window.setTimeout("runClock()",1000);
     // opětovné spuštění funkce za jednu sekundu
}
// -->
</SCRIPT>
</HEAD>
<BODY ONLOAD="runClock()">
   // prvotní vyvolání funkce pro počítaní času
<P ID="hodiny"></P>
</BODY>
</HTML>
```

Program je prakticky úplně stejný jako v kapitole 6.3.1, ovšem použil jsem menších úprav. Za prvé čas není přiřazován hodnotě políčka formuláře, ale přiřazuje se jako vlastnost **innerText** objektu *hodiny*, který je definován odstavcem pomocí identifikátoru „hodiny“ ve značce **P**. Hodiny se tak zobrazují v místě, kde je ve zdrojovém kódu HTML umístěna značka

```
<P ID="hodiny"></P>
```

Touto definicí vzniká v objektu **document.all** tak unikátní prvek, objekt *hodiny*, k němu lze přistupovat jako k normálnímu objektu. Vždy, když se změní čas, který se promítne do proměnné *timeValue*, je změněn i text objektu hodiny a ten je následně zobrazen. Tímto způsobem lze měnit jakýkoliv text ve stránce, je však třeba jej předtím nějakým způsobem označit, a sice identifikátorem **ID**.

## 3.2 Odpočítávání před přechodem na další stránku

Tento program opět používá časovače a stejné aktualizace obsahu textového prvku definovaného pomocí identifikátoru. Tentokrát se jedná o použití automatického přechodu na novou stránku po uplynutí stanoveného intervalu v sekundách. Pro efekt se také tento odpočet zobrazuje na stránce.

```
<HTML>
<HEAD>
<TITLE>Odpočítávání</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var Odpocet = 11;

function Prechod() {
    if (0 == Odpocet) // Přechod na další stránku.
        document.location = "index.html";
    else {
        Odpocet -= 1; // Snížení čítače a jeho zobrazení.z
        document.all.pocitadlo.innerText = Odpocet;
        // Čekat na další vteřinu.
        setTimeout("Prechod()", 1000);
    }
}
</SCRIPT>
</HEAD>
<BODY ONLOAD="setTimeout('Prechod()', 1000)">
Přechod na stránku <A HREF="index.html">home.html</A>
bude proveden za <SPAN ID="pocitadlo">10</SPAN> sekund.
</BODY>
</HTML>
```

Na stránce se tedy ukáže text „Přechod na stránku index.html bude proveden za 10 sekund." a postupně se bude odpočítávat. Aktualizace času je podobná jako v předchozím programu, jen čas není čten z objektu **Date** a z proměnné *odpocet*. Ta je každou sekundu snížena o jednotku a jakmile je rovna nule, je proveden přesun na další stránku novým obsahem objektu **window.location**.

## 3.3 Pulzující text

Pulzující text, tak by se dal nazvat tento program. Využívá vlastnosti **style** textového objektu definovaném ve stránce. Tomuto textu se jednou za určitý časový interval změní písmo na tučné a zpět, což vzbudí zdání pulzujícího textu.

```
<HTML>
<HEAD>
<TITLE>Pulzování textu</TITLE>
<SCRIPT LANGUAGE="JavaScript">

var tucny = false;

function Pulse() {
  if (tucny == false) {
     document.all.puls.style.fontWeight="normal";
     tucny = true;
  }
  else {document.all.puls.style.fontWeight="bold";
       tucny = false;
  }
  setTimeout("Pulse()", 500);
}
</SCRIPT>
</HEAD>
<BODY ONLOAD="setTimeout('Pulse()', 500)">
<SPAN ID="puls">Tento text pulzuje.</SPAN>
</BODY>
</HTML>
```

V tomto programu je využito stylu, tedy jeho změny za určitý časový okamžik. Textu „Tento text pulzuje." je přiřazen identifikátor *puls*, který vytvoří v objektu **document.all** nový objekt, *puls*. Změnou vlastnosti **fontWeight** vlastnosti **style**, a tedy cyklickým nastavením tučného a normálního písma se docílilo efektu, že text pulzuje. Změna stylu může být samozřejmě libovolná. Opět se tu využívá časování a metody **setTimeout**. Až by se zdálo, že v dynamickém HTML neexistuje nic jiného. To však naštěstí vůbec není pravda.

## 3.4 Obrázky měnící se při přejetí myši

Velice zajímavým efektem je blikající obrázek, zejména pokud je zároveň odkazem. Jde o efekt, kdy je do dokumentu vykreslen obrázek, který je tmavší, než by odpovídalo normálu. Pokud na něj najedete myší, rozsvítí se, stane se aktivním. A opět, pokud z něj myší uhnete, zhasne, stane se opět normálním.

```
<HTML>
<HEAD>
<TITLE>Měnící se obrázky</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
  function makeCool(obrazek) {obrazek.src="obrazek_svetly.jpg"}
  function makeNormal(obrazek) {obrazek.src="obrazek_tmavy.jpg"}
-->
</SCRIPT>
</HEAD>
<BODY>
<IMG ID="Obrazek1" ONMOUSEOVER="makeCool(Obrazek1)"
     ONMOUSEOUT="makeNormal(Obrazek1)" SRC="obrazek_tmavy.jpg"
     BORDER="0">
</BODY>
</HTML>
```

V tomto programu se poprvé setkáváte s objektem typu **image**, tedy objektem, který v dané webové stránce reprezentuje obrázek. Stejně jako všechny objekty na webové stránce i objekt typu **image** patří do objektu **document.all**. Odkaz na tento objekt je zde záměrně vynechán, abyste viděli, že program pracuje i bez něj. Objekt, který JavaScript nezná, se totiž nejdříve hledá právě v souboru objektů **document.all**. Co z toho plyne? Ve všech předchozích příkladech, kde jsem použil odkaz na objekt document.all, můžete tento odkaz vynechat.

Ale zpět k objektu typu **image**. Tento objekt má základní vlastnost **src**, která je prvotně definovaná parametrem **SRC** značky **IMG**. V této vlastnosti je uložen název obrázku, který k obrázku patří. Pokud jej změníte, změní se i obrázek na webové stránce.

Stejně jako v předchozích programech, i zde je objekt, který se dynamicky mění, označen identifikátorem, konkrétně „Obrazek1“, aby skript věděl, s jakým objektem typu **image** má pracovat.

## Mrkající obrázky

Pokud využijete znalostí obou předchozích programů, můžete vytvořit mrkající obrázky. Tentokrát se nebudou měnit tehdy, když budete hýbat myší, ale v závislosti na časovači. Lze tak vytvářet i jednoduché animace, pokud počet obrázků ze dvou zvýšíte na více.

```
<HTML>
<HEAD>
<TITLE>Pulzování obrázku</TITLE>
<SCRIPT LANGUAGE="JavaScript">

var tmavy = true;

function Pulse(obrazek) {
  if (tmavy == true) {
     obrazek.src="obrazek_svetly.jpg";
     tmavy = false;
  }
  else {obrazek.src="obrazek_tmavy.jpg";
     tmavy = true;
  }
  setTimeout("Pulse(Obrazek1)", 1000);
}
</SCRIPT>
</HEAD>
<BODY ONLOAD="setTimeout('Pulse(Obrazek1)', 1000)">
<IMG ID="Obrazek1" SRC="obrazek_tmavy.jpg" BORDER="0">
</BODY>
</HTML>
```

Program se liší jen v tom, že místo změny stylu textu provádí cyklickou výměnu obrázků, které jsou definované objektem typu **image**, zde *Obrazek1*. Obrázkem může být třeba písmo, které budí dojem pohybu, nebo třeba měnící se logo.

## Změna barvy pozadí dokumentu

Jednoduchou aplikací změny vlastnosti **bgColor** objektu **document**, což je synonymum pro barvu pozadí stránky, je tento program. Ten nastaví barvu pozadí v závislosti na tlačítku, které bylo stisknuto.

```
<HTML>
<HEAD>
<TITLE>Změna barvy pozadí</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function zmena(barva) {
   document.bgColor=barva}
-->
</SCRIPT>
</HEAD>
```

```
<BODY>
<INPUT TYPE="button" VALUE="Červená"
       OnClick="zmena('#ff0000')">
<INPUT TYPE="button" VALUE="Oranžová"
       OnClick="zmena('#993300')">
<INPUT TYPE="button" VALUE="Žlutá"
       OnClick="zmena('#ffff00')">
<INPUT TYPE="button" VALUE="Zelená"
       OnClick="zmena('#6b8e23')">
<INPUT TYPE="button" VALUE="Modrá"
       OnClick="zmena('#000080')">
<INPUT TYPE="button" VALUE="Bílá"
       OnClick="zmena('#ffffff')">
</BODY>
</HTML>
```

Pokud odstraníte tlačítka a necháte pozadí náhodně generovat, např. pomocí programu na sázení do loterie ze začátku třetí části můžete generovat náhodná čísla, která budete přiřazovat barvě textu nebo pozadí a vytvořit tak efekt duhy. Tehdy lze použít časovač z předchozích příkladů, aby se generování provádělo třeba každých deset sekund.

Praktický význam tohoto příkladu asi není žádný, měl však ukázat, že dynamicky se dá v HTML měnit snad úplně všechno, i ta barva pozadí.

Tady bych celé povídání o JavaScriptu, objektovém programování a dynamickém HTML skončil. Tato kniha měla za úkol naučit základy programování webu, a toho, myslím si, dosáhla. Měli byste ji zavírat s tím, že znáte minimum pro vytváření jednoduchých skriptů, které umí obsluhovat základní události prohlížeče, které umí pracovat s objekty nebo dynamicky mění prvky webového dokumentu. Pokud máte jakékoliv přípomínky, výhrady nebo nápady, jak knihu v dalších vydáních vylepšit, pište na adresu *Petr.Broza@cpress.cz*. Budu vděčný za odezvu a pokud to bude v mých silách, rád na vaše dotazy odpovím.

V závěru bych vás tedy odkázal na několik dalších pramenů, z nichž lze čerpat; jsou sice obsáhlé a místy těžko srozumitelné, ale při troše vůle se z nich stanou opravdu dobří pomocníci při programování webu. První knihou je *JavaScript, kompletní průvodce* od Davida Flanagana, druhým pak *Dynamické HTML* od Scotta Isaacse. Obě vydalo nakladatelství Computer Press, stejně jako tuto publikaci.

# Příloha:
# Co je Hyperlink
# a jak jej používat

# 1. K čemu slouží Hyperlink

Na adrese *http://www.hyperlink.cz* si může kdokoliv založit svou stránku, ať už pro soukromé či firemní účely, zcela zdarma, a to velmi jednoduchým způsobem i bez znalostí jazyka HTML nebo tvorby webových stránek. Hyperlink je jakousi databází stránek a prezentací, které však fungují samostatně a nejsou nijak vázané. Nyní je na Hyperlinku přes 9 tisíc stránek a prezentací. Je to výjimečně výhodná služba nejen u nás, a proto se na ni zaměříme.

## 1.1 Co vám Hyperlink nabízí?

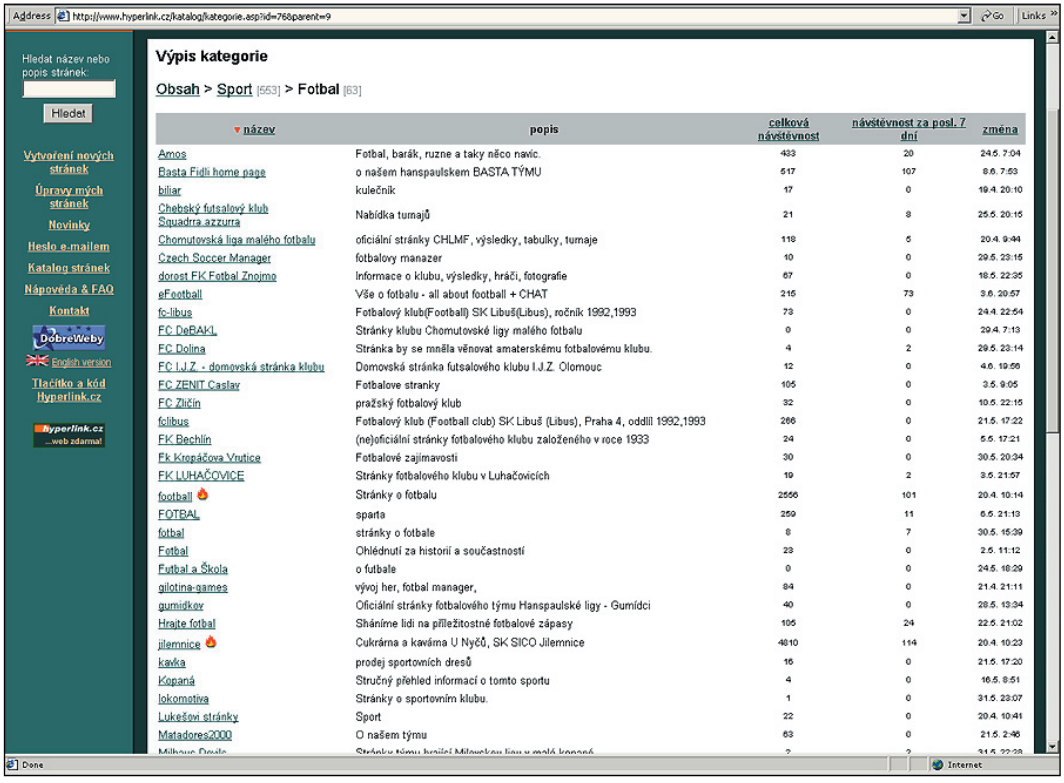
V prvé řadě je to prostor na Internetu, a to celých 12 MB, a věřte mi, že vám dá práci tento prostor zaplnit. Pokud by vám však přesto nestačil, pošlete e-mail a Adresa vaší stránky pak může být buď *http://www.hyperlink.cz/cokoliv* nebo si můžete nechat snadno zaregistrovat doménu třetího stupně a pak bude adresa *http://www.cokoliv.hyperlink.cz*. Aktualizujete-li často své stránky nebo potřebujete-li přesunout větší množství souborů včetně složek, můžete využít možnosti přístupu k vašemu webu přes *FTP*. FTP (File Transfer Protocol) je jednou z možných a častých cest, jak posílat soubory přes Internet. Každý den vám pracovník Hyperlinku zpracuje statistiky přístupu na vaše stránky. Můžete tak sledovat nejen, jaký je o vaše stránky zájem, ale také zda figurují například v často navštěvovaných stránkách Hyperlinku.

Co opravdu činí tuto službu výjimečnou je fakt, že vám její provozovatel nenutí vůbec žádnou reklamu, to znamená nesnaží se na vašich stránkách vydělat. Žádné nevyžádané ban-

Úvodní obrazovka služby Hyperlink.cz

nery nebo reklamní proužky na svých stránkách mít nebudete. Naopak provozovatel vám dovoluje si na své stránky umístit vlastní reklamu. Pokud si na Hyperlink umístíte například stránku, kde budete nabízet nějaké služby nebo prodej, i to je dovoleno či dokonce doporučeno. Autoři doslova píší: „Hyperlink.cz je webhosting, který umožní vaše podnikání na Internetu. Neumísťujeme na vaše stránky žádné reklamní bannery; naopak, vy máte volnou ruku v umísťování vašich vlastních bannerů na Hyperlink.cz. Máte rovněž povoleno provozování jakékoli vlastní komerční činnosti, pokud tím neporušujete smluvní podmínky při registraci na Hyperlink.cz. Ba přímo, budeme potěšeni, když Hyperlink.cz pomůže k vašemu podnikání na Internetu! Protože bychom ale rádi, abyste s námi vytvářeli velkou rodinu uživatelů Hyperlinku, je jedinou malou podmínkou umístění malého tlačítka na vaši titulní stránku.“

Dalším velkým plus je možnost ukládat na Hyperlink WML stránky, které si mohou prohlížet uživatelé mobilních telefonů s podporou WAPu. Tato nová technologie se pomalu dostává do podvědomí uživatelů mobilních telefonů a pokud i vy, zručnější, své stránky alespoň z části převedete do formátu WML, máte šanci, že si tak získáte nejen nové návštěvníky, ale také uznání.




Výpis kategorií jednotlivých stránek na Hyperlinku

## 1.2 Jak na Hyperlinku založit stránku?

Postup je velmi jednoduchý a je přímo určen i pro úplné začátečníky. Stačí pozorně sledovat instrukce a řídit se jimi.

Na titulní stránce Hyperlinku klepnete vlevo na odkaz vytvoření nových stránek. To vás přenese na stránku, kde najdete větu: Registraci nového uživatele služby Hyperlink zahájíte na následující stránce. Klepněte na výraz „následující stránce“ a dostanete se k formuláři, kam vyplníte své údaje a odsouhlasíte tyto podmínky:

1. Stránky, které si u nás vytvoříte, jsou zdarma – nyní i kdykoli pro budoucnost. Můžete je používat pro svou osobní potřebu, zábavu, výuku nebo pro své podnikání.

2. Můžete na svých stránkách uveřejňovat reklamní odkazy, placené i neplacené. Můžete si na nich vytvořit elektronický obchod či poskytovat jinou službu – stránky na Hyperlinku jsou určeny pro to, aby vám pomohly v podnikání!

3. Nežádáme uveřejňování žádných reklamních bannerů pocházejících od služby Hyperlink.cz. Protože bychom ale rádi, abyste s námi vytvářeli velkou rodinu uživatelů Hyperlinku, je jedinou malou podmínkou umístění malého tlačítka na vaši titulní stránku; tlačítko a kód najdete pod odkazem „Tlačítko a kód Hyperlink.cz“ v levém menu. Pokud chcete vytvářet takový design stránky, který by i toto tlačítko znemožňoval, napište nám.

4. Máte k dispozici bezkonkurenčně velký prostor pro vaše stránky – celých 12 MB, můžete využívat i podadresáře. Nesmíte však zvyšovat prostor vašich stránek „navazováním“, vytvořením dalších účtů a prostorů a jejich propojováním do jediného webu.

5. Stránky, které vytvoříte a soubory, které na weby Hyperlink.cz uložíte, musí být v souladu s právním řádem České republiky. Zvláštní důraz klademe na dodržování autorského zákona. Za stránky, které vytvoříte, nese po právní stránce plnou zodpovědnost jejich autor.

6. Na vašich stránkách na Hyperlink.cz nepovolujeme publikování erotických ani pornografických materiálů.

7. V případě porušení bodů 4 – 6 jsme nuceni vaše stránky smazat i bez předchozího upozornění; provozovatel Hyperlink.cz nenese odpovědnost za případné škody tím způsobené.

8. Naším úsilím je udržet servery Hyperlink.cz provozuschopné 24 hodin denně a sedm dní v týdnu; mohou se však vyskytnout plánované výpadky (údržba, rozšiřování) či výpadky neplánované, dané technickými poruchami na straně poskytovatele či

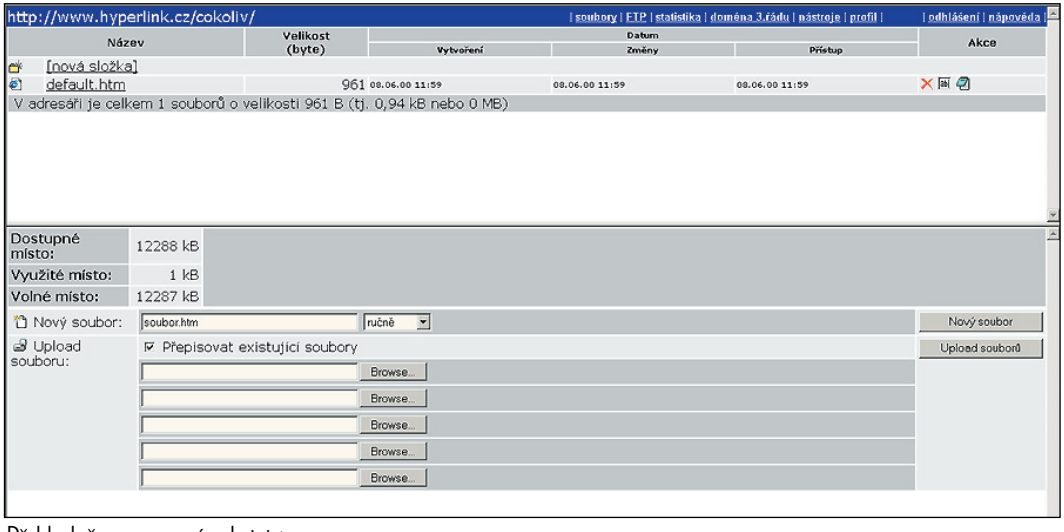
Hyperlink.cz. Provozovatel serveru nenese odpovědnost za jakékoli škody způsobené nedostupností těchto stránek, nicméně se zavazuje učinit vše pro nápravu situace.

9. Vaše registrační informace se zavazujeme nepředat žádné třetí straně.

Důležitý je hlavně bod číslo 5, který je opravdu nutné dodržovat. Do základních údajů zadáte přihlašovací jméno, což je v podstatě název vašich stránek. Pokud tedy zadáte jako přihlašovací jméno „chlupatý_králíček“, pak bude adresa vašich stránek *http://www.hyperlink.cz/ /chlupaty_kralicek*. Doporučuji zadávat pokud možno jednoslovné názvy a bez háčků a čárek. Budou se pak nejen lépe pamatovat vašim návštěvníkům, ale také se budou snadněji zadávat do prohlížeče. Určitě se vyhýbejte takovým názvům, jaký jsem uvedl v příkladu. Samozřejmě jiný případ je pokud zakládáte nebo přemísťujete z jiného webu firemní stránky, to pak název nelze snadno měnit. I tak se pokuste název co nejvíce upravit, adresa *http://www.hyperlink.cz/moravska_vinice_nedaleko_Brna* možná bude tou nejpřesnější, ale rozhodně nebude vypadat dobře. Přitom by stačilo nahradit jej prostým „vinice“. Další věc, kterou musíte vyplnit co nejpečlivěji, a to hned dvakrát, je heslo, na které budete pak dotázáni pokaždé, když budete chtít své stránky editovat.

Naopak v položce název můžete zadat i delší název vašich stránek, například právě plný název firmy. Název bude tím, pod čím budou moci vaši stránku nalézt ostatní uživatelé na Hyperlinku. Čím bude název výstižnější, tím lépe. Další položkou je popis a ten by měl plně vystihovat obsah vašich stránek, měl by být stručný a jasný. Pokud možno jedna věta, která zaujme člověka natolik, že právě na odkaz na vaši stránku klepne.

Kategorie stránek jsou položky, kde si můžete vybrat ze všech kategorií na Hyperlinku a zařadit svou stránku do tří z nich. Záleží jen na vás, kam stránky zařadíte. Zbylé kontaktní údaje jsou již obvyklé osobní informace o vás (jméno, adresa, firma atd). Tady byste si měli




Přehledně zpracovaná administrace

dát pozor na jednu položku, kterou je e-mail. Raději třikrát zkontrolujte, zda je správně, protože jakmile zadáte chybný, nebude možné stránky vytvořit.

Jakmile totiž odešlete vyplněný formulář, objeví se vám zpráva o tom, že na váš e-mail byly zaslány pokyny, jak pokračovat. Tam se dozvíte, že je nutné své stránky aktivovat do sedmi dnů, jinak bude registrace neplatná.

Na stránce, která se vám v e-mailu objevila, se pak poprvé přihlásíte do systému pomocí svého přihlašovacího jména, hesla a kódu, který jste obdrželi e-mailem. Následuje gratulace provozovatele stránky a vy můžete začít své stránky buď tvořit nebo nahrávat.

Tvorba stránek není přes administraci nijak komplikovaná, ale je jednodušší, pokud si své stránky vytvoříte u sebe na lokálním disku a pak je pouze nahrajete přes FTP. Postup je následující:

Přihlaste se do systému, tím se dostanete do své administrace. Tam klepněte vlevo nahoře na FTP. Zobrazí se stránka, kde jsou všechny potřebné údaje k tomu, abyste mohli ihned přes vašeho FTP klienta (například Cute FTP) nahrávat své stránky, pokud patříte k těm zručnějším. Pokud ne, stačí klepnout na slovo zde ve větě „pokud váš prohlížeč podporuje FTP protokol klepněte zde a můžete začít hromadně přenášet soubory". Otevře se vám nové okno a v něm soubory, které na stránce zatím máte. Tím nejjednodušším způsobem, zkopírováním Ctrl+C a Ctrl+V sem vložíte svou stránku. Pamatujte jen na to, že úvodní strana musí být vždy pojmenována index.html, index.htm nebo default.html či default.htm.

Pokud v administraci klepnete v horní liště na nástroje, tak se vám objeví možnost vkládat předvolené skripty jako vlastní počítadlo na stránce, anketu, můžete zaheslovat část vašich stránek, umístit na ně vlastní guestbook či hodiny.

Pomocí odkazu doména 3. řádu můžete aktivovat své stránky pod adresou *http://cokoliv.hyperlink.cz*.

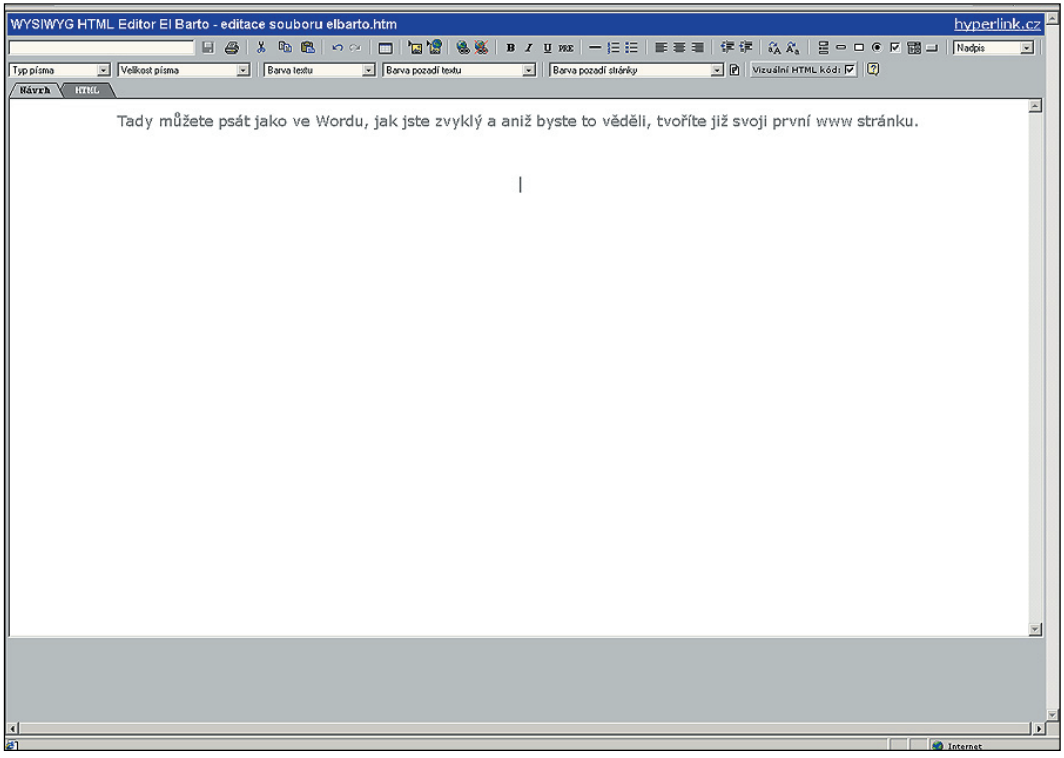
To, jak budou stránky vypadat, a jak často budou navštěvované, už záleží jen a jen na vás.

## 1.3 Co je El Barto?

El Barto je HTML editor pro úplné začátečníky napsaný právě pro uživatele serveru Hyperlink. Můžete v něm velmi jednoduše své stránky vytvářet či editovat. Dá se říci, že jde o Front Page s omezenými funkcemi. Nevíte co je Front Page? To je program podobný Wordu, ale namísto textu je výsledkem stránka HTML. V El Bartu můžete naprosto pohodlně stránky vytvářet, jako byste psali dokument. Aniž byste o tom věděli, tak si text, obrázky, tabulky, rámce a další funkce převádí do HTML program sám. Měnit můžete cokoliv od barvy a typu písma, přes barvu pozadí vaší stránky až po vkládání obrázků a tabulek. Ovládání je opravdu jednoduché a je určeno pro úplné začátečníky.

Nejlepší bude, když si sami El Barta vyzkoušíte, k čemuž nepotřebujete mít vlastní stránku, a to klepnutím na jeho logo na titulní straně Hyperlinku nahoře.

Petr Král




Vytvářet stránky webovou aplikací je stejně jednoduché, jako psát ve Wordu

# Rejstřík

# Rejstřík příkladů

# Doporučená literatura:


1. **Tvorba WWW stránek pro úplné začátečníky,**
   Petr Broža, Computer Press, 144 stran, [K0247].
2. **Vytváříme WWW stránky a spravujeme moderní web site,**
   Jiří Hlavenka, Radek Sedlář, 2. dopl. vydání, 350 stran, [K0132].
3. **Příručka tonoucího webmastera,**
   Microsoft Press, Marry Haggard, 186 stran, [K0214].
4. **Dynamické HTML,**
   Microsoft Press, Scott Isaacs, 490 stran + CD-ROM, [K0163].
5. **Tvorba internetových aplikací v XML,**
   Simon St. Laurent, MIS Press, 240 stran, [K0245] .
6. **1001 tipů a triků pro Internet, 2. doplněné vydání,**
   David Morkes, Jan Vořech, Computer Press, 432 stran, [K0286].
7. **Využíváme Internet s programem Microsoft Internet Explorer 5 CZ,**
   Jiří Hlavenka, Computer Press, [K0266].
8. **JavaScript – kompletní průvodce,**
   O'Reilly, David Flanagan, 725 stran, [K0170].