1 Úvod

Tabulky kaskádových stylů (Cascading Style Sheets, CSS) jsou nadstavbou vyznačovacích jazyků HTML, XHTML či XML. **Slouží k popisu prezentace dokumentů,** aniž by jakkoli ovlivňovaly jejich obsah a strukturu.

Tato kniha se věnuje kompletnímu popisu jazyka CSS, způsobům jeho použití, praktickým radám a řešením nejčastějších problémů, s nimiž se autoři mohou setkávat. V budou představeny základní informace o CSS, proč je používat a jaké jsou jejich výhody. Ve druhé kapitole jsou na příkladu jedné typické stránky představeny základní možnosti formátování pomocí kaskádových stylů. Kapitola třetí je podrobným popisem CSS, naleznete zde souhrn všech vlastností CSS a přehled jejich podpory v nejznámějších prohlížečích. Závěrečná, čtvrtá kapitola se věnuje praktickým radám a postupům při používání stylů. Odkazy na důležité internetové adresy a dokumenty i ukázky a příklady použité v textu jsou uvedeny v Referencích na konci knihy.

1.1 Co jsou kaskádové styly, kde se vzaly a k čemu jsou dobré?

Ještě než se začneme věnovat popisu a používání kaskádových stylů, je na místě zmínit některé souvislosti a důvody nástupu této technologie. Kde se vzala, co její použití může přinést autorům, zadavatelům i uživatelům a jaké jsou možnosti i limity formátování pomocí CSS.

1.1.1 Formátování dokumentů HTML

> *HTML nebyl nikdy zamýšlen jako jazyk vzhledu a interaktivity. Byl navržen, aby tvůrcům umožnil dokumenty řádně označit a uspořádat. S růstem trhu prohlížečů jsme z tohoto jednoduchého jazyka udělali nástroj na tvorbu prezentací. Skončili jsme s komplikovaným značkováním určeným pro vizuální efekty. … Návrat k ne-prezentačním dokumentům nám otevírá nové možnosti.*
>
> *Molly Holzschlag,*
> *webdesignerka, konzultantka, autorka více než dvaceti knih a množství článků,*
> *členka organizace Web Standards Project (z rozhovoru pro WebBuilder.com)*

Původním záměrem jazyka HTML byl kromě definování vazeb mezi dokumenty především popis jejich struktury. Určit, která část je nadpisem, která odstavcem, tabulkou, co je citací z jiného zdroje, co jménem autora či který úsek tvoří seznam a který jeho položky. To, jak se tyto úseky dokumentu mají zobrazit, již mělo být řečeno jinak a jinde. Zpočátku to záviselo prakticky jen na použitém výstupním zařízení — jímž je nejen obrazovka počítače (a různé typy prohlížečů), ale i tiskárna, projektor nebo třeba čtecí zařízení. Později byly vyvinuty další technologie, které umožňovaly popisovat podobu dokumentů a přitom do struktury stránek nezasahovaly.

Jenže mezitím, vyhovujíce rychle rostoucím potřebám autorů i zadavatelů, se postarali výrobci prohlížečů o množství doplňků a rozšíření jazyka HTML nad rámec standardu. HTML se tak roztříštil na mnoho druhů a poddruhů, které odpovídaly pouze jednotlivým verzím konkrétního prohlížeče. Od prapůvodního návrhu tak v HTML přibylo postupně velké množství funkcí a značek, sloužících jen pro účely formátování dokumentů — namátkou značka `<font>` pro určení stylu písma či atribut `bgcolor` pro barvu pozadí. Je to pochopitelné, autoři potřebovali možnost dát svým stránkám na webu barvy, vlastní písma, navrhovat vzhled podle své fantazie či potřeb zadavatele. Chtěli tvořit a odlišovat se od ostatních, a standardy jazyka za jejich požadavky dalece zaostávaly. Pohříchu tak všechny nové funkce do HTML živelně implantovali jen výrobci prohlížečů a původní autoři jazyka jim bezmocně přihlíželi.

1.1.2 Problémy a potíže na každé straně

S přibývajícími nestandardními rozšířeními se z dokumentů postupně začala vytrácet jejich struktura. Kód se stále víc zaměřoval jen na popis výsledného vzhledu. S nástupem tabulek (a

především několikanásobně vnořených tabulek), coby mohutného nástroje pro pozicování prvků, zmizela struktura z HTML dokumentů takřka úplně.

> *Formátování tabulkami působí problémy (také) postiženým lidem, především slepým. Čtecí zařízení neví, jak číst sloupce. … Tabulky nikdy nebyly zamýšleny pro rozmísťování prvků, vytvářejí mřížku. … Čtečka může přečíst první řádek prvního sloupce, poté první řádek druhého sloupce a první řádek třetího, čímž učiní stránku uživateli naprosto nesrozumitelnou.*
>
> *Molly Holzschlag, z rozhovoru pro WebBuilder.com*

Kdyby strukturu v dokumentech postrádali pouze autoři původních standardů, nikoho by to netrápilo. Její absence má ale mnoho nepříjemných důsledků, které již leckoho trápit mohou. A trápí.

Automatické zpracování dokumentů je stále složitější — kvůli zmizelé struktuře se ve stránkách vedle sebe objevují úseky, které spolu vůbec nesouvisejí, někdy je naopak souvislý text rozdělen na několika vzdálených místech dokumentu, vyhledávače na Internetu si s takovým obsahem jen těžko vědí rady. Vytrácejí se některé půvabu původního HTML, především přenositelnost a kompatibilita. Stránky jsou formátovány jen pro jeden typ počítače či dokonce pro jednu verzi jednoho konkrétního prohlížeče. Handicapovaní uživatelé, uživatelé používající jiné prohlížeče či dokonce úplně jiné zařízení stránku nemusí vůbec přečíst.

> *Oddělení prezentace od struktury je také ekonomicky výhodné. … Naleznete zdroj pro fantastické snížení režijních nákladů: např. pokles objemu dat ze 150 KB na 25 KB na jednu stránku … Přináší to také obrovský rozdíl ve správě grafiky a údržbě stránek — všechno zjednodušíte a tady mluvíme o čase, penězích, o počtu lidí, které musíte najmout.*
>
> *Molly Holzschlag, z rozhovoru pro WebBuilder.com*

Formátovací značky ve stránkách WWW mohou tvořit třeba 80 % jejich datového objemu, což by mělo zajímat všechny, kteří internetové projekty financují. Mnoho manažerů by se nestačilo divit při zjištění, že pokud by jejich webdesigneři odstranili ze stránek veškeré formátovací značky a ponechali pouze strukturované dokumenty, přenesla by se ze serveru k uživatelům třeba jen čtvrtina dat a oni by tak mohli ušetřit i mnoho set tisíc korun, které nyní vydávají za každý gigabajt navíc. A podobných souvisejících problémů existuje celá řada.

1.1.3 Řešením jsou standardy

Právě kvůli takovýmto potížím se trend pomalu obrací. Před lety nabrala na síle standardizační organizace W3C (World Wide Web Consortium). Začala zastřešovat projekty související s publikováním na webu a konečně koordinovaněji řídit nové sjednocující standardy. I zmínění, dosud živelně působící výrobci prohlížečů (především Netscape a Microsoft) se do projektu W3C zapojili a postupně — sice pomalu, ale jistě — začínají nové standardy zavádět do praxe a prosazovat ve svých prohlížečích. Nové verze HTML již postupně odstraňují nepatřičně naroubované formátovací „přívažky“ a označují je za *nedoporučené.* Zatím poslední norma

XMTHL 1.1 (nástupce jazyka HTML, jehož vývoj byl již ukončen) všechny formátovací značky a podobná rozšíření dokonce zcela vypustila. Značky jako `<big>`, `<small>`, `<font>`, atributy typu `bgcolor`, `alink`, `vlink` či `align` už v XHTML vůbec nenajdete. Jejich použití v dokumentech není ani tolerováno a považuje se za chybu.

Webové prohlížeče pak na druhé frontě tyto standardy začínají dodržovat: aktuální verze (např. MSIE 6 či Mozilla 1) již pracují v různých režimech podle typu dokumentu. Ke stránkám ve starších verzích HTML jsou tolerantnější a úchylky od norem přehlížejí (jako to bylo obvyklé u dřívějších verzí prohlížečů), k dokumentům podle novějších standardů jsou však méně kompromisní a leckterou nekorektní konstrukci odmítnou a prohlásí za chybnou. A lze očekávat, že další verze prohlížečů budou ještě striktnější a důslednější k daným normám — což je ostatně zřejmý trend, nezbytný k masovému nasazení dokumentů XML, které striktní přístup ze své podstaty přímo vyžadují.

Oklikou přes několik pionýrských let se tak HTML vrací zpět ke své dřevní podstatě: poskytovat strukturované dokumenty s jasným logickým členěním. A všechny konstrukce, které netvoří samotný obsah nebo popis struktury dokumentu (jako např. formátování, dynamické akce, prvky ActiveX, skripty atd.), jsou vykázány mimo HTML a nesmí být nadále jeho součástí.

1.1.4 Na formátování jsou kaskádové styly

Jednou z takových technologií je právě jazyk **Tabulek kaskádových stylů** (CSS), o němž pojednává tato kniha. Umožňuje formátovat dokumenty, definuje způsob jejich prezentace na koncových zařízeních, popisuje podobu stránek a styl jednotlivých prvků, a přitom nijak neovlivňuje obsah dokumentů samotných. Když definice stylu odstraníme nebo nepoužijeme, zůstane nezměněný původní dokument.

I když se ještě nacházíme v přechodném období, kdy má starý přístup k WWW stránkám stále mohutné zázemí s velkou setrvačnou tendencí a přístup nový je teprve v počátcích — s prvními projekty progresivnějších webdesignerů a jen zvolna „přitvrzujícími“ moderními prohlížeči. Je načase se na tuto změnu připravit. Kompletní přechod, kdy bez CSS nebude možné formátovat dokumenty vůbec, je sice ještě hudbou vzdálené budoucnosti, ale je patrně nezvratný. Přitom na nový styl práce s dokumenty HTML je možné přejít již nyní. Když už nechcete být na špici a připraveni, „až to přijde“, můžete si znovu přečíst výše uvedené problémy s nestrukturovanými stránkami nebo si vzpomenout na některé další z vlastní zkušenosti. S oddělením popisu prezentace od struktury a obsahu tyto komplikace zmizí a objeví se i mnohé výhody.

Stačí se jen naučit dobře používat CSS, odhalit všechny jeho možnosti a především přijmout nový styl tvorby stránek WWW. Chcete-li začít právě teď, tato kniha může být dobrým výchozím bodem.

1.2 Používat, či nepoužívat kaskádové styly?

Že je používání technologie CSS výhodné a někdy v budoucnu bude i nezbytné, jsme si řekli. O tom, jak se kaskádové styly používají, pojednává zbytek této knihy. Co však s nimi dokážeme? Co je naopak nemožné? Jsou styly vhodné opravdu pro každého, nebo mají pravdu autoři, kteří jsou přesvědčeni, že pro ně je lepší zůstat u „osvědčených“ postupů?

1.2.1 Na změnu musíme být připraveni

Zkušeným webdesignerům lze jistě *doporučovat,* aby na CSS přešli co nejdříve. Ale asi by nebylo šťastné nutit autory, aby ihned, okamžitě a teď! nechali svých pracně naučených znalostí HTML a po hlavě se pustili do nové, neznámé technologie, kterou možná ještě dostatečně nepochopili. Samotnému mi až po dlouhém experimentování došlo, že vzdát se mých oblíbených tabulek pro mě může být velkým přínosem. Že i když bez nich několik drobností zatím nejde udělat, oněch *pro* na druhé straně je tolik, aby případné nevýhody několikanásobně převážila. Dospěl jsem k tomu však až poté, co jsem si CSS náležitě osahal, na mnoha webech vyzkoušel a všechny postupy se mi dostaly náležitě pod kůži. A ačkoli jsem kaskádové styly aspoň zčásti používal od samého začátku, k jejich plnému využití jsem se odvážil až po delší době.

Kaskádové styly se totiž mohou používat v zásadě dvojím způsobem — jako doplněk „starého“ formátování pomocí značek HTML, i jako plnohodnotný formátovací nástroj.

1.2.1.1 CSS jako doplněk

První přístup je snadnější a také nejčastější. Rozšíříme-li své stávající stránky o CSS, můžeme tím odpoutat svůj grafický rozlet. Použít barvy a obrázky na pozadí, definovat snadněji písma, rozestupy mezi prvky, rámečky. Ale všechny uvedené nevýhody tím pořád ještě neodstraníme. Dokumentu stále chybí struktura, kód je stále málo přehledný, složitěji se upravuje a udržuje, stále obsahuje množství nadbytečných značek.

Starý přístup k formátování stránek je ale hluboce vrytý do způsobu práce většiny autorů a je mnohdy jen těžko překonatelný. Mnoho let se web dělá právě takto. Každý prvek na stránce se navrhuje již s přesnou představou jeho budoucího vzhledu v prohlížeči. Nejprve se vymýšlí, jak stránka bude vypadat a tomu se přizpůsobuje rozvržení obsahu v kódu. Někdy bohužel i obsah samotný.

Autor, který to chce změnit a chce začít vytvářet stránky s plným využitím všech možností nových technologií, ale musí udělat víc, než jen „naučit se“ nějaké definice a konstrukce: především musí **naprosto změnit přístup.** Bez pochopení, že s CSS se stránky dělají úplně jinak než dřív, to prostě fungovat nebude.

1.2.1.2 CSS v plné síle

Přístup k tvorbě stránek, tak jak je zamýšlen standardy (X)HTML a CSS, je z tohoto pohledu skutečně úplně jiný, odlišný v samotném principu. Při návrhu dokumentu se totiž o jeho budoucí podobě vůbec neuvažuje, postupuje se přesně opačně. Autor vychází z obsahu, který má dokument sdělit, a jeho úkolem je jej co nejlépe strukturovat. Určit pořadí jednotlivých částí, jejich hierarchii, vzájemné vazby a vhodně označit typizované úseky (adresa, citát atd.). Cílem je, aby byl výsledný dokument přehledný, začínal nejdůležitějšími informacemi a postupoval dále k méně důležitým, aby jeho celková struktura odpovídala sdělovanému obsahu. Používá se k tomu jen „čistý" jazyk (X)HTML, který zaručuje opravdu univerzální funkčnost výsledného dokumentu — přehlednost, nasazení v rámci sémantického webu, přenositelnost dat, snadnou konfigurovatelnost stránek, použitelnost na libovolném koncovém zařízení atd.

Až s takovýmto dokumentem pracuje grafik. Pokud potřebuje, jednotlivé značky v kódu *pojmenuje* či zařadí do *tříd,* případně může uzavřít některé celky do „neškodných" značek `<div>` a `<span>`, které nemají vliv na strukturu ani obsah dokumentu. Prezentaci dokumentu pak definuje pomocí tabulek kaskádových stylů, které se k dokumentu připojí. V jediné tabulce stylů má možnost současně určit podobu skupiny dokumentů (třeba celé website najednou), definovat různé styly pro různá zařízení (jiný pro obrazovku, jiný pro tiskárnu, zvukový styl pro čtecí zařízení atd.). Pouhou záměnou souborů může okamžitě změnit podobu celého webu. Správa webu je snazší a může být i bezpečnější: administrátor např. nastaví externímu grafikovi přístupová práva pro editaci souborů CSS, aniž by mu přitom povolil přístup k samotným dokumentům s obsahem.

1.2.1.3 Co je lepší?

Nebudeme zde posuzovat, zda je jeden přístup *špatný* a druhý *dobrý.* Každý autor má jiné nároky a jim přizpůsobuje i svůj styl práce. Můžeme pouze říci, že první způsob je *starý* a ten druhý *nový.* **Vývoj HTML a s ním i podpora starých metod byly ukončeny.** Budou sice nadále podporovány v prohlížečích a není důvod je nepoužívat napořád, autoři se tu ale více nedočkají žádných vylepšení a inovací. Všechny nové technologie a standardy už vycházejí z XHTML, postaveného výhradně na novém stylu tvorby dokumentů.

Není však nezbytné se okamžitě rozhodovat „ostrým řezem", buď—anebo. Nové standardy připravuje rozsáhlý tým praktiků z celého světa a ti vědí, že nemohou náhle nutit autory k tak radikální změně. Každý má příležitost dopracovat se k ní postupně — je to důvod, proč byly vytvořeny *přechodové (transitional)* typy dokumentů HTML 4 a XHTML 1. Tyto verze jsou určeny právě pro usnadnění přechodu na nový způsob práce. Zachovávají mnohé ze starých konstrukcí a současně přidávají většinu z nových technologií. Kvůli některým principiálním konfliktům však musela být část ze starších „přívažků" z HTML odstraněna a obdobně není možné použít úplně všechny funkce z verzí nových.

I tak jsou tyto *přechodové verze* pro webdesignery velice užitečnou pomůckou. Všechny nové prohlížeče již rozpoznávají *typ dokumentu* a pokud je definován jako *přechodový (transitional),*

přepnou se samy také do *přechodového režimu,* kdy tolerují většinu dříve běžných a dnes již nestandardních konstrukcí. Současně v rámci možností zpracují i rozšíření použitá z nových standardů. Autor tak má možnost na stránkách postupně odkrývat a zkoušet možnosti XHTML, DOM či CSS a přitom není nucen k příliš radikálním změnám. A až si jejich použití dostatečně ozkouší a osvojí si jiný styl práce, pouhou změnou typu dokumentu na *striktní (strict)* přepne režim prohlížeče, který pak pracuje už jen podle nových standardů a staré chyby více netoleruje.

Každý webdesigner by však měl mít stále na paměti, že se jedná se o dva diametrálně rozdílné přístupy k tvorbě WWW stránek — chce-li plně využít možností všech technologií souvisejících s dnešním i budoucím webem, je třeba se také podřídit jejich požadavkům: (X)HTML popisuje pouze strukturovaný obsah a vše ostatní musí být umístěno mimo něj. Jiné postupy těmto předpokladům nevyhovují. Používá-li proto autor ve stránce ještě jakékoli (i sebemenší) formátování pomocí HTML, měl by používat *přechodový typ dokumentu (Transitional),* nebo dokonce nějakou starší verzi HTML. *Striktní dokumenty (Strict)* musí v souladu se svým názvem striktně dodržovat použité standardy — včetně zásad, které byly zmíněny výše.

1.2.2 Jenže, udělám „tohle“ s CSS?

Často opakovanou otázkou je, zda lze pomocí CSS vytvořit stejné stránky jako s pomocí dřívějšího HTML-formátování. Obecně se dá říci, že kaskádovými styly můžeme zformátovat dokumenty takřka libovolně. Schopnosti CSS vysoce předčí možnosti formátování jen pomocí značek HTML a lze je přirovnat k možnostem nejnovějších editorů typu MS Word. Blíží se dokonce schopnostem aplikací DTP, určených k profesionální sazbě tiskovin. Některé z postupů používaných v sazbě však přesto zatím k dispozici nejsou (sloupcová sazba, oboustranné obtékání obrázků, správa barev atp.), ale i ty jsou již postupně zapracovávány do nových specifikací CSS. V právě připravovaném návrhu CSS3 najdeme nejen zmíněnou sloupcovou sazbu, správu barev a obtékání, ale mnoho dalších vymožeností, které z CSS dělají opravdu mohutný designérský i aplikační nástroj.

Na druhé straně je namístě připomenout, že specifikace CSS a realita v praxi nejsou vždy totéž. I když nové verze prohlížečů postupně zahrnují stále více funkcí ze standardu CSS, žádný z nich je ještě nepodporuje plně. Ty nejnovější prohlížeče často víceméně vyhovují první verzi CSS1, ale z CSS2 podporují jen část. Navíc je mezi uživateli stále jisté zastoupení starších prohlížečů, které CSS nepodporují vůbec, nebo jen z malé části. A často také (což je nejhorší) interpretují některé funkce CSS chybně. To je realita, s níž musíme při návrhu stránek neustále počítat. Dobrý webdesigner připravuje stránky tak, aby dobře fungovaly i bez použití stylů. Navíc má přehled o poměrných zastoupeních jednotlivých prohlížečů v cílové skupině svých čtenářů a styly používá tak, aby případné chyby prohlížečů nepůsobily překážky ve vnímání obsahu. Několik postupů, jak se s různými skupinami prohlížečů vypořádat a jak se vyhnout nejznámějším chybám najdete i ve poslední kapitole. Navíc v této knize naleznete i přehledy úrovně podpory jednotlivých funkcí CSS v nejznámějších prohlížečích.

1.2.2.1 Ukázka možností CSS

Co tedy CSS opravdu dokáže? Lepší obrázek si lze patrně udělat po přečtení této knihy a hlavně po mnoha pokusech a experimentování s používáním stylů. První představu si ale můžeme načrtnout již nyní.

Následující ukázku tvoří jednoduchý a čistě strukturovaný dokument, vyhovující *striktnímu typu dokumentu* HTML 4.01. V kódu je určena pouze základní struktura (`<H1>` titulek, `<H4>` mezititulek, `<P>` odstavec atd.), pro potřeby dalšího formátování sem navíc byly postupně přidány „neškodné“ značky `<DIV>` a `<SPAN>`. Stránka v základním formátování (bez použití stylů) bude v běžném prohlížeči vypadat například takto:

Obr. 1 — Stránka se základním formátováním (beze stylů)

Takový dokument neobsahuje nic „nepatřičného“ a stejně či obdobně bude zobrazen **v každém prohlížeči** — včetně těch textových (např. Lynx). Navíc je snadno čitelný i při automatickém zpracování, lze jej dobře indexovat ve vyhledávačích, čtecí zařízení s ní nebude mít potíže, uživatel není obtěžován nadbytečnými informacemi a stránka je minimalizovaná co do objemu dat. Taková stránka by měla být vždy primárním cílem každého autora.

Většina autorů ihned namítne, že tento dokument možná obsahuje dobře uspořádané a přehledné informace, ale rozhodně nevypadá nijak k světu a s něčím takovým by se chlubit nechtěli. Ale měli by — výhod jsme už zmínili dostatek.

A že vypadá tak nějak *nijak,* mdle, nezajímavě? To, jak je stránka zobrazena v prohlížeči, již není otázkou jejího kódu. Důsledné oddělení obsahu stránky od popisu prezentace je alfou a omegou nového pohledu na strukturované dokumenty — definování podoby prvků je

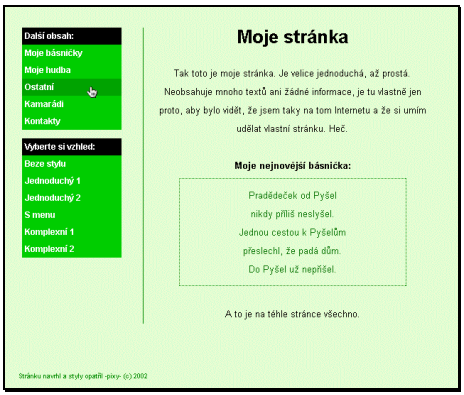
samostatnou, nezávislou informací, která do stránek více nepatří a pro kterou poskytují mohutný nástroj kaskádové styly (CSS). Pomocí nich můžeme určit formátování do nejmenších detailů. Do dokumentů samotných musíme umístit hlavně kvalitní obsah a přehlednou strukturu. O jejich budoucí podobu se přitom vůbec nebudeme nestarat. To je to první, čím by měl každý autor začít.

Pro naši ukázkovou stránku jsem připravil několik různých stylů pro prezentaci na obrazovce, abychom si ukázali, jak odlišně může výsledná stránka vypadat. A také aby bylo zřejmé, že jednoduchý obsah stránky nemá nic společného s její konečnou, třeba i komplikovanou a graficky výraznou podobou.

Nejprve si ukážeme použití dvou jednodušších stylů, které pouze popisují podobu jednotlivých oblastí stránky — typ a styl písma, barvy textu a pozadí, rámečky, styly odstavců. Oba styly jsou prakticky shodné, liší se pouze v použitých barvách:

Obr. 2 a 3 — Stránka zformátovaná s použitím dvou jednoduchých stylů

Takovéto styly jsou jednoduché na použití a jsou ideální pro autory, kteří s CSS teprve začínají. To ale zdaleka není vše, co CSS umožňuje. Prvky stránky mohou být zformátovány úplně jinak, než by byly bez použití stylů — z úseků textu se mohou stát obdélníkové bloky, z bloků naopak části textu; prvky mohou být zobrazeny na jiném místě stránky, než kde jsou zapsány v kódu HTML. V následujícím stylu je hlavní obsah stránky odsunut doprava, do levého horního rohu je umístěna navigace (která se v dokumentu nachází až za obsahem) a je zformátována jako jednoduché menu:



Obr. 4 — Stránka zformátovaná s navigačním menu

A pochopitelně lze jít dál — prvky stránky můžeme zcela přeskupit a pomocí CSS dokonce přidat obsah, který původní dokument vůbec neobsahuje (obrázky, značky u odkazů atd.) a dát tak své prezentaci vlastní výraz, vytvořit z ní dílo přesně podle svého záměru, vkusu a grafického cítění:

Obr. 5 a 6 — Stránka s komplexním formátováním pomocí CSS

Rád bych, aby bylo zřejmé, že **ve všech uvedených ukázkách se jedná stále o tentýž dokument.** Nebyl nijak změněn, do jeho obsahu se nezasáhlo — pouze k němu byl připojen pokaždé jiný soubor s kaskádovými styly.

Poznámka

Tuto ukázku si můžete sami vyzkoušet v prohlížeči. Adresu najdete v Referencích na konci knihy (Příklad 1). Pro ověření, že se jedná stále o stejnou stránku, jsou do dokumentu načteny všechny tabulky stylů najednou jako *alternativní styly* a pokud váš prohlížeč podporuje jejich přepínání (např. Mozilla), můžete jednotlivé styly postupně aktivovat a vidět okamžitou změnu vzhledu stránky.

Pokud mezi alternativními styly přepínat nedokážete, stránka obsahuje i skript, který jednotlivé tabulky stylů aktivuje. Stačí klepnout na příslušnou položku z nabídky „Vyberte si vzhled“ ve stránce. A konečně, pokud nemůžete použít ani Javascript, po klepnutí na odkaz se načte nová stránka, vždy s týmž obsahem a odlišným stylem. Na stránce s příklady najdete odkazy i na tyto samostatné stránky.

Určitě ale zjistíte, že v každém prohlížeči bude stránka zobrazena trochu jinak. Je to ukázka zmíněné nekompatibility a různé úrovně podpory CSS v prohlížečích. Některé prohlížeče interpretují určité definice chybně a stránka tak v důsledku takových chyb softwaru může vypadat ošklivě až nečitelně. Úmyslně jsem ale nepoužil žádný z postupů, které tyto chyby dokáží obejít či omezit (takové postupy naleznete i v této knize) a připravil styly přesně podle specifikace CSS. Každý si tak může ověřit, jaké chyby jeho prohlížeč dokáže udělat. Příklad však funguje dobře v každém prohlížeči, který dodržuje standardy HTML a CSS. Takovým je např. Mozilla 1 a vyšší (výše uvedené obrázky pocházejí z Mozilly 1.0), nebo Internet Explorer 6, v nichž byste měli mít ukázky zobrazeny shodně či velmi podobně.

2 Formátování s CSS v kostce

2.1 Ukázková stránka

V této kapitole si na jedné ukázkové stránce představíme základní možnosti formátování s pomocí kaskádových stylů. Podrobný popis všech funkcí a postupů pak naleznete ve druhé kapitole.

2.1.1 Výchozí ukázka bez použití stylů

Pro ukázku jsem připravil typickou stránku fiktivní firmy, podobných lze kdekoli na webu najít stovky (všechny údaje jsou smyšlené, jakákoli podobnost se skutečností je pouze náhodná).

Obr. 7 — Ukázková stránka, formátovaná pouze pomocí HTML

Pomiňme vhodnost obsahu stránky — časté klišé *„Vítejte na našich stránkách“* a *počítadlo* jsem sem umístil záměrně. Možná vás během práce s touto prezentací napadne, jakým titulkem byste uživateli mnohem lépe uvedli její obsah, a zda někoho opravdu zajímá, kolik lidí už stránku vidělo. Věnujme se jejímu kódu.

Stránka byla vytvořena pouze s pomocí jazyka HTML a různých proprietárních konstrukcí, bez použití kaskádových stylů. Rozmístění prvků na stránce je řešeno tabulkami. Rámečky kolem bloků pak zajistilo několik vzájemně vnořených tabulek. Okraje a pozadí stránky, základní barva textu a barvy odkazů jsou definovány atributy v `<body>`. Pro typ, velikost a barvu písma jsou použity značky `<font>`. Menu na levé straně je vytvořeno pomocí obrázků, dynamické efekty zajišťují události a funkce Javascriptu. Pro zarovnání textu a obtékání obrázku byl použit atribut `align`. Stránku i její kód najdete na Internetu jako *Příklad 2a* (viz [Reference]).

Jak si sami můžete ověřit, ačkoli je tato stránka celkem jednoduchá a není ani příliš rozsáhlá, její kód už není moc přehledný. Nebude snadné se v ní vyznat, až budeme za pár týdnů potřebovat něco upravit. Pokud bychom třeba chtěli změnit bílý rámeček kolem „novinek“ v pravé části, museli bychom nejprve najít, které tabulky jej tvoří a nakonec změnit pozadí střední vnořené tabulky na řádku 76. Změna položky menu „O FIRMĚ“ například na „O NÁS“ obnáší vytvoření dvou nových obrázků v grafickém editoru (pro dva stavy) a jejich export do vhodného formátu. Kód stránky tvoří asi 5,5 KB (5.511 znaků) textu včetně nezbytného javascriptového kódu.

2.1.2 Doladění vzhledu pomocí CSS

Z pohledu běžné praxe se jedná o normálně fungující stránku a uvedené řešení je stále běžné. Dokument je deklarován jako *přechodová* verze HTML 4 *(HTML 4.0 Transitional),* nic nám tedy nebrání používat i novější technologie, které tato verze poskytuje. Například využít kaskádových stylů a některá formátování s jejich pomocí zjednodušit.

Můžeme odstranit vnořené tabulky a definovat rámečky pomocí CSS. Kaskádovými styly nastavíme okraj stránky, vzhled pozadí a výchozí styl písma pro celý dokument, čímž se zbavíme mnoha zbytečných atributů i všech značek `<font>`. Můžeme definovat zarovnávání textu i vytvořit obtékaný obrázek, tím se stanou nepotřebnými také atributy `align`.

Tak bychom mohli pokračovat dál — až do chvíle, kdy už nám nezbudou žádné nepovolené, *přechodové* značky a atributy, kdy už nebude co nahrazovat. Zůstala by stránka vyhovující i *striktní verzi* HTML 4. Avšak pouze formálně. Tímto reverzním postupem sice **získáme bezchybný kód, ale nikdy nevytvoříme strukturovanou stránku.** V uvedené ukázce by zůstal obsah stránky rozdělen do tří sloupců tabulky. Čtecí zařízení by takový dokument mohlo interpretovat třeba jako: „O firmě, vítejte na našich stránkách! Novinky: Katalog…“ Před hlavním sdělením jsou umístěny méně podstatné informace, např. navigace. Vyhledávače by nebyly schopny rozeznat, co je nejdůležitějším obsahem stránky.

Jediný způsob, jak vytvořit strukturovaný dokument, je začít úplně od začátku, s „čistým“ obsahem. Abychom si mohli ukázat všechny možnosti CSS, budeme se tohoto postupu držet i zde.

2.1.3 Základem pro formátování je dobrý dokument

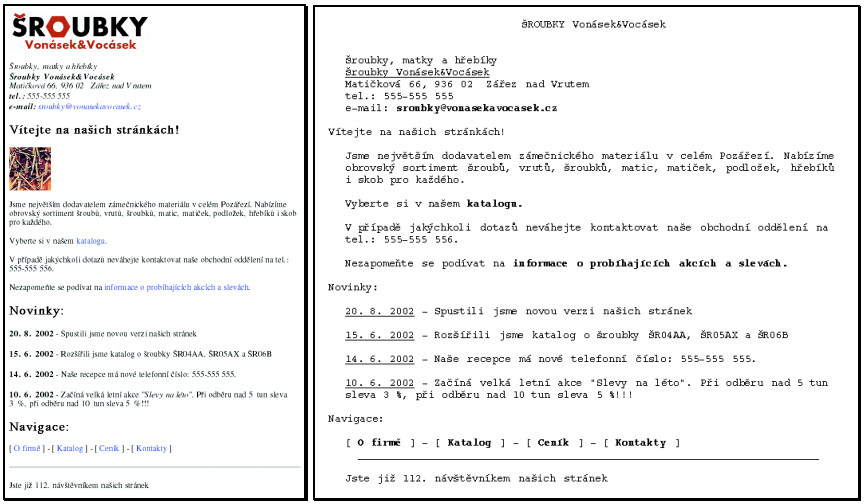
Chceme-li začít s formátováním kaskádovými styly od úplného počátku, vytvoříme nejprve strukturovaný dokument, který nijak nezohledňuje budoucí prezentaci stránky. Popíšeme v něm pouze uspořádání obsahu, jeho rozdělení do logických celků, posloupnost a hierarchii informací, které má výsledná stránka sdělit.

Obsah z původní ukázky můžeme uspořádat například do takovéhoto dokumentu *(Příklad 2b):*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/1999/REC-html401-19991224/strict.dtd">
<html>
<head>
    <title>Šroubky Vonásek&amp;Vocásek</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
    <meta name="author" content="Gorila Design, 2002 (copyleft) Safari nad Labem">
    <meta name="copyright" content="Šroubky Vonásek&amp;Vocásek (L) 2002">
</head>

<body>

    <h1><img src="img/logo.gif" width="270" height="80" alt="SROUBKY Vonasek&amp;Vocasek"></h1>

    <address>
        Šroubky, matky a hřebíky<br>
        <strong>Šroubky Vonásek&amp;Vocásek</strong><br>
        Matičková 66, 936 02  Zářez nad Vrutem<br>
        <strong>tel.:</strong> 555-555 555<br>
        <strong>e-mail:</strong> <a href="mailto:sroubky@vonasekavocasek.cz">sroubky@vonasekavocasek.cz</a>
    </address>

    <h2>Vítejte na našich stránkách!</h2>

    <img src="img/vruty.jpg" width="80" height="83" alt="">

    <p> Jsme největším dodavatelem zámečnického materiálu v celém Pozářezí.
        Nabízíme obrovský sortiment šroubů, vrutů, šroubků, matic, matiček,
        podložek, hřebíků i skob pro každého.
    </p>
    <p>Vyberte si v našem <a href="katalog.html">katalogu</a>.</p>
    <p>V případě jakýchkoli dotazů neváhejte kontaktovat naše obchodní oddělení na tel.: 555-555 556.</p>
    <p>Nezapomeňte se podívat na <a href="akce.html">informace o probíhajících akcích a slevách</a>.</p>

    <h2>Novinky:</h2>
        <p><strong>20. 8. 2002</strong> - Spustili jsme novou verzi našich stránek</p>
        <p><strong>15. 6. 2002</strong> - Rozšířili jsme katalog o šroubky ŠR04AA, ŠR05AX a ŠR06B</p>
        <p><strong>14. 6. 2002</strong> - Naše recepce má nové telefonní číslo: 555-555 555.</p>
        <p><strong>10. 6. 2002</strong> - Začíná velká letní akce &quot;Slevy na léto&quot;.</em>
            Při odběru nad 5 tun sleva 3 %, při odběru nad 10 tun sleva 5 %!!!</p>

    <h2>Navigace:</h2>
    <p>
        [ <a href="onas.html">O firmě</a> ] -
        [ <a href="katalog.html">Katalog</a> ] -
        [ <a href="cenik.html">Ceník</a> ] -
        [ <a href="kontakty.html">Kontakty</a> ]
    </p>

    <hr>
    <p>Jste již 112. návštěvníkem našich stránek</p>

</body>
</html>
```

Obr. 8 — Kód upraveného dokumentu

Dokument se zobrazí čitelně a přehledně v libovolném prohlížeči, například takto:



Obr. 9 a 10 — Zobrazení upraveného dokumentu v prohlížečích MSIE (vlevo) a Lynx (vpravo)

Informační hodnota stránky je přinejmenším stejná jako u původní verze, sdělovaný obsah je shodný. Informace o autorovi stránek a copyrightu byly přesunuty do značek `<meta>` v hlavičce dokumentu, kam logicky patří. Kvůli přehlednosti byl přidán nadpis „Navigace“ před položky s

odkazy. Naopak byl ze stránky odstraněn obrázek `"img/sroub.jpg"`, který nemá informační hodnotu a ve stránce byl jen coby jako grafický prvek — grafiku přidáme až pomocí stylů. Obrázek `"img/vruty.jpg"` zalomený v textu je stejného druhu a mohli bychom jej rovněž přidat do stránky později. Jak ale záhy zjistíme, funkce CSS, které to zajišťují, ještě nejsou v mnoha prohlížečích podporovány. Proto jej ve stránce prozatím ponecháme.

Objem dokumentu je nyní asi 2 kB (2.225 znaků), což není ani polovina objemu stránky předchozí. Pokud budeme přidávat kaskádové styly, musíme k tomu jejich velikost připočíst. Jestliže je však umístíme do samostatného souboru, budou se obvykle načítat pouze jednou. Při další návštěvě už je uživatelé načtou z *vyrovnávací paměti* na svém disku či *proxy serveru.* Když zaktualizujeme obsah stránky, uživateli postačí načíst jen hlavní dokument, nezměněné styly ze serveru podruhé stahovat nemusí a vice versa. Změníme-li soubor se styly, nemusí se už načítat stránka samotná.

Takto připravený dokument už můžeme formátovat pomocí kaskádových stylů.

2.2 Připojení stylů do stránky

Do stránek HTML lze kaskádové styly připojit více způsoby. Chceme-li předepsat styl pouze jednomu konkrétnímu prvku, můžeme napsat definici stylu **přímo do jeho značky,** jako hodnotu atributu `style="..."`. Pro celý dokument můžeme použít **tabulku stylů (stylesheet) zapsanou do značky** `<style>` nebo **načíst externí tabulku značkou** `<link>`**.**

—> Použití CSS v dokumentech [3.1]

Definování stylů přímo ve značce je neefektivní a neposkytuje tolik možností. **Přednostně se používají tabulky stylů.** *Striktní dokumenty* podle nejnovějších standardů již používají pouze *tabulky* a navíc se doporučuje dávat přednost externím souborům se styly. Budeme se proto držet modernějšího přístupu a styly pro naši ukázkovou stránku umístíme do samostatného souboru, který k dokumentu připojíme.

Soubory s tabulkami stylů je obvyklé pojmenovávat s příponou `".css"`. V libovolném textovém editoru vytvoříme nový soubor, který uložíme např. pod jménem `"styl.css"` do stejného adresáře (složky) jako je naše stránka. Do něj pak budeme zapisovat všechny styly.

Ke stránce jej připojíme pomocí značky `<link>`. Ta musí být umístěna v hlavičce dokumentu (mezi `<head>` a `</head>`) a musí mít určeny přinejmenším atributy `rel`, `type` a `href`. V našem případě proto použijeme:

```
<head>
...
<link rel="stylesheet" type="text/css" href="styl.css">
...
</head>
```

Při dalším otevření stránky už se prohlížeč bude snažit tento soubor načíst a použít jeho obsah jako tabulku stylů. Pokud jej nenajde nebo soubor nebude obsahovat použitelnou tabulku, nic se nestane. Příkaz bude ignorován a stránka se zobrazí beze stylů.

2.3 Definování stylů v tabulce

Do tabulky stylů budeme zapisovat **pravidla.** V každém pravidle se nejprve uvádí, pro jaké *prvky* stránky má pravidlo platit, a následně se do složených závorek zapíše předpis stylu. Tabulku stylů potom tvoří **seznam takových pravidel.**

—> Syntaxe CSS [3.2]

2.3.1 Přiřazení hodnot vlastnostem

Každý prvek HTML má několik desítek **vlastností** popisujících podobu jeho prezentace. Vlastnosti jsou rozděleny do několika skupin — určují vzhled písma a textu, rozměry rámečků, barvy obsahu i pozadí, pozici prvku vůči ostatním, vlastnosti zvuku pro čtecí zařízení a další. Každá vlastnost má jednoznačný **název** a má určeny povolené **hodnoty** i svou *výchozí hodnotu.* Ta se použije pokud vlastnost sami nedefinujeme — není proto nutné jmenovitě popisovat všechny vlastnosti všech prvků. Stačí pouze změnit ty vlastnosti, které se mají lišit od výchozí hodnoty.

V definici přiřazujeme vlastnosti hodnotu pomocí zápisu `vlastnost:hodnota`. V pravidle můžeme uvést více definic najednou, jednotlivé definice oddělujeme středníkem. Např.: `color:red;␣font-size:small;␣margin:0`.

—> Hodnoty a jednotky v CSS [3.3]

Některé vlastnosti přímo nepopisují podobu prvku, ale slouží pro současné definování více vlastností. Nazývají se **sdružené vlastnosti** (též *zkratky*) a jejich hodnotou bývá seznam dílčích hodnot jednotlivých vlastností, které tato *zkratka* sdružuje.

—> Sdružené vlastnosti [3.2.2.2]

2.3.2 Přiřazení definic prvkům stránky

Prvky, pro které má definice platit, popisují **selektory.** Je jich velké množství, zatím si vystačíme se třemi nejjednoduššími.

Selektorem může být **název prvku,** např. `p`, `body`, `h1` atd., pravidlo se pak vztahuje na všechny prvky tohoto typu. Např. `h1␣{␣color:red␣}` definuje vlastnost `color` všem prvkům `<h1>`.

Druhou možností je **prvek dané třídy** *(class)*. Zapisuje se s tečkou za jménem prvku, např. `h1.nadpis`, `p.poznamka`. Takové pravidlo se vztahuje na prvky, které jsou daného typu a mají navíc danou třídu, tedy `<h1␣class="nadpis">`, resp. `<p␣class="poznamka">`. Prvků stejné třídy může být na stránce libovolné množství. Pokud v selektoru vynecháme jméno prvku, bude

pravidlo platit pro všechny prvky s touto třídou, např. `.poznamka` se bude vztahovat na *jakýkoli prvek,* který má `class="poznamka"`.

Třetím z nejpoužívanějších selektorů je **prvek s daným pojmenováním.** Každý prvek na stránce můžeme pojmenovat pomocí atributu `id` (pozor, jméno z atributu `name` nelze použít), např. `<div id="zahlavi">`. Na stránce se nesmí vyskytovat dva prvky se stejným `id`, toto pojmenování je tedy jednoznačné a pravidlo bude platit jen pro jediný prvek. Selektor se jménem prvku se označuje dvojkřížkem „`#`“, např. `#zahlavi`, `#podpis`, `#prvek123` atd.

2.3.2.1 Pravidla

S těmito (a mnoha dalšími) selektory vytváříme **pravidla,** jimiž přiřazujeme definice **všem prvkům, které vyhovují použitému selektoru.** Pravidlo se zapisuje ve tvaru `selektor { definice }`, např.:

```
h1.nadpis { color:red; font-size:xx-large }
```

Pokud definujeme styl přímo ve značce prvku, selektor se již nepoužívá. Do atributu `style` se zapíší přímo definice stylu, např. `<h1 style="color:red;font-size:xx-large">`.

—> Selektory a jejich používání [3.6]

2.3.3 Seznam pravidel tvoří tabulku stylů

Každá tabulka stylů je tvořena **seznamem pravidel.** Pravidla se v tabulce uvádějí za sebou, pro přehlednost je vhodné psát každé na samostatný řádek. Prohlížeč pravidla prochází a pokud najde prvek, který vyhovuje použitému *selektoru,* přiřadí jeho vlastnostem zadané hodnoty.

V rozsáhlejších tabulkách se mohou hodit i **komentáře.** V CSS se nepoužívají komentáře jazyka HTML (`<!-- -->`), ale syntaxe známá spíše z programovacích jazyků: `/* komentář */`.

Typická tabulka stylů může vypadat např. takto:

```
/* moje styly */
body { background-color:white; margin:25pt; }
p { font-size:12pt; line-height:150% /* odstavec */ }
p.poznamka { color: blue } /* barva pro poznámky */
#zahlavi { background-color:yellow }
```

2.3.4 Další tabulky stylů a dědičnost

Jak jsme si již řekli, pokud prvek nemá nějakou vlastnost definovánu, použije se její **výchozí hodnota.** Musíme ale počítat i s tím, že ne všechny definice můžeme vidět.

Kromě našich tabulek stylů existují i tabulky další, pro nás skryté. Vlastní tabulku stylů může používat **uživatel,** jeho styly se podle přesných pravidel spojí se styly definovanými na stránce.

Svou tabulku stylů má povinně také každý **prohlížeč** — pomocí ní formátuje prvky, které nemají žádný styl určen. Je v ní například řečeno, že nadpisy `<h1>` mají být dvakrát větší než základní písmo stránky, mezi odstavci `<p>` jsou mezery, značky `<b>` a `<strong>` se zobrazují tučně atd. Příklad takové výchozí tabulky stylů najdete také na konci této knihy.

Ale ani když prvek vlastnost nemá definovánu ani v jedné z těchto tabulek, nemusí ještě použít svou *výchozí hodnotu.* Některé vlastnosti CSS jsou totiž **dědičné,** což znamená, že nejsou-li nikde definovány, převezmou hodnotu od svého *nadřazeného prvku* (tedy prvku, uvnitř nějž leží). To je velmi užitečná funkce CSS. Díky dědičnosti můžeme některé *vlastnosti* nastavit jen některým prvkům a tuto hodnotu pak použijí i všechny prvky uvnitř nich. Např. typ písma můžeme definovat jen pro značku `<body>` (v ní je umístěn celý obsah stránky) a stejnou hodnotu pak použijí všechny prvky na stránce. Teprve když některému z nich vlastnost nadefinujeme jinak, použije on i jeho *potomci* (prvky umístěné uvnitř něj) novou hodnotu. Platí zde pravidlo „bližší košile než kabát“ — pokud se hodnota dědí, dědí se vždy z nejbližšího nadřazeného prvku.

—> Strom dokumentu [3.5]

V popisu jednotlivých vlastností ve druhé kapitole je vždy výslovně uvedeno, zda se daná vlastnost dědí, či nikoli. Obecně se ale dá říci, že vesměs **jsou dědičné vlastnosti textu** a **ostatní vlastnosti obvykle dědičné nejsou.**

Z těchto zásad CSS tedy vyplývá, že prohlížeč nejprve hledá hodnotu vlastnosti v tabulkách stylů (ve všech, které má k dispozici). Pokud ji nenajde, může hodnotu převzít z nadřazeného prvku (je-li vlastnost dědičná). Jen když hodnotu nenajde a vlastnost není dědičná, teprve použije výchozí hodnotu.

—> Zdroje definic stylů a dědičnost [3.7]

2.3.5 Kaskáda

Prvky také mohou mít stejnou vlastnost definovanou současně na více místech. Výběr té správné hodnoty, která se má použít, zajišťuje **kaskáda CSS.** Platí především zásada, že **přednost mají přesnější, konkrétnější pravidla před pravidly obecnějšími.** Pokud dvě pravidla definují jednomu prvku stejnou vlastnost, použije se to, jehož selektor popisuje prvky **konkrétněji.** Například selektor s třídou definuje styl *jen pro některé prvky* daného typu, je konkrétnější. Pravidla s třídou proto mají přednost před pravidly bez třídy. Selektor se pojmenováním (id) prvku platí dokonce jen pro jediný konkrétní prvek, má proto prioritu ještě vyšší. Použijeme-li např. tabulku stylů:

```
p { color:blue }
p.poznamka { color:red }
#pozn1 { color:green }
```

bude prvek `<p class="poznamka" id="pozn1">` vyhovovat všem třem pravidlům. Vlastnost `color` (barva textu) je definována třikrát, pokaždé jinak. Selektor `p.poznamka` je konkrétnější než `p`, druhé pravidlo má tedy přednost před prvním. Třetí pravidlo ale používá `id` prvku, je ještě konkrétnější a bude mít nejvyšší prioritu. Použije se proto definice `color:green` z třetího pravidla a prvek bude zobrazen zelenou barvou textu.

V případě, že se sejdou **stejně konkrétní definice** téže vlastnosti, záleží na jejich pořadí. Použije se ta, která je zapsaná nejpozději. Například:

```
p { color:black }
p { color:blue }
p { color:yellow; color:red }
```

Pravidla jsou stejně konkrétní, záleží jen na jejich pořadí — pro vlastnost `color` se proto použije poslední pravidlo a v něm poslední definice, tedy `color:red`.

Styly definované přímo ve značkách prvků (atributem `style`) mají nejvyšší prioritu a tedy i přednost před definicemi pocházejícími z tabulek stylů.

—> Pravidla kaskády CSS [3.7]

2.4 Formátování textu

S definováním vzhledu naší ukázkové stránky začneme u textu. S CSS máme stejné možnosti, které poskytuje původně použitá značka `<font>` a navíc je k dispozici mnoho dalších vlastností, které prostředky HTML nebylo možné vůbec popsat.

2.4.1 Typ písma

V CSS se typ písma určuje vlastností `font-family`. Její hodnotou je **seznam názvů písem.** Položky se oddělují čárkami, **názvy, které obsahují mezeru, musí být uzavřeny v uvozovkách.** Tento seznam se zpracovává zleva — prohlížeč zkusí první definované písmo a pokud jej má k dispozici, použije jej a dál se do seznamu nedívá. Pokud jej použít nemůže, zkusí druhé v pořadí, pak třetí atd. Seznam by proto měl obsahovat nejprve písma zřídka dostupná, poté písma dostupnější obecněji. Jako poslední by mělo být vždy uvedeno jeden z *obecných typů písma* definovaných CSS.

—> V našem jazykovém prostředí je problematika definování typu písma poněkud složitější [5.2].

CSS definuje pět *obecných typů písma.* V prohlížeči se pak nahrazují konkrétním *fontem* podle nastavení programu či volby uživatele. Jsou to `serif` (patkové písmo, např. rodiny písem Times, Bodoni apod.), `sans-serif` (bezpatkové písmo, typ Helvetica, Arial), `cursive` (kurzíva či kaligrafické písmo, typ Chancery, SnellRoundhand apod.), `monospace` (neproporční písmo, např. typ Courier) a `fantasy` (dekorativní písmo, příp. to, které nelze zařadit do ostatních skupin).

Typické definice vlastnosti `font-family` vypadají např. takto:

```
font-family: 'Times CE','Times New Roman CE','Times New Roman',serif;
font-family: "Arial CE","Helvetica CE",Arial,sans-serif;
font-family: 'Courier CE','Courier New CE','Courier New',monospace;
font-family: 'Poptics One', 'Funny Fellow', MerryXmass, fantasy;
font-family: cursive;
```

V původní stránce jsme všem textům předepsali bezpatkové písmo pomocí značek `<font>`. S pomocí stylů je budeme definovat vlastností `font-family`. Postačí přitom použít jen *obecný typ,* který se v prohlížeči nahradí vhodným písmem. A protože jsou vlastnosti písma dědičné, nemusíme písmo definovat každému prvku. Chceme-li písmo stejné na celé stránce, nadefinujeme `font-family` pro `<body>` a ostatní prvky potom tuto hodnotu převezmou *(zdědí).* Do naší zatím prázdné tabulky stylů tedy můžeme zapsat první pravidlo, kterým definujeme typ písma pro celou stránku:

```
body { font-family: sans-serif }
```

Načteme-li dokument znovu, uvidíme, že všechny texty jsou nyní zobrazeny bezpatkovým písmem.

2.4.2 Velikost písma

Velikost písma se určuje vlastností `font-size`. Jako její hodnota se používají klíčová slova nebo číselné hodnoty s jednotkou (např. `12pt`), případně procenta.

—> Velikost písma [4.4.1.5]; jednotky v CSS [3.3]

Všechny texty na stránce by neměly mít stejnou velikost. Pouhou dědičností z `<body>` to beztak ani nemůžeme předepsat: mnoho prvků má totiž definovánu velikost písma ve *výchozí tabulce prohlížeče.* V ní je například určeno, že nadpisy `<h1>` jsou dvakrát větší než písmo stránky, `<h2>` jsou větší 1,5krát atd. Pokud tedy neurčíme jejich velikost ve *své tabulce stylů,* použijí se pravidla z tabulky prohlížeče a na dědičnost se už nedostane řada.

Ale kromě nadpisů už u většiny ostatních prvků nemá prohlížeč velikost písma definovánu. Můžeme tedy považovat velikost definovanou pro `body` za velikost *základní,* od níž se některé prvky budou odvozovat (např. zmíněné nadpisy). Většina prvků na stránce, především texty odstavců, tuto velikost použijí beze změny. Pro naši stránku použijeme základní velikost písma `12pt`, do tabulky stylů tedy přidáme novou definici:

```
body {
      font-family: sans-serif;
      font-size: 12pt
      }
```

Nadpisy `<h2>` jsou pro naše potřeby příliš velké, přidáme pro ně pravidlo definující jim velikost třeba jen `14pt`. Adresu na začátku stránky chceme zobrazit ještě menším písmem — je uzavřena ve značce `<address>`, takže pro ni nadefinujeme písmo velikosti `10pt`:

```
h2 { font-size: 14pt }
address { font-size: 10pt }
```

Také v odstavci s „počítadlem“ na konci stránky budeme chtít změnit velikost písma. Abychom mu mohli definovat styl, musíme jej nejprve nějak odlišit od ostatních odstavců. Proto si jej pojmenujeme. Můžeme mu přiřadit nějakou třídu, ale protože podobné prvky se na stránce už nevyskytují, bude vhodnější pojmenovat jej pomocí `id`:

```
<p id="pocitadlo">Jste již ...</p>
```

Nyní pro něj můžeme přidat pravidlo do tabulky:

```
#pocitadlo { font-size: 9pt }
```

Jako poslední bychom chtěli zmenšit velikost písma „novinek“. To už bude poněkud obtížnější. V kódu se nevyznačují žádnou speciální značkou, abychom tento úsek mohli odlišit v tabulce stylů od ostatního obsahu. Máme ale přinejmenším dvě možnosti.

Všem odstavcům v novinkách můžeme přiřadit stejnou třídu, např. `<p class="novinky">` a definovat jí styl: `p.novinky { font-size:10pt }`. Protože ale později budeme chtít celému bloku novinek také přiřadit také jiné pozadí a další vlastnosti, bude nyní výhodnější druhá možnost.

Celou oblast totiž můžeme uzavřít do nějaké „neškodné“ značky (bezpříznakového bloku), která neovlivní strukturu stránky. Pro obdélníkové bloky je to značka `<div>` (pro úseky textu značka `<span>`). Důležité je pouze dodržet pravidlo, aby se v kódu značky nikdy nekřížily — např. konstrukce `<div><p></div></p>` je chybná.

Úsek, který tvoří „novinky“ uzavřeme do značky `div`, kterou pojmenujeme `"novinky"`:

```
<div id="novinky">
<h2>Novinky:</h2>
      <p><strong>20. 8. 2002</strong> - Spustili jsme...
      ...
      ... </p>
</div>
```

Na takto pojmenovaný úsek se již můžeme ve stylech odkazovat a přiřadit mu požadovanou velikost:

```
#novinky { font-size: 10pt }
```

Všechny prvky uvnitř `<div id="novinky">` nadále nebudou dědit hodnotu `font-size` od prvku `body`. Podle pravidla „bližší košile než kabát“ zdědí hodnotu od nejbližšího prvku a použijí velikost `10pt` definovanou pro blok `"novinky"`.

2.4.3 Barva písma

Na původní stránce byla barva textu určena atributem `text="#666666"` ve značce `<body>`, v posledním odstavci novinek byla pro zvýraznění slevové akce použita i značka `<font color="#0000CC">`. V CSS se pro barvu písma používá vlastnost `color`. Její hodnotou může být hexadecimální kód barvy stejně jako ve výše uvedených atributech.

—> Definování barev v CSS [3.3.6]

Protože vlastnost `color` je *dědičná*, můžeme v prvku `body` definovat výchozí barvu textu pro celou stránku:

```
body {
      font-family: sans-serif;
      font-size: 12pt;
      color: #666666
      }
```

Nyní budou všechny texty (vyjma odkazů) zobrazeny šedou barvou. **Odkazy barvu textu a některé další vlastnosti nepřebírají** a styl pro ně budeme definovat samostatně (viz dále).

Text slevové akce jsme v nové verzi označili jako *zvýrazněný* značkou `<em>`. Chceme-li jej zobrazit modře jako ve verzi původní, můžeme barvu definovat pro prvky `em`, neboť se další takový prvek na stránce už nevyskytuje. To nás však může v budoucnu omezovat. Prvek `em` můžeme chtít později použít i jinde, s jiným formátováním. Můžeme zde proto využít **kontextový selektor.**

CSS umožňuje upřesnit prvky podle kontextu, v němž se nalézají. Tvoří-li selektor **dva prvky oddělené mezerou,** říkáme tím, že pravidlo platí jen pro druhý prvek, který se nachází kdekoli uvnitř prvku prvního. Např. selektor `p em` popisuje takové prvky `em`, které jsou uvnitř nějakého prvku `p`; selektor `div p` popisuje pouze prvky `p` uvnitř nějakého `div`. A podobně selektor `#novinky em` bude definovat jen ty prvky `em`, které jsou uvnitř prvku pojmenovaného `novinky`. A to je přesně to, co potřebujeme. Přidáním pravidla:

```
#novinky em { color: #0000CC; }
```

definujeme modrou barvu jen prvkům `em` v „novinkách“. Případných dalších prvků `em` jinde na stránce se tato definice týkat nebude.

2.4.4 Řez písma

Možná jste si všimli, že váš prohlížeč zobrazil na stránce úvodní adresu i obsah prvku `em` v „novinkách“ skloněným písmem (*italikou*). Je to proto, že *výchozí tabulky stylů* mnoha prohlížečů definují *italiku* právě pro prvky `em` a `address`, které jsme zde použili. My však toto výchozí formátování můžeme změnit — naše tabulka má před styly prohlížeče vždy přednost.

Pro styl (řez) písma se používají vlastnosti `font-style` (sklon písma) a `font-weight` (síla písma). Prvky zobrazené *italikou* mají definováno `font-style:italic`, chceme-li sklon zrušit a nastavit základní, neskloněné písmo, použijeme hodnotu `normal`. Pravidla pro `address` i `em` už v tabulce máme, přidáme do nich jen další definici:

```
address { font-size: 10pt; font-style: normal }
#novinky em { color: #0000CC; font-style: normal }
```

Načteme-li dokument znovu, zjistíme, že prvky `em` a `address` nejsou zobrazeny skloněným písmem.

Vlastnost `font-weight` definuje sílu písma, nejčastěji používanými hodnotami jsou `bold` (tučné písmo) a `normal` (základní, netučné písmo). Hodnota `bold` je typickou výchozí hodnotou pro prvky `strong`, které používáme. Není však nikde řečeno, že to tak musí být — některé prohlížeče mohou text `strong` zvýraznit jinak, třeba právě *italikou.* Chceme-li, aby se prvky `strong` zobrazily vždy neskloněným písmem a tučně, přidáme pro jistotu takové pravidlo do tabulky. Tím zajistíme shodné formátování těchto prvků ve všech prohlížečích.

```
strong { font-style:normal; font-weight:bold }
```

Nakonec přidáme i ztučnění textu slevové akce:

```
#novinky em { color:#0000CC; font-style:normal; font-weight:bold; }
```

2.4.5 Dekorace písma

Pomocí **dekorace písma** (vlastnost `text-decoration`) můžeme text opatřit dalšími efekty. V CSS jsou zatím k dispozici **podtržení** (hodnota `underline`), **nadtržení** (linka nad textem, `overline`), **přeškrtnutí** (`line-through`), **blikající text** (`blink`), případně **žádný** efekt (hodnota `none`). Tyto dekorace se většinou používají jen pro zvýraznění odkazů, např. `a { text-decoration:underline }`. Pro běžný text je obvykle nebudeme potřebovat.

2.4.6 Zarovnávání textu

Obsah prvků je možné vodorovně zarovnat pomocí vlastnosti `text-align`. Ta může mít hodnoty `left` (zarovnání doleva), `right` (doprava), `center` (na osu) a `justify` (zarovnání do bloku). V našem prostředí bývá výchozí hodnotou `left` (v jazycích píšících zprava doleva to je zase hodnota `right`). Pokud tedy neurčíme jinak, text bude zarovnán doleva. Na ukázkové stránce pomocí této vlastnosti zarovnáme záhlaví (logo a adresu) na osu.

Záhlaví zatím nemáme v kódu nijak vyznačeno, ohraničíme jej proto v dokumentu obdobně jako dříve „novinky“ — tentokrát pomocí `<div id="zahlavi">`:

```
<div id="zahlavi">
      <h1><img src="img/logo.gif" ...
      <address>
      ...
      </address>
</div>
```

Do tabulky stylů teď můžeme přidat pravidlo:

```
#zahlavi { text-align: center; }
```

Záhlaví stránky bude vodorovně zarovnané na osu stránky. Obdobně můžeme vycentrovat i počítadlo na konci stránky.

2.4.7 Shrnutí formátování textu

Popsali jsme základní formátování textů na ukázkové stránce. Protože vlastnosti písma jsou vesměs *dědičné,* využili jsme toho a některé vlastnosti definovali více prvkům najednou tak, že jsme přiřadili styl jejich nadřazenému prvku. Když bylo třeba odlišit prvek od ostatních, pojmenovali jsme jej. Některé oblasti stránky jsme uzavřeli do pojmenovaných bloků `<div>`, na něž se pak v tabulce stylů odkazujeme. Tabulka nyní obsahuje tyto definice:

```
body          { font-family: sans-serif;
                font-size: 12pt;
                color: #666666 }
h2            { font-size: 14pt }
address       { font-size: 10pt; font-style: normal }
strong        { font-style: normal; font-weight: bold }
#pocitadlo    { font-size: 9pt; text-align: center }
#novinky      { font-size: 10pt }
#novinky em   { color:#0000CC; font-style:normal; font-weight:bold; }
#zahlavi      { text-align: center; }
```

Takto upravenou stránku i její tabulku stylů najdete na Internetu jako *Příklad 2c* (viz [Reference]).

2.5 Formátování bloků stránky

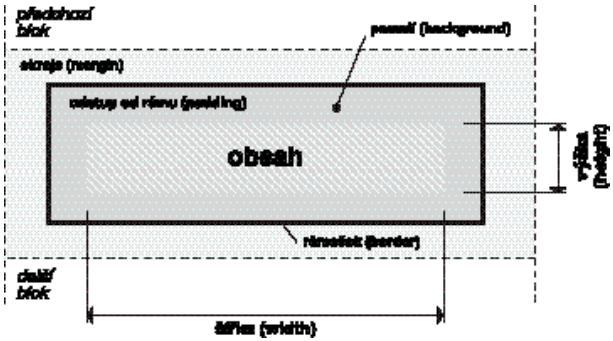
Další vlastnosti můžeme použít pro značky HTML, které na stránce tvoří nějakou obdélníkovou oblast. Nejedná se tedy o *úseky textu* (značky `<em>`, `<strong>`, `<span>`), ale o prvky, které se zobrazují jako samostatný *blok* (v novém odstavci). Typickými představiteli jsou značky `<p>`, `<div>`, nadpisy `<h1>` až `<h6>`, seznamy `<ul>` či tělo dokumentu (prvek `<body>`). **Blokové vlastnosti CSS** umožňují definovat odstupy mezi jednotlivými bloky, jejich rozměry, pozadí a rámečky, dokonce jejich umístění na přesné souřadnice na stránce. Některé z těchto vlastností lze použít i pro neblokové prvky, tam se ale jejich použití může lišit.

Na rozdíl od vlastností textu se blokové vlastnosti **nedědí.** Pokud tedy nějakou vlastnost neurčíme, použije se *výchozí hodnota* platná pro danou vlastnost.

2.5.1 Oblasti bloků

Každý *blok* se skládá z několika oblastí, které postupně „obalují“ jeho obsah.

Obsahem bloku je vše, co je uvnitř značky, která blok definuje — text odstavce, část dokumentu uvnitř `<div>`, obrázek v `<img>` atd. Kolem bloku může být zobrazen **rámeček (border).** Aby nebyl rámeček „nalepený“ přímo na obsah, může mezi nimi být prázdný prostor, určený **výplní,** neboli **odstupem obsahu od rámečku (oblast padding).** Pokud má prvek definováno nějaké **pozadí (background),** zobrazí se uvnitř celého rámečku: pod obsahem, výplní i rámečkem. Vně rámečku je potom průhledný **okraj (oblast margin).** Jeho velikost udává, jaký bude odstup mezi jednotlivými bloky.



Obr. 11 — Oblasti blokových prvků

Za rozměry bloku — **šířku** (width) a **výšku** (height) — se považují pouze rozměry obsahu, **rozměry ostatních oblastí se k nim přičítají.**

—> Výpočet rozměrů bloků [3.8.7]

Pokud není řečeno jinak, **bloky vodorovně zabírají celý prostor, který mají k dispozici.** Definujeme-li jim menší *šířku,* zbylý prostor se rozdělí mezi jejich postranní *okraje.* Menší rozměr a shodné okraje po stranách tak zajistí vodorovné vystředění prvku, nulový okraj na jedné straně pak zarovnání prvku doprava či doleva. Jenom když současně určíme šířku i oba vodorovné okraje, prvek může zabírat menší prostor, než má k dispozici.

Výpočty rozměrů a okrajů prvků, rozmísťování a zarovnávání prvků na stránce patří k nejnáročnějším oblastem celé technologie CSS. V dalších částech knihy je jim věnováno několik kapitol, kde jsou všechny související otázky podrobně vysvětleny.

2.5.2 Rámečky

Pro styl rámečku se v CSS používá celá řada vlastností, definujících jeho parametry (barvu, sílu a typ čáry), dokonce i samostatně pro každou ze čtyř stran. K nim existuje ještě skupina *sdružených vlastností* (zkratek), pomocí nichž je možné definovat současně všechny parametry pro jednu stranu rámečku (`border-left`, `border-top` atd.) nebo naopak jeden parametr pro celý rámeček (`border-style`, `border-color`, `border-width`). Pro nás nyní bude nejužitečnější vlastnost, která shrnuje všechny parametry pro celý rámeček.

Touto vlastností je `border` a jejími hodnotami je trojice hodnot (oddělených mezerami), udávajících sílu, styl a barvu celého rámečku. **Síla** určuje tloušťku čáry, kterou má být rámeček zobrazen. Na obrazovce počítače je nejobvyklejší používat pro rámečky jednotku `px` — např. `1px`, `3px`. **Styl** definuje druh použitého rámečku. Používají se předdefinovaná klíčová slova pro plnou čáru (`solid`), čárkovanou (`dashed`) a tečkovanou čáru (`dotted`), hodnota `none` (žádný rámeček) a další. **Barva** rámečku se definuje stejně jako barva textu — např. `#FFCC99`, `black` atd.

—> Vlastnost border [4.3.3], Jednotky CSS [3.3]

Jako hodnotu vlastnosti `border` zapisujeme celou trojici (v libovolném pořadí), např.: `border:␣1px␣solid␣black`. Pokud prvek nemá mít rámeček, stačí použít pouze hodnotu `border:none`, ostatní dva údaje jsou zbytečné.

Na ukázkové stránce můžeme definovat rámeček pro záhlaví. V původní ukázce bylo orámováno s použitím vnořených tabulek, které ve výsledku vytvořily zdání tenkého černého rámečku. S pomocí CSS postačí vhodně použít vlastnost `border`:

```
#zahlavi {
      text-align: center;
      border: 1px solid black;
      }
```

Tím jsme určili, že záhlaví bude orámováno jednobodovou plnou čarou černé barvy. Při pohledu na stránku si však můžeme všimnout, že nahoře je mezi rámečkem a firemním logem

příliš prostoru, zatímco dole je rámeček nehezky „nalepený“ na text. V dalším kroku proto upravíme příslušné okraje.

2.5.3 Odstup od rámečku

Skupinou vlastností *padding* se definuje vzdálenost obsahu od rámu. Tyto vlastnosti určují odstup od jednotlivých stran rámečku (`padding-left`, `padding-top` atd.) a sdružená vlastnost `padding` pak umožňuje shrnout všechny tyto hodnoty do jediné definice.

Vlastnost `padding` může mít jednu až čtyři hodnoty, které po řadě udávají **odstup od horního, pravého, spodního a levého rámu** (vždy shora po směru hodinových ručiček), např. `padding: 1em 2pt 0.5em 4pt`. Pokud použijeme méně než čtyři hodnoty, použije se pro chybějící směr hodnota stejná jako pro směr protější. Tři hodnoty udávají po řadě horní, současně levý+pravý a nakonec spodní odstup. Dvě hodnoty vyjadřují současně horní+dolní a levý+pravý odstup, jediná hodnota definuje odstup na všech stranách stejný.

Používají se číselné hodnoty s jednotkou. Často je zde vhodné použít jednotku `em`, která odpovídá velikosti písma použité v prvku. Pokud si uživatel v prohlížeči zvětší či zmenší písmo, přizpůsobí se mu tak proporčně i velikosti okrajů. Výchozí hodnotou je 0 a prohlížeče ve své *výchozí tabulce stylů* `padding` většinou prvkům nijak nedefinují. To znamená, že pokud `padding` neurčíme sami, bude u většiny prvků nulový.

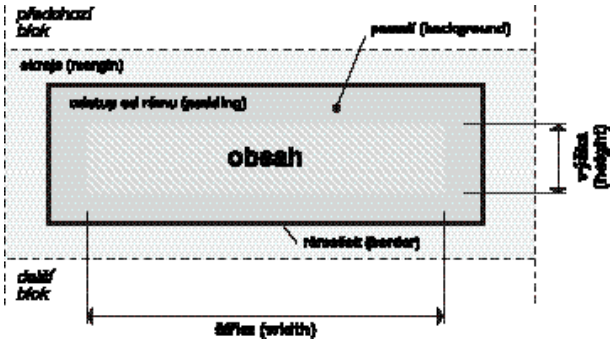
—>Vlastnost padding [4.3.2]

V záhlaví naší stránky můžeme použít odstup např. `0.6em` na všech stranách stejný:

```
#zahlavi { ... padding: 0.6em; }
```

Prostor nad obrázkem se tím sice zvětšil, ale to má jinou příčinu (viz dále). Odstupy od rámečku jsou však v pořádku, odpovídají velikosti písma definované pro záhlaví. Připomeňme, že jelikož jsme pro záhlaví velikost písma nedefinovali, zdědilo hodnotu z nadřazeného prvku — tedy `12pt` definovaných pro `body`. Odstup od rámečku `0.6em` bude odpovídat šesti desetinám této velikosti, tj. `7.2pt`.

Jak je to tedy s tím prázdným prostorem nahoře? Jelikož prvky mají vesměs průhledné pozadí, není vidět, které oblasti bloků toto prázdné místo tvoří. Při ladění stránek bývá užitečné některým prvkům dočasně definovat rámeček či pozadí, aby tyto vztahy byly lépe vidět. Zde si ale ukážeme situaci v záhlaví na schématickém obrázku:



Obr. 12 — Struktura bloků v záhlaví stránky

Je vidět, že velkou část místa zabírá okraj prvku `<h1>`, obsahujícího titulní obrázek. Ve *výchozí tabulce prohlížečů* mají nadpisy často definovaný větší okraj, proto se projeví i zde. Kdybychom tento obrázek neoznačili jako nadpis, okraj by zde nebyl. Bylo to ale potřebné kvůli struktuře stránky — logo firmy zde úspěšně zastupuje název stránky a je tedy umístěno v titulku nejvyšší úrovně. Druhý prvek, `<address>`, zde má okraje nulové, protože je v tabulce prohlížeče nemá nijak definovány. V jiném prohlížeči to však může být odlišné. Pro zachování shodného vzhledu je proto dobré tyto vlastnosti definovat přesně tak, jak požadujeme.

2.5.4 Okraje

Okraje *(margin)* se definují obdobně jako *výplně (padding)* skupinou vlastností (`margin-left`, `margin-top` atd.). *Sdružená vlastnost* `margin` pak popisuje současně všechny čtyři okraje kolem prvku. Také zde jsou hodnotou jeden až čtyři údaje (shodně jako u vlastnosti `padding`), také zde je výchozí hodnotou 0.

—> Vlastnost margin [4.3.1]

Narozdíl od vlastnosti `padding` je ale `margin` definován ve *výchozí tabulce stylů* v prohlížečích celkem hojně. Najdete tu především horní a dolní okraje u odstavců `<p>`, nadpisů `<h1>` až `<h6>` i další definice okrajů u mnoha prvků. Stačí se podívat, jak se v prohlížeči zobrazí libovolná stránka beze stylů. Pouze prvek `div` má obvykle zajištěny nulové okraje (proto se používá jako „neškodný“ pro uzavření úseků stránky). Levé a pravé okraje se vesměs ponechávají nulové, používají se jen pro zleva odsazované prvky, např. seznamy a jejich položky.

A právě okraje u prvku `<h1>` způsobují velkou mezeru mezi rámečkem záhlaví a logem firmy na ukázkové stránce. V tabulce stylů proto nadefinujeme okraje a odstupy pro oba prvky v záhlaví podle svých potřeb. Z dřívějška už víme, že pomocí *kontextových selektorů* se můžeme omezit jen na prvky v záhlaví:

```
#zahlavi h1 { padding: 0; margin: 0 0 3pt 0; }
#zahlavi address { padding: 0; margin: 0; }
```

Pro nadpis `h1` zde definujeme dolní okraj (třetí hodnota `margin`) celkem malý, pro naše potřeby je ale postačující. Za povšimnutí stojí fakt, že spolu sousedí dva prvky (`h1` a `address`), které mají v místě dotyku definován různý okraj — `h1` má spodní okraj `3pt`, `address` má horní okraj nulový. V CSS platí pravidlo, že **okraje *(margin)* sousedících prvků se nesčítají, ale spojí se do okraje jediného o velikosti větší z obou hodnot.** Výsledek by tedy byl stejný, i když by prvek `address` měl horní okraj rovněž `3pt` (anebo méně).

Pozn.: Pomocí vlastnosti `margin` prvku `body` či `html` můžeme definovat okraje stránky. Pokud nám nebudou vyhovovat výchozí okraje v prohlížeči, můžeme je takto nastavit na jinou hodnotu. Např.:

```
body { margin: 1em 2em }
```

2.5.5 Styl pozadí

„Staré“ formátování prostředky HTML umožňovalo nastavit barvu nebo dlaždicově se opakující obrázek na pozadí stránky a v tabulkách. CSS má podstatně bohatší možnosti. Pozadí již mohou mít **všechny prvky,** obrázky lze libovolně zarovnat vodorovně i svisle, opakování může být nastaveno jen v jednom směru, případně vůbec ne atd.

Pozadí můžeme definovat *sdruženou vlastností* `background`. Její hodnotou je barva (stejně jako u vlastnosti `color`) nebo adresa obrázku a jeho parametry, případně obojí. Obě hodnoty současně mají smysl, pokud použijeme částečně průhledný obrázek, nebo jej opakujeme jen v jednom směru. Tam, kde nebude obrázek, bude vidět použitá barva.

—> Vlastnosti pozadí [4.5.2]

Adresa obrázku na pozadí zapisuje pomocí funkce `url()`, jejímž parametrem je relativní či absolutní URL souboru, volitelně uzavřená v uvozovkách (adresy jsou relativní k souboru s tabulkou stylů). Stejně jako v původní ukázce můžeme umístit obrázek `"sroub.jpg"` na pozadí stránky — definujeme pozadí pro prvek `body`:

```
body { ... background: url("img/sroub.jpg") }
```

Pokud neuvedeme žádný parametr, obrázek se bude opakovat jako dlaždice. Nyní je tento obrázek na celém pozadí stránky, a pokud si ji načteme v prohlížeči, uvidíme, že pozadí prosvítá skrze všechny prvky. Ty totiž stále používají výchozí hodnotu pozadí, jíž je `transparent` (průhledné pozadí). Budeme jim muset postupně nastavit nějaké vhodné pozadí, aby byla stránka čitelnější.

Začneme se záhlavím, kterému nastavíme bílé pozadí:

```
#zahlavi { ... background: white }
```

S tím, co už víme, však můžeme také vytvořit záhlaví ve stejné podobě jako na původní stránce. Na pozadí umístíme obrázek šroubu, zarovnaný do pravého horního rohu bez opakování. Změníme tedy poslední definici:

```
#zahlavi { ...
      background: white url("img/sroub.jpg") top right no-repeat;
      }
```

Obsah záhlaví zarovnaný na osu však do obrázku zasahuje. Postačí, když nastavíme odstup od pravého rámečku na velikost odpovídající šířce obrázku (`300px`). Změníme proto definici `padding` pro záhlaví:

```
#zahlavi { ... padding: 0.6em 300px 0.6em 0.6em }
```

Nyní již je záhlaví zformátováno tak, jak má být.

Podobně můžeme upravit i další části stránky. V původní ukázce je stránka tvořena dvěma velkými bloky, záhlavím a obsahem. Záhlaví už na stránce máme vyznačeno, nejlepší nyní bude — podle původní předlohy — označit si na stránce i druhou oblast, kterou si pojmenujeme třeba `"obsah"`:

```
<div id="obsah">
      <h2>Vítejte na našich ...
      ...
              <div id="novinky">
              ...
              </div>
      <h2>Navigace:</h2>
      ...
      </p>
</div>
```

Tuto oblast pro jednoduchost zatím zformátujeme stejně jako záhlaví — černý rámeček, bílé pozadí, odstup od rámečku `0.6em`:

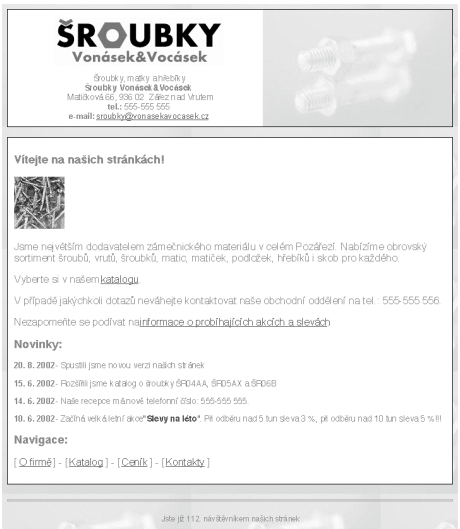
```
#obsah {
      border:1px solid black;
      padding: 0.6em;
      background: white;
      }
```

Obě oblasti (záhlaví a obsah) však spolu nyní přímo sousedí a jejich rámečky se dotýkají. Je to proto, že používáme prvky `div`, které mají nulové okraje. Rozestup mezi nimi vytvoříme tím, že jim nějaký okraj nastavíme. Protože se u sousedících prvků používá větší z obou okrajů, stačí ho definovat jen jednomu z nich, např.:

```
#obsah { ... margin: 1em 0 }
```

Tím jsme nastavili obsahu horní a dolní okraje velké `1em`, mezi záhlavím a obsahem bude mezera odpovídající velikosti písma.

V tuto chvíli již máme hotovou velkou část formátování stránky. Ta nyní v prohlížeči vypadá takto (na Internetu ji najdete jako *Příklad 2d*):



Obr. 13 — Zobrazení stránky z Příkladu 2d v prohlížeči

Stránka již začíná dostávat jistou grafickou formu, s tímto základním formátováním si lze vystačit na mnoha jednodušších stránkách. Nyní půjdeme dál a ukážeme si některé další možnosti formátování, které CSS poskytuje.

2.6 Další formátovací nástroje CSS

2.6.1 Plovoucí prvky

Podobně, jako se v HTML pomocí atributu `align` v prvcích `img` vytvářely plovoucí (obtékané) obrázky, můžeme pomocí CSS nastavit obtékání jakéhokoli prvku. Zajistí to vlastnost `float`, která má hodnotu `left`, `right` či `none`. Pokud definujeme prvku `float:left`, resp. `float:right`, stane se *plovoucím* — přesune se k levému (resp. pravému) okraji svého nadřazeného prvku a všechny další prvky, budou zobrazeny ve zbývajícím prostoru vpravo, resp. vlevo od něj.

Je jasné, že má-li být prvek obtékán dalším obsahem, musí být užší než prostor, který je pro něj určen. Obrázky mají šířku určenu svými rozměry, ostatním obtékaným prvkům musíme šířku definovat vlastností `width` tak, aby se vedle nich ještě další obsah vešel.

—> Plovoucí prvky [3.8.6.5], vlastnost float [4.3.5.5]

Nyní již můžeme definovat obtékání pro obrázek `"vruty.jpg"` v ukázkové stránce. Nejprve si jej pojmenujeme (např. `<img id="vruty" ...>`) a do tabulky stylů přidáme pravidlo:

```
#vruty { float:left }
```

Chceme-li, aby obtékající text začínal dále od obrázku, můžeme mu ještě definovat nějaký pravý okraj:

```
#vruty { float:left; margin: 0 1em 0 0 }
```

2.6.2 Způsob zobrazování prvků

Již jsme zmínili, že některé prvky jsou *bloky (blokové prvky)* a jiné jsou pouze *úseky textu (řádkové prvky).* Je to pouze hrubé rozlišení, CSS rozeznává na sedmnáct typů prvků. Tento typ je popsán vlastností `display` a její hodnotu je možné změnit. Z *řádkových prvků* tak můžeme vytvořit bloky (`display:block`) a naopak z *blokových prvků* zase úseky textu (`display:inline`). Pomocí `display:none` dokonce zcela zamezíme zobrazení prvku; stránka se chová, jako by v ní prvek vůbec neexistoval.

—> Vlastnost display [4.3.5.1]

To může být užitečné pro skrytí prvků, které mají smysl pouze v základním formátování, bez použití stylů. V naší ukázkové stránce tak například můžeme skrýt horizontální linky `<hr>`, protože vodorovné čáry vytváříme pomocí rámečků.

```
hr { display: none }
```

2.6.3 Pozicování prvků

Rozmísťování prvků na stránce (pozicování) je velmi komplexní a poměrně komplikovanou částí CSS. Pracuje zde v součinnosti mnoho parametrů prvků — jejich rozměry i způsob jejich definování, hodnoty vlastností `margin`, `padding` i `border`, vlastnosti `position`, `display`, `float`, `clear` a mnoho dalších. Pro různé kombinace hodnot těchto vlastností existují různá pravidla pro zobrazování prvku. Ve hře je i použitý *typ dokumentu,* chyby v prohlížečích, nejednotnost v implementacích CSS atd.

Stejného či podobného rozvržení prvků na stránce lze často dosáhnout několika různými způsoby. Pro smysluplné používání pozicování prvků na stránce je nezbytné nastudovat i další kapitoly této knihy, které se této problematice podrobněji věnují. Zde si alespoň představíme některé možnosti, jak dosáhnout kýženého vzhledu naší ukázkové stránky.

Nejčastěji se pro rozmístění prvků na stránce používají **plovoucí prvky, absolutní pozicování** či kombinace obou metod.

—> Plovoucí prvky [3.8.6.5], absolutní pozicování prvků [3.8.6.6]

V prvním případě můžeme některý úsek stránky pomocí vlastnosti `float` odsunout ke straně a další obsah bude tento úsek *obtékat.* Často se používá hned několika plovoucích prvků vedle sebe, které tak vytvoří *sloupce.* Nevýhodou této metody je, že vyžaduje určité pořadí úseků v dokumentu: prvky, které mají být plovoucí, musí být v kódu uvedeny dříve. Na naší stránce bychom mohli např. vytvořit plovoucí úsek z hlavního textu stránky (plovoucí vlevo), vedle něj by byly umístěny „novinky“. Navigace pak již plovoucí nebude a bude zobrazena pod nimi. Všem oblastem pak ještě vhodně doplníme formátování (pozadí, rámečky atd.). Takto upravená stránka, její kód a doplněná tabulka stylů se nachází na Internetu jako *Příklad 2e,* v prohlížeči bude zobrazena takto:

Obr. 14 — Zobrazení stránky z příkladu 2e v prohlížeči

S využitím **absolutního pozicování** už lze zformátovat stránku prakticky libovolně. Tento formátovací model dokáže určit každému prvku přesnou pozici na stránce. Pomocí definice `position:absolute` se prvek vyjme ze stránky (jeho okolí zůstane zformátováno, jako by tam tento prvek nikdy nebyl) a umístí se na souřadnice dané vlastnostmi `top`, `bottom` a `left`, `right`. Absolutně pozicovaná stránka může vypadat např. takto:

Obr. 15 — Zobrazení stránky s absolutním pozicováním

Oblast s navigací byla úmyslně ponechána nezformátovaná — posledním příspěvkem v této kapitole budou právě styly odkazů, které na stránce ještě upravíme.

2.6.4 Styly odkazů

Odkazům můžeme definovat styl stejně jako jiným prvkům. K tomu máme navíc v CSS možnost definovat styl i *pro různé stavy odkazů.* Odkaz, který již byl navštíven, tak může mít podobu odlišnou od odkazu dosud nenavštíveného. Jinak může vypadat odkaz, na který uživatel ukáže myší, jinak odkaz označený klávesnicí. Pro tyto stavy existují speciální selektory, které můžeme použít v tabulce stylů:

- `a` — styl všech odkazů `<a>`
- `a:link` — styl nenavštívených odkazů
- `a:visited` — styl navštívených odkazů
- `a:hover` — styl odkazu, na který se právě ukazuje myší
- `a:focus` — styl označeného odkazu
- `a:action` — styl aktivního odkazu (uživatel jej aktivoval)

—> Selektory pro odkazy [3.6.5.1]

Do tabulky stylů naší stránky přidáme např.:

```
a { color: #CC0000; text-decoration: underline }
a:hover { color: white; background: #CC0000; text-decoration:none }
```

Tato pravidla zajistí, že všechny odkazy (navštívené i nenavštívené) budou zobrazeny tmavě červeně a budou podtrženy. Pokud na ně uživatel ukáže, změní se barva na bílou s tmavočerveným pozadím a podtržení se zruší.

Na stránce také využijeme zmíněnou změnu formátování prvků pomocí vlastnosti `display`. Zobrazíme odkazy v „navigaci“ v podobě menu, obdobně jako v původní ukázce. Tam byly k vytvoření menu použity obrázky ve spojení s událostmi a funkcemi Javascriptu. Nyní si vystačíme pouze s možnostmi CSS.

Do tabulky stylů přidáme pravidlo:

```
#navigace a { display: block }
```

Tím vytvoříme z odkazů blokové prvky, které se stejně jako odstavce zobrazují na samostatných řádcích:

Obr. 16 — Zobrazení blokových odkazů

Závorky a oddělovače kolem odkazů v této podobě stránek potřebovat nebudeme, označíme si je proto v dokumentu, abychom je mohli skrýt. Jelikož jsou to *úseky textu,* použijeme pro ně „neškodné“ textové značky `<span>`. Těm přiřadíme stejnou třídu, např. `"schovat"`:

```
<span class="schovat">[ </span>
<a href="onas.html">O firmě</a>
<span class="schovat">] - </span>
...
```

a do tabulky stylů přidáme

```
.schovat { display: none }
```

Tím jsme v navigaci skryli vše kromě odkazů `<a>`. Jim nadefinujeme vhodný styl. Můžeme použít vše, co už známe z formátování textu i bloků. Odkazům předepíšeme velikost písma, tučnost, barvu, zrušíme případné podtržení, určíme zarovnání, okraje, odstup od rámečku a barvu pozadí:

```
#navigace a {
      display:block;
      font-size: 15pt;
      font-weight: bold;
      color: #CC0000;
      text-decoration: none;
      text-align:left;
      margin: 0 0 0.5em 0;
      padding: 0.3em 0.5em;
      background: white;
      }
```

V odkazech, na které uživatel ukáže, pak definujeme negativní barevnost:

```
#navigace a:hover { color:white; background:#CC0000 }
```

Tím jsme vytvořili jednoduché dynamické *menu.* Můžeme ještě doplnit obrázek na pozadí, abychom se přiblížili původnímu vzhledu. Obrázek umístíme doleva (výškově na střed) a přizpůsobíme mu odsazení textu odkazu:

```
#navigace a {
      ...
      padding: 0.3em 0.5em 0.3em 40px;
      background: white url("img/sr_out.gif") center left no-repeat;
      }
#navigace a:hover {
      ...
      background: #CC0000 url("img/sr_over.gif") center left no-repeat;
      }
```

Výsledná stránka již prakticky odpovídá původní podobě. Její kód a tabulku stylů najdete na Internetu jako *Příklad 2f:*

Obr. 17 — Výsledné formátování ukázkové stránky

Obsah dokumentu je stále strukturovaný a zobrazí se přehledně i bez použití stylů **v každém prohlížeči.** Samotný dokument má 2,5 kB, tabulka stylů necelé 2 kB. To je dohromady stále méně, než měla původní stránka, navíc jsme se obešli bez použití skriptů. Obsah a popis vzhledu jsou umístěny ve dvou samostatných souborech, takže je usnadněna správa stránek a navíc se nezměněné soubory mohou načítat z *cache,* což sníží přenos dat ze serveru.

Ve stránce zatím nebyly ošetřeny chyby některých prohlížečů. V nich se **s použitím stylů** stránka nezobrazí zcela korektně (např. v MSIE 5.5 a starších). I těmto chybám se však dá zamezit (některým vhodným postupům se budeme věnovat v poslední kapitole). Stránku, v níž již byly chyby prohlížečů ošetřeny, najdete na Internetu jako *Příklad 2g*.

V další kapitole začneme s podrobným popisem všech doposud uvedených funkcí a vlastností CSS, včetně mnoha dalších, které zatím zmíněny nebyly.

3 Detailní pohled pod kapotu CSS

3.1 Definice a použití CSS

Vývoj CSS spravuje a specifikace jednotlivých verzí publikuje organizace World Wide Web Consortium (W3C). Na jejím serveru je k nalezení mnoho informací s touto technologií souvisejících.

—> Odkazy na důležité dokumenty [Reference]

W3C definuje CSS jako **jazyk tabulek stylů,** který umožňuje autorům připojit popis stylu (písmo, uspořádání, barvy, zvukové smyčky atd.) do strukturovaných dokumentů (stránky HTML, aplikace XML). Oddělením stylu dokumentu od jeho obsahu jazyk CSS usnadňuje tvorbu stránek i správu celých webových serverů.

Aktuální verzí je **CSS 2 (Cascading Style Sheets, level 2),** vycházející z předchozí verze CSS 1, kterou rozšiřuje, doplňuje a je s ní také kompatibilní (všechny styly definované podle CSS 1 jsou platné i pro CSS 2). Rozdílná je pouze jiná priorita tabulek v obou verzích [3.7.3]. V době přípravy této knihy je před dokončením i verze CSS 3, která kaskádové styly modularizuje a významně rozšiřuje jejich možnosti.

Na základě praktických zkušeností byla publikována také předběžná specifikace **CSS 2.1,** která zohledňuje reálnou podporu kaskádových stylů v prohlížečích a změny připravované v CSS 3. Některé z původně zamýšlených funkcí byly odstraněny (zatím nejsou nikde podporovány a v CSS 3 budou řešeny jinak), u některých vlastností bylo přizpůsobeno jejich chování realitě a byly také opraveny některé drobné chyby ze specifikace CSS 2.

V této knize se věnujeme použití kaskádových stylů podle (praxi nejvíce odpovídající) **specifikace CSS 2.1.** Předpokládáme použití CSS v dokumentech HTML, XHTML a XML. Podrobnosti o používání stylů v jednotlivých verzích těchto jazyků naleznete v jejich specifikacích [Reference], způsob užití v jiných typech dokumentů je třeba konzultovat s dokumentací příslušného jazyka.

3.1.1 Použití CSS v dokumentech HTML

Tabulky kaskádových stylů mohou být součástí dokumentů HTML, nebo být umístěny v samostatném souboru. Ten se k dokumentu připojí během jeho načítání. Možné způsoby použití jsou tři:

- **přímá definice stylu** jednoho prvku pomocí atributu `style="..."`
- **vložení tabulky stylů** do stránky pomocí HTML značky `<style>`
- **načtení externí tabulky stylů** pomocí HTML značky `<link>`

3.1.1.1 Přímá definice stylu

Pomocí atributu `style` můžeme definovat styl konkrétnímu prvku zapsáním definice přímo do jeho značky. Prvky, k nimž lze připojit atribut `style`, jsou definovány v DTD příslušné danému typu dokumentu (v HTML 4.01 např. tento atribut nelze použít ve značkách `BASE`, `BASEFONT`, `HEAD`, `HTML`, `META`, `PARAM`, `SCRIPT`, `STYLE` a `TITLE`). Definice se používá ve tvaru:

```
<značka ... style="seznam_definic" ...>
```

např.:

```
<a href="stranka.html" style="color:green;font-size:smaller">
```

Takto definovaný **přímý styl** platí jen pro daný prvek, hodnoty se ale mohou dále dědit.

3.1.1.2 Vložení tabulky stylů

Pro zápis tabulky stylů se používá značka `<style>`. Je párová a umísťuje se do sekce `<head>` dokumentu (tj. mezi značky `<head>` a `</head>`). Jejím obsahem je seznam pravidel, neboli **tabulka stylů.** Tu je většinou dobré uzavřít ještě do komentářů `<!-- -->`, aby popis stylu ignorovaly prohlížeče, které CSS neznají (styly jsou podporovány až od HTML 3.2).

Do značky `<style>` je třeba uvést i příslušný *MIME-typ* `"text/css"`. Tabulky stylů se tedy do stránek vkládají ve tvaru:

```
<head>
...
<style type="text/css">
<!--
   tabulka stylů
-->
</style>
...
</head>
```

3.1.1.3 Načtení externí tabulky stylů

V HTML slouží k připojování jiných dokumentů značka `<link>`. Prostřednictvím jejích atributů určujeme, o jaký dokument se jedná a jak se s ním má nakládat. Pro připojení souborů s tabulkami stylů je třeba použít minimálně atributy `rel="stylesheet"`, `type="text/css"` definující *MIME-typ* dokumentu a `href="URL"` udávající adresu souboru s tabulkou stylů. Značka `<link>` musí být rovněž umístěna v sekci `<head>`. Syntaxe je tedy:

```
<link rel="stylesheet" type="text/css" href="URL">
```

např.

```
<link rel="stylesheet" type="text/css" href="mujstyl.css">
```

Pozn.: Externí soubor lze načíst i pomocí příkazu jazyka CSS `@import` [3.7.3.3]. Pokud jej použijeme jako jediný příkaz ve značce `<style>`, zajistíme načtení CSS do stránky obdobně jako s pomocí značky `<link>`, např.:

```
<style type="text/css">
<!--
   @import url('mujstyl.css');
-->
</style>
```

V externích souborech jsou uloženy tabulky stylů ve stejné podobě, jako se zapisují do značky `<style>`; syntaxe tabulek je popsána dále.

—> Tipy pro připojování stylů [5.1.4], Alternativní styly [5.1.5]

3.1.2 Použití CSS v dokumentech XHTML

V XHTML se CSS používá obdobně jako v HTML, jsou zde pouze menší rozdíly související s odlišnostmi obou jazyků.

V souladu s trendem úplného oddělení obsahu od formy již nové specifikace XHTML počítají pouze s externími styly. Ve standardu XHTML 1.1 už nenajdete *přímé styly* (atribut `style` byl definitivně odstraněn) a navíc se doporučuje používat přednostně externí tabulky stylů (což ostatně platí i pro skripty a další data mimo rámec „čistého“ XHTML).

Při použití značky `<link>` je třeba pamatovat na povinné uzavírání i nepárových značek, proto načtení externího CSS musí být např. ve tvaru:

```
<link rel="stylesheet" type="text/css" href="URL" />
```

Ve značce `<style>` je pak v XHTML nutné deklarovat tabulku stylů jako CDATA, kvůli zpětné kompatibilitě se tato deklarace navíc často uzavírá do komentářů CSS — značka se tedy používá většinou ve tvaru:

```
<head>
...
<style type="text/css">
/* <![CDATA[ */
   tabulka stylů
/* ]]> */
</style>
...
</head>
```

```
<?xml-stylesheet type="text/css" href="URL"?>
```

3.1.3 Použití CSS v dokumentech XML

V jazyce XML se externí soubory s CSS načítají do dokumentu pomocí deklarace:

```
<?xml-stylesheet type="text/css" href="URL"?>
```

Dokumenty XML závisejí na tabulkách stylů daleko více než dokumenty HTML a XML je s CSS také hlouběji svázáno. Ale CSS není jedinou možností formátování dokumentů XML. Mohutnějším nástrojem s většími možnostmi je jazyk XSL, jeho popis je však mimo rámec této knihy. Podrobnosti naleznete ve specifikacích XML a XSL [Reference].

Pozn.: Protože podmnožinou XML je i XHTML, můžeme v něm používat i konstrukce jazyka XML — např. výše uvedené načtení stylů je správné i v dokumentech XHTML.

3.1.4 Podpora CSS v prohlížečích

Protože se implementace kaskádových stylů v prohlížečích výrazně liší, u popisu vlastností CSS najdete i tabulky kompatibility v nejčastěji používaných prohlížečích. Můžete ihned zjistit, jak jsou popsané funkce CSS podporovány, zda je možné je v praxi využívat či s jakými problémy se při jejich používání můžete setkat.

3.2 Syntaxe CSS

Jak plyne z předchozí kapitoly, definice kaskádových stylů se používají ve dvojí podobě. Buďto zapíšeme styl přímo do značky konkrétního prvku (tzv. **přímé styly**), nebo jej definujeme v **tabulkách stylů** *(stylesheets)* — ty jsou potom platné pro celý dokument.

Přímé styly nejsou tak úplně „čistou“ formou CSS. Do jazyka HTML se spíše vloudily v průběhu času a v nových verzích jsou z něj zase odstraňovány. Jejich funkčnost (a potažmo i syntaxe) je omezená a měly by se používat jen pro „doladění“ podoby dokumentů ve starších a *přechodových* verzích HTML. CSS preferuje *tabulky stylů,* o nichž především pojednává a které by měly být používány přednostně. Dokumenty vyhovující nejnovějším standardům pak už mohou používat pouze tyto tabulky, navíc raději načítané z externích souborů na úkor jejich zápisu přímo do stránek.

3.2.1 Symbolický popis syntaxe

3.2.1.1 Speciální znaky v CSS

Při definování kaskádových stylů se používají znaky podle normy ISO 10646. V našem prostředí, kde je situace komplikována množstvím kódování a znakových sad, je lepší se „neobvyklým“ znakům raději vyhnout a používat pouze sadu ASCII (písmena latinské abecedy, číslice a běžné interpunkční značky). Některé znaky mají v CSS zvláštní význam, níže je uveden popis těch nejdůležitějších.

Uvozovky. Pro uvození textových řetězců se používají pouze znaky `"` a `'` (dvojité/jednoduché uvozovky, ASCII znaky 34 a 39). Pozor na záměnu s typografickými uvozovkami či apostrofy, které zde nejsou povoleny.

Mezera. Není-li uvedeno jinak, na místě každé mezery může být libovolný *prázdný prostor (whitespace),* tedy jakákoli posloupnost znaků *mezera* (ASCII znak 32), CR (13), LF (10), FF (12) či tabelátor (9). Pozor, jiné znaky, např. em/en-mezera, nedělitelná mezera atd., se za prázdný prostor nepovažují, ačkoli nejsou většinou v kódu vidět.

Spojovník. Znak „`-`“ (ASCII znak 45), nezaměňovat za delší typografické pomlčky, které zde nejsou akceptovány. Tento znak se používá jako spojovník v názvech vlastností CSS i jako znaménko *mínus* v číslech (`margin-top`, `-3`).

Tečka. V číslech se používá jako oddělovač desetinné části zásadně znak „`.`“ a u nás obvyklý zápis s desetinnou *čárkou* je chybný.

Čárka a středník. Používají se jako oddělovače seznamů, před i za nimi může být libovolný *prázdný prostor.*

3.2.1.2 Symboly v zápisu syntaxe

V popisech syntaxe a v popisech vlastností CSS budou pro přesný popis povolených znaků a hodnot použity také symbolické zápisy. Některé znaky (|, [,], {, }, ?, *, +, - mezera) mají zvláštní význam, pokud se má tento znak vyskytovat v textu, bude uveden v uvozovkách (např. `']'`).

`[ … ]`	hranaté závorky uzavírají položky pro následující varianty
`[X]*`	výraz X se může vyskytovat v libovolném počtu (0 až n-krát)
`[X]+`	výraz X se může vyskytovat v libovolném počtu, ale aspoň jednou (1 až n-krát)
`[X]?`	výraz X se může i nemusí vyskytovat (0 nebo 1-krát)
`[X]{m,n}`	výraz X se opakuje nejméně m-krát a nejvýše n-krát
`[X|Y|Z]`	seznam alternativ, vyskytuje se *právě jedna* z uvedených položek
`[X||Y||Z]`	seznam alternativ, vyskytuje se *jedna či více* z uvedených položek v libovolném pořadí
mezera	všude, kde je uvedena mezera, může být libovolný *prázdný prostor* (viz výše).

3.2.2 Popis syntaxe CSS

3.2.2.1 Definice CSS

V jazyce CSS **definicemi stylů** přiřazujeme **hodnoty** jednotlivým **vlastnostem.** Tyto definice jsou vždy ve tvaru `vlastnost:␣hodnota` (mezera je nepovinná) a vyjadřují *přiřazení* příslušné hodnoty této vlastnosti. Definice se mohou spojovat do **seznamu definic,** v němž se jednotlivé definice oddělují znakem „`;`“ (středník).

```
vlastnost: hodnota [;]*
```

Dvojtečka musí následovat přímo za názvem vlastnosti, ostatní *mezery* jsou libovolné. Např.:

```
margin-top:5pt
margin-top:␣␣␣␣␣␣␣5pt;
margin-top: 5pt ;
```

3.2.2.2 Sdružené vlastnosti

Některé vlastnosti CSS jsou deklarovány jako **vlastnosti sdružené** (zkratky). Jejich hodnotou není přímo žádný parametr prvku, umožňují pouze v jediné definici nastavit více dalších vlastností současně. Hodnotou *sdružené vlastnosti* je seznam dílčích hodnot (oddělený *mezerami*). V jejím popisu je vždy uvedeno, jaké vlastnosti sdružuje a jakým způsobem se jim dílčí hodnoty přiřazují. Např. skupinu definic:

```
font-variant:normal; font-style:italic; font-weight:bold;
font-size:12pt; line-height:120%; font-family:serif;
```

lze nahradit definicí sdružené vlastnosti `font`:

```
font: normal italic bold 12pt/120% serif;
```

Pokud je možné nějakou z dílčích hodnot ve sdružené vlastnosti vynechat a skutečně ji vynecháme, odpovídající vlastnost nebude definována. Např. ve zmíněné vlastnosti `font` jsou povinné pouze dílčí hodnoty pro `font-size` a `font-family`, ostatní zde být nemusí. Zapíšeme-li pouze:

```
font: 12pt serif;
```

ostatní dílčí vlastnosti, které vlastnost `font` sdružuje, nebudou mít definovánu hodnotu (jejich hodnota se *zdědí,* nebo se použije *výchozí* hodnota — viz dále).

3.2.2.3 Přímé styly

Přímé styly, které definují styl pouze pro jeden prvek, jsou tvořeny **jen seznamem definic.** Zapisují se tedy (do atributu `style` příslušné značky) ve tvaru:

```
style = " vlastnost: hodnota [; vlastnost: hodnota ]* "
```

např.:

```
<a href="a.htm" style="color:red">...
<a href="b.htm" style="color: blue; text-decoration: none">...
```

3.2.2.4 Tabulky stylů

Tabulky stylů definují styl pro celý dokument a tvoří je seznam **pravidel** a **@pravidel,** případně doplněných **komentářů.** Protože definice zde již nejsou přiřazeny konkrétnímu prvku stránky, musí jim v tabulkách stylů předcházet ještě určení, na které prvky se tyto definice mají vztahovat. To se popisuje tzv. **selektory** [3.6].

3.2.2.5 Pravidla

Dvojice selektor + seznam definic se nazývá **pravidlo** a zapisuje se ve tvaru

```
selektor { seznam_definic }
```

neboli

```
selektor { vlastnost: hodnota [; vlastnost: hodnota ]* }
```

V následující ukázce je na každém řádku jedno pravidlo CSS. Výrazy před složenými závorkami jsou příklady *selektorů* (všechny budou popsány dále), uvnitř složených závorek jsou *seznamy definic* jejich stylů. Za poslední definicí středník být může i nemusí.

```
body { background:white; color:black; }
p.nadpis { margin: 1em; padding: 1ex 20px 1em 40px }
#zahlavi > h1 { z-index:2; }
a.externi:focus:hover { color:silver; border:1px solid silver }
ul+ul:before { content:"> " }
```

Jádro každé tabulky stylů je potom tvořeno **seznamem pravidel** (výše uvedené příklady tvoří takový seznam). Kvůli přehlednosti je obvyklé psát každé pravidlo na samostatný řádek, ale není to nezbytné — všude, kde je v popisu syntaxe uvedena *mezera,* může být libovolný *prázdný prostor* (*whitespace;* viz [Legenda]), ale také tam nemusí být nic. Nějaký *prázdný prostor* musí být pouze tam, kde mezi výrazy už není jiný oddělovač (závorka, čárka) a nedalo by se tak poznat, kde jeden výraz končí a kde začíná druhý.

3.2.2.6 @-pravidla

Tabulky stylů mohou kromě *pravidel* obsahovat i tzv. **@-pravidla.** Ta používají speciální příkazy (uvozené znakem `@`), které rozšiřují CSS o další funkce a jsou popsány dále. Tato @-pravidla jsou tvořena **@-příkazem** (např. `@import`, `@media` atd.) za nímž následuje jeho *blok* (podle typu příkazu), např.:

```
@import url('mujstyl.css');
@media print {...}
```

atd.

Pozn.: Pravidla **@import** [3.7.3.3] jsou výjimečná tím, že jim nesmí předcházet žádná *pravidla*, ani nesmí být umístěna uvnitř jiného *bloku* (např. uvnitř bloku `@media`) — jinak je klient povinen je ignorovat. Pro běžné použití si postačí pamatovat, že je nutné uvádět pravidla *@import* vždy jako první, hned na začátku tabulky.

3.2.2.7 Komentáře

Prakticky kdekoli v tabulce stylů se může vyskytovat **komentář.** Uvozuje se znaky `/*` (začátek komentáře) a `*/` (konec komentáře) — vše, co je mezi nimi, se považuje za komentář a je při načítání tabulky ignorováno. Komentáře ale mohou být pouze v *mezerách* tabulek stylů, nelze je vsouvat doprostřed výrazů a slov; následující komentáře jsou tedy korektní:

```
/* moje styly */
body { font-size:12pt; /* velikost písma */ }
h1, h2 /* titulky */ { font-size: /* chci malé titulky */ 12pt; }
```

zatímco tyto jsou umístěny **chybně:**

```
body { font-/*písmo*/size: 12pt; }
h1, h2 { font-size: 12/* chci malé titulky */pt }
```

Komentáře HTML `<!--` … `-->` se v CSS mohou také objevit, ale nijak syntaxi neovlivňují a jejich obsah *není* ignorován. Jsou zde povoleny především proto, aby se pomocí nich mohly skrýt tabulky stylů ve stránkách před „starými“ prohlížeči.

Příklad:

Ukázka tabulky stylů:

```
<!--
/* nejprve načteme externí tabulku */
@import url("spolecnestyly.css");

/* styly pro značky */
body {
   margin:2em;
   font-size:large;
   }
p { margin:1em }

/* styly pro třídy */
.nadpis { color:blue } /* proč nemít modré titulky */

@media print {       /* styl pro tisk */
   p { margin:1cm; }
/* h1 { font-size:200% } nepoužito */
   }
-->
```

Pozn.: Ti, kdo potřebují exaktní syntaxi jazyka CSS nebo chtějí tabulky stylů zpracovávat svými algoritmy, najdou *tokenizaci,* symbolický popis syntaxe i podrobnou *gramatiku* jazyka ve specifikaci příslušné verze jazyka CSS.

3.3 Hodnoty a jednotky

V CSS se používají hodnoty několika typů a společně s nimi i celá řada jednotek. Každá vlastnost má dán seznam povolených hodnot, všechny ostatní musí prohlížeč ignorovat. Typy hodnot jsou zde pojmenovány symbolickým jménem (např. `<číslo>`)**.** Na něj se budeme v definicích vlastností odkazovat.

3.3.1 Klíčová slova

Některé hodnoty mohou být zadány pomocí **klíčových slov.** Ta obvykle **zastupují jinou hodnotu,** na niž se dané klíčové slovo nakonec převede. Mohou představovat „přezdívku“ konkrétní hodnoty pro zjednodušení složitého zápisu (např. barva `yellow` představuje hodnotu `rgb(100%,100%,0)`), nebo mohou být pouze symbolickým vyjádřením hodnoty závislé na použitém zařízení (např. velikost písma `medium` reprezentuje střední velikost písma, která bude různá v každém prohlížeči).

Klíčová slova se zapisují bez uvozovek (s uvozovkami se jedná o `<řetězec>`). Např.: `color:yellow` či `display:none` (zápisy `color:"yellow"` a `display:'none'` jsou chybné).

3.3.2 <číslo>, <celé-číslo>

V CSS se používají dva typy čísel: celá a reálná. **Celá čísla** (`<celé-číslo>`) se zapisují pouze pomocí číslic 0–9, před nimiž může být případně jedno znaménko „`+`“ či „`-`“ (bez mezery). **Reálná čísla** (`<číslo>`) jsou buďto *celá čísla,* nebo čísla desetinná. Desetinná část se odděluje znakem „`.`“ (tečka), nikoli čárkou jako je zvykem u nás. Pokud je celá část desetinného čísla rovna nule, není třeba nulu psát a lze začít přímo tečkou (např. `0.9` i `.9`). Vlastnosti mohou dále omezovat rozsah svých povolených hodnot (nenulová čísla, kladná čísla atd.).

Příklad:

`<celé-číslo>`: `0`, `1`, `+10`, `-156`
`<číslo>`: `0`, `1`, `+10`, `-156`, `3.14159`, `-2.1`, `.3`

3.3.3 <velikost>

Pomocí **velikostí** se definují rozměry a vzdálenosti. Zapisují se ve tvaru `<číslo>jednotka` (bez mezery). Pokud je číslo 0, jednotka se obvykle vynechává (nula je nula s jakoukoli jednotkou). Některé vlastnosti povolují i záporné hodnoty, ale ty nemusí být všude podporovány — v tom případě se použije nejbližší vhodná hodnota. Velikosti používají jednotky *relativní* (ty závisejí na jiných hodnotách) a *absolutní.*

3.3.3.1 Relativní jednotky

em — velikost `1em` odpovídá *vypočítané hodnotě* [3.7.1.3] vlastnosti `font-size` pro daný prvek. Jinými slovy je to **velikost písma právě používaného v prvku.** Pouze při použití ve vlastnosti `font-size` samotné značí `em` velikost písma *rodičovského prvku* [3.5].

ex — velikost `1ex` odpovídá *x-výšce* použitého písma. Správně to má být výška malého písmene „x“ a měla by se tedy lišit podle zvoleného typu písma. Většina prohlížečů však často interpretuje `1ex` zjednodušeně jako polovinu `em` — jakkoli to není přesná a korektní interpretace, je to většinou postačující. Je ale nutné počítat s tím, že novější prohlížeče budou tuto jednotku interpretovat korektněji a neměli bychom se nikdy spoléhat na to, že `ex` je `0.5em`.

px — velikost `1px` většina autorů chápe pouze jako 1 bod obrazovky. Na obrazovkách počítače to sice obvykle platí, ale pro jiná zařízení by taková definice nebyla použitelná. Jednotka `px` (pixel) je *relativní* a závisí na rozlišení a typu výstupního zařízení — `1px` bude jiný na obrazovce počítače, jiný na obrazovce televizoru, jiný na tiskárně nebo při zobrazení velkoplošným projektorem. Pokud se cílové rozlišení výrazně liší od typického rozlišení obrazovky počítače, měl by mu klient velikost `1px` přizpůsobit. Nebudeme proto pixelu říkat „bod obrazovky“, ale přesněji „obrazový bod“, protože přesně to pixel je — jeden bod obrazu.

CSS doporučuje, aby klienty vycházely z velikosti 1 bodu obrazovky při rozlišení 96 dpi, pozorovaného ze vzdálenosti natažené paže. Nedefinuje se zde tedy žádný absolutní rozměr, ale pouze *vjem,* který by měli výrobci prohlížečů uživateli poskytnout — jeden `px` má být bod, který je pro typického uživatele stále *snadno rozlišitelný,* přitom ale ještě není *příliš velký.* Pro obrazovku počítače vychází velikost `1px` např. 0,27 mm (1/96 palce), ale pro tisk na papír, který se většinou čte z menší vzdálenosti než délka natažené paže, může `1px` odpovídat menší velikost — např. 0,2 mm. Při zobrazení na televizní obrazovce, která se typicky sleduje z větší vzdálenosti, zase musí být `1px` úměrně větší — např. pro vzdálenost 3,5 metru to dělá 1,3 mm. Na plátně (obraz z velkoplošného projektoru) pozorovaném ze vzdálenosti 10 m by už měl být `1px` velký přibližně 4 mm.

3.3.3.2 Absolutní jednotky

Absolutní jednotky je možné použít tam, kde známe fyzické vlastnosti výstupního média, např. velikost papíru v tiskárně. Jinak je vhodnější používat jednotky relativní. V CSS se používají tyto absolutní jednotky:

mm — milimetr
cm — centimetr
in — palec (angl. *inch*); 25,4 mm
pt — typografický bod (angl. *point*); 1/72 palce (`72pt` = `1in`, `1pt` = `0,3528mm`)
pc — typografická jednotka *pica,* `1pc` = `12pt` (`1in` = `6pc` = `72pt`).

Příklad použití velikostí:

`<velikost>`: `1px`, `3.5cm`, `.738in`, `-2pt`, `5pc`, `0mm` i `0`

3.3.4 <procenta>

Procentní hodnoty se zapisují ve tvaru `<číslo>%` (bez mezery). Procenta se vždy musí vztahovat k jiné hodnotě. U každé vlastnosti, která použití procentních hodnot dovoluje, je proto vždy výslovně uvedeno, k čemu se procenta vztahují.

Příklad:

`<procenta>`: `100%`, `33.33%`, `-10%`, `.5%`

3.3.5 <uri>

Pomocí URI (*Uniform Resource Identifier*) se popisuje **platná adresa** nějakého zdroje na webu. V CSS se zapisuje pomocí funkce `url()`, jejímž parametrem je webová adresa, volitelně uzavřená v jednoduchých či dvojitých uvozovkách — např. `url(adresa)`, `url('adresa')` i `url("adresa")`. Při zápisu adresy je třeba dodržovat normy pro tyto internetové odkazy [Reference]. Především musíme používat název protokolu na začátku absolutních adres (např. `http://`) a převádět *nepovolené znaky* na příslušné *escape-sekvence* v UTF-8 (mezera je `%20`, pravá závorka `%29` atd.)

Adresy se používají *relativní* i *absolutní*. Relativní adresy se vztahují k umístění souboru s CSS, v němž jsme tuto hodnotu použili. Např. pokud v souboru `http://www.nejakyserver.cz/styly/styl.css` použijeme relativní URI `url("obr.gif")`, odkazujeme tím na soubor `http://www.nejakyserver.cz/styly/obr.gif`. Obrázek stejného jména z nadřazeného adresáře můžeme relativně zapsat jako `url("../obr.gif")`. Absolutní URI je nutné zapisovat se všemi náležitostmi (viz výše) — např. `url("www.neco.cz/muj obr.gif")` prohlížeč obvykle nenajde, správný zápis je `url("http://www.neco.cz/muj%20obr.gif")`.

3.3.6 <barva>

Barvy použité v CSS se zapisují pomocí *klíčových slov* nebo pomocí *číselného zadání* barvy z barevného prostoru *sRGB* [Reference]. Barvy jsou přizpůsobeny zobrazovacím možnostem koncového zařízení (gamut, gama korekce).

Číselně můžeme určit barvu definováním jednotlivých složek červeného (R), zeleného (G) a modrého (B) světla. Barevné složky RGB můžeme v CSS zadávat hexadecimálně, dekadicky i procenty. Používá se rozsah celočíselných hodnot 0–255 (00–FF hexadecimálně), procentní zadání 0–100% se na tyto hodnoty převádí (100% = 255).

Hexadecimální hodnoty se zapisují ve tvaru `#RRGGBB`, kde `RR`, `GG` a `BB` jsou po řadě hodnoty červené, zelené, resp. modré složky barvy. Např.: `#B1F836` (R=`B1`, G=`F8`, B=`36`). Pro barvy, jejichž složky jsou zapsány vždy dvěma stejnými číslicemi (např. `#FF6633`) můžeme použít i zkrácený zápis `#RGB` (např. `#F63`), oba zápisy jsou přitom rovnocenné (např.: `#B58`=`#BB5588`).

Dekadické a procentní hodnoty se zapisují pomocí funkce `rgb()`, která má tři parametry oddělené čárkou, udávající po řadě složky R, G a B. Všechny tři hodnoty musí být shodně dekadické nebo shodně procentní — např. `rgb(127,0,255)`, `rgb(50%,0,100%)`.

Jako **klíčová slova** se používají barvy definované v HTML 4.0 (v závorce je uvedena hodnota barvy, kterou klíčové slovo zastupuje):

```
black (#000000), silver (#C0C0C0), gray (#808080), white (#FFFFFF),
maroon (#800000), red (#FF0000), purple (#800080), fuchsia (#FF00FF),
green (#008000), lime (#00FF00), olive (#808000), yellow (#FFFF00),
navy (#000080), blue (#0000FF), teal (#008080), aqua (#00FFFF).
```

CSS2 pak navíc přidává další klíčová slova, která se odkazují na **systémové barvy** podle grafických prvků prostředí (GUI), v němž uživatel pracuje: `ActiveBorder`, `ActiveCaption`, `AppWorkspace`, `Background`, `ButtonFace`, `ButtonHighlight`, `ButtonShadow`, `ButtonText`, `CaptionText`, `GrayText`, `Highlight`, `HighlightText`, `InactiveBorder`, `InactiveCaption`, `InactiveCaptionText`, `InfoBackground`, `InfoText`, `Menu`, `MenuText`, `Scrollbar`, `ThreeDDarkShadow`, `ThreeDFace`, `ThreeDHighlight`, `ThreeDLightShadow`, `ThreeDShadow`, `Window`, `WindowFrame` a `WindowText`.

Klienty přitom mohou barvy přizpůsobit svým možnostem (zaokrouhlením podle barevného rozlišení a úpravou podle použitého gamutu a nastavení *gama* obrazovky, převodem na prostor CMYK na tiskárně atd.). V nově připravovaných verzích CSS3 se již objevuje i *správa barev,* v současnosti ale rozhodně nelze počítat s žádným věrným zobrazováním námi předepsaných barev. **Není proto vhodné v dokumentech poskytovat informace, které jsou závislé na přesném odstínu barvy** (vzorníky barev atd.), pro takové údaje je vždy lepší použít samostatný soubor, který by věrnou interpretaci barev umožňoval.

Částečnou jistotu poskytují pouze tzv. *bezpečné barvy pro web (Web Safe Colors).* Je to podmnožina barevného prostoru RGB tvořená hodnotami zadanými hexadecimálně jen pomocí čísel `00`, `33`, `66`, `99`, `CC` a `FF` (6□6□6 = 216 barev; např. `#FF9933`, `#0C9`). Prohlížeče by tyto barvy měly zobrazovat přednostně co nejvěrněji, ani to však není stoprocentně zaručeno.

3.3.7 <řetězec>

Řetězce reprezentují nějaký text či skupinu znaků a zapisují se do jednoduchých (`'řetězec'`) nebo dvojitých (`"řetězec"`) uvozovek. Znaky, které se v řetězci nesmí vyskytovat (např. stejné uvozovky jako ty, jimiž je řetězec omezen, či odřádkování), se uvozují znakem `\` — např. `'Hell\'s□Bells'`, `"Řekla:□\"Ahoj!\""`. Odřádkování se označuje sekvencí `\A`, např. `"první□řádek\Adruhý□řádek"`. Potřebujeme-li zapsat přímo znak `\`, musíme použít sekvenci `\\` — např. `"soubor□C:\\texty\\132.txt"`.

U delších řetězců můžeme chtít kvůli přehlednosti rozdělit text na více řádků. Odřádkování se sice v textu neprojeví, musíme však vždy na konci řádků uvést opět znak `\`. Např.:

```
P.pozn:before {
      content:
              "Tady je nějaký text\
              ale stejně by se mi nevešel na\
              jeden řádek, tak jsem jej pro\
              přehlednost rozdělil." }
```

3.3.8 <úhel>, <čas> a <frekvence>

Tyto jednotky se v CSS používají pouze ve vlastnostech zvukových stylů. **Úhly** se zapisují ve tvaru `<číslo>jednotka`, kde jednotkou je buďto `deg` (stupně), `rad` (radiány), nebo `grad` (grady). Úhel může být i záporný, hodnota se potom převádí do intervalu 0–360° (např. `-10deg` = `350deg` = `6.1087rad` = `388.9grad`). **Čas** se definuje ve tvaru `<číslo>s` (sekunda) nebo `<číslo>ms` (milisekunda), hodnoty nemohou být záporné. Např.: `120ms`, `5s`. **Frekvence** jsou obdobně ve tvaru `<číslo>Hz` (Hertz) nebo `<číslo>kHz` (kiloHertz). Ani hodnoty frekvencí nesmějí být záporné. Např.: `2400Hz`, `4.2kHz`.

3.4 Média

Určení, jakým způsobem se má zobrazit dokument na různých typech koncových zařízení, je jedním z hlavních úkolů kaskádových stylů. Tato výstupní zařízení se označují jako **média** a CSS je rozděluje do několika skupin. Pro každé *médium* je potom možné definovat jiný styl prezentace dokumentu. Různé typy *médií* někdy vyžadují různé hodnoty jednotlivých vlastností. Některá *média* mají dokonce definovány vlastnosti, které lze použít pouze pro ně — zvukové styly pro hlasový výstup, vlastnosti stránky pro tiskárnu atd.

3.4.1 Typy médií v CSS

CSS v současnosti používá následující názvy médií. Tímto názvem autor definuje styl pro dokumenty použité na příslušném koncovém zařízení:

- **all:** všechny typy zařízení
- **aural:** zvukový výstup na hlasovém syntezátoru
- **braille:** braillská dotyková zařízení (*braillský řádek*)
- **embossed:** stránka zobrazená plastickým tiskem na braillské tiskárně
- **handheld:** obrazovka kapesního počítače (typicky malý monochromatický displej, pomalejší přenos dat)
- **print:** stránky vytištěné na tiskárně či dokumenty zobrazené na obrazovce při náhledu tisku
- **projection:** promítané prezentace (např. zpětné či velkoplošné projektory)
- **screen:** obrazovka počítače (s možností použití barev a bitmapové grafiky)
- **tty:** neproporční znakový výstup (dálnopisy, terminály apod.)
- **tv:** televizní obrazovky a podobná zařízení (nízké rozlišení, omezené možnosti posouvání obsahu, možnost použití barev a zvuku).

Jednotlivá média jsou **navzájem neslučitelná.** Tzn. že každý klient musí při zpracování dokumentu podporovat **pouze jediný typ média.** To ovšem nevylučuje, že klienty mohou mít více modulů či režimů, z nichž každý podporuje jiné médium (vždy ale pouze jediné). Typický prohlížeč tak např. běžně pracuje s médiem *screen,* ale obsahuje i tiskový modul, který zase pracuje pouze s médiem *print.* Má-li zvukový výstup, pracuje v tomto režimu pouze s *aural* atd.

Zvuková média *aural* mají mnoho specifik a podrobněji se jim věnuje zvláštní kapitola. Obdobně se zvláštním způsobem používají kaskádové styly ve stránkovaných médiích (*emboss, print, projection*), práce s nimi však ještě není v CSS jednoznačně definována a uspokojivé řešení by měla přinést až specifikace CSS 3. Na médiích *tty* nelze používat jednotky `px`.

3.4.2 Skupiny médií

Každou vlastnost CSS můžeme použít pouze pro určitou **skupinu médií.** V popisu vlastnosti je proto vždy uvedeno, pro kterou skupinu má její definování smysl. Vlastnost pak platí pro všechna média, která do této skupiny patří. CSS definuje několik takových skupin, jež rozdělují média podle různých hledisek.

- **Média stránkovaná–plynulá:**
  - **stránkovaná:** emboss, handheld, print, projection, tv
  - **plynulá:** aural, braille, handheld, screen, tty, tv
- **Média zvuková–vizuální–dotyková:**
  - **zvuková:** aural, tv
  - **vizuální:** handheld, print, projection, screen, tty, tv
  - **dotyková:** braille, emboss
- **Média s pevnou mřížkou–bitmapová:**
  - **s pevnou mřížkou:** braille, emboss, handheld, tty
  - **bitmapová:** handheld, print, projection, screen, tv
- **Média interaktivní–statická:**
  - **interaktivní:** aural, braille, handheld, projection, screen, tty, tv
  - **statická:** aural, braille, emboss, handheld, print, projection, screen, tty, tv

3.4.3 Použití médií v tabulkách stylů

K dokumentu můžeme připojit tabulku stylů, která má platit pouze pro dané médium. V rámci jedné tabulky pak lze vyznačit pravidla určená konkrétnímu médiu pomocí pravidla *@import* nebo příkazem *@media.*

3.4.3.1 Specifikování média v dokumentech

Jazyky dokumentů rozpoznávají atribut `media` ve značkách `<link>` a `<style>`. Jako hodnotu tohoto atributu můžeme definovat seznam médií (oddělených čárkou). Tabulka stylů se pak použije pouze pro tato média. Např.:

```
<link rel="stylesheet" type="text/css" href="styl.css" media="screen, h
andheld">

<style type="text/css" media="print">
...
</style>
```

Podle specifikace HTML 4 má sice být výchozí hodnotou `media="screen"`, prohlížeče se však většinou chovají, jako by to byla hodnota `"all".` V XHTML je výchozí hodnotou `"all"`.

3.4.3.2 Příkaz @media a média v příkazu @import

Pro definování médií přímo v tabulce stylů můžeme použít příkaz **@media.** Tímto příkazem označíme část tabulky stylů jako platnou pouze pro daná média. Zapisuje se ve tvaru:

```
@media seznam_médií {
      ...
      pravidla platná pro tato média
      ...
}
```

Například:

```
@media screen {
      body { font-family: sans-serif; font-size: 12pt }
}
@media print, projection {
      body { font-family: serif; font-size: 10pt }
}
@media handheld, tv {
      body { font-size: medium }
}
```

Uvnitř bloku `@media` již nesmíme importovat další tabulky příkazem `@import`. Namísto toho můžeme použít druhou možnost, a to definování média přímo v příkazu `@import`, ve tvaru:

```
@import url(adresa) seznam_médií;
```

Tedy např.:

```
@import url("styl_tisk.css") print;
```

3.5 Strom dokumentu

Protože CSS pracuje se strukturou dokumentu a často se na ni odkazuje, je zde na místě připomenout i několik základních informací o **stromu dokumentu.**

Vyznačovací jazyky, jako je HTML či XML, popisují dokument pomocí stromové struktury. Pomocí vzájemně vnořených značek uzavírají části dokumentu do určitých celků a vytvářejí tím jeho **hierarchii.** Aby bylo snazší tuto hierarchii popisovat a pohybovat se v ní, používá se terminologie, která je blízká jednak názvosloví biologickému (analogie se stromem, kořenem a větvemi), jednak genealogickému — vztahy příbuznosti a dědičnosti v lidských pokoleních mají ostatně také stromovou strukturu.

Každý dokument vždy vychází z jediného prvku, který tvoří **kořen dokumentu** a obsahuje pak všechny prvky ostatní (jsou v něm vnořeny). V případě HTML je kořenem dokumentu prvek `<html>`. Všechny prvky, které obsahuje (a prvky v nich) pak tvoří jednotlivé **větve** stromu dokumentu.

Prvek, který obsahuje další prvky, se nazývá jejich **rodič,** prvky v něm přímo vnořené pak označujeme jako jeho **potomky.** Prvky, které jsou vnořené v těchto potomcích, již nenazýváme *potomky,* ale pouze **následovníky** původního prvku; ten je potom jejich **předek.**

Pokud tedy prvek A obsahuje prvek B a ten zase obsahuje prvek C, potom A je *rodič* B a B je *potomek* A — a také B je *rodič* C a C je *potomek* B. Prvek C však již není *potomkem* A (a A není *rodičem* C). Prvky B i C jsou přitom stále *následovníky* A a A je jejich *předkem.* Každý *potomek* jiného prvku je tedy současně i jeho *následovníkem,* ale ne všichni *následovníci* jsou jeho (přímými) *potomky.*

Dva *potomci* stejného rodiče se nazývají **sourozenci** (tj. dva prvky obsažené na stejné úrovni v jednom prvku). Prvek A se nazývá **předcházející sourozenec** B, pokud se nachází před prvkem B ve stromu dokumentu (a je současně jeho sourozencem); naopak A je **následující sourozenec** B, pokud je ve stromu dokumentu až za ním. Sourozenecké prvky A a B jsou **sousední sourozenci,** pokud se mezi nimi nenachází žádný další jejich sourozenec.

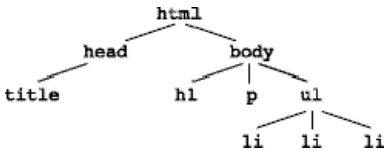
Prvek A se nazývá **předcházející prvek** prvku B, pokud je jeho *předcházejícím sourozencem,* nebo jeho *předkem.* Prvek A je pak **následující prvek** prvku B, pokud B je *předcházejícím prvkem* prvku A.

Příklad

Na následující jednoduché stránce HTML si ukážeme *strom dokumentu* a některé vztahy, které v něm platí:

```
<html>
<head>
      <title>Moje stránka</title>
</head>
<body>
      <h1>Vítejte na mé stránce</h1>
      <p>Na této stránce se především
      píše o muzice, kterou mám rád.
      Nejraději mám kapely:</p>
      <ul>
              <li>Pink Floyd</li>
              <li>Deep Purple</li>
              <li>Led Zeppelin</li>
      </ul>
</body>
</html>
```

Strom tohoto dokumentu vypadá takto:



Obr. 18 — Strom dokumentu

Prvek `html` je *kořenem dokumentu*. Má dva *potomky:* `head` a `body`. Prvek `head` má jediného potomka `title`, `body` už má potomky tři — `h1`, `p` a `ul`.

Prvky `h1`, `p`, `ul` a `li` jsou *následovníky* `body`. Všechny prvky `li` jsou navzájem *sourozenci,* jejich *rodičem* je `ul`. Obdobně jsou *sourozenci* prvky `h1`, `p` a `ul`.

Poslední prvek `li` má dva *předcházející sourozence* (dva prvky `li`). *Předcházejícími prvky* posledního prvku `li` jsou ostatní dva `li` a také prvky `ul`, `body` a `html`. Prvky `h1` a `p` a prvky jsou `p` a `ul` jsou *sousední sourozenci*. Prvky `h1` a `ul` však sousedními sourozenci nejsou (vyskytuje se mezi nimi ještě `p`).

3.6 Selektory v tabulkách stylů

Jádro tabulek stylů tvoří tzv. *pravidla*. Ta přiřazují definice stylu konkrétním částem dokumentu pomocí dvojice `selektor { definice }`. Takové pravidlo vyjadřuje, že uvedená *definice* se bude vztahovat pouze na prvky nebo úseky dokumentu, **které vyhovují zadanému selektoru.**

Selektor (angl. *selector,* česky přibližně *volič, přepínač*) je symbolický popis prvku či skupiny prvků, které se v dokumentu mohou vyskytovat. Pokud nějaký prvek v dokumentu danému selektoru *vyhovuje,* bude se na něj vztahovat styl definovaný v tomto pravidle. Každý prvek ale může současně vyhovovat více pravidlům — v tom případě se ještě použijí *pravidla kaskádování a dědičnosti,* popsaná v kapitole 3.7.

Selektorů je více druhů a existuje pro ně i několik *operátorů,* pomocí nichž lze vyjadřovat vztahy mezi selektory a jejich kombinace. Tím můžeme přesněji popsat prvky dokumentu, na něž se má styl vztahovat — např. omezit pravidlo pouze na prvky v určitém kontextu.

3.6.1 Základní selektory

3.6.1.1 Univerzální selektor

Nejobecnějším ze selektorů je tzv. **univerzální selektor.** Označuje se symbolem „`*`“ (hvězdička) a vyhovují mu **všechny prvky dokumentu.** Pokud tedy uvedeme nějaké pravidlo se selektorem `*`, znamená to, že jeho definice budou platit pro všechny prvky na stránce.

Příklad

Pravidlo v tabulce stylů:

```
* { font-size:small }
```

definuje, že všechny prvky v dokumentu budou mít vlastnosti `font-size` přiřazenu hodnotu `small`. Všechny texty na stránce se zobrazí malým písmem.

3.6.1.2 Selektor typu

Tomuto selektoru vyhovují **všechny prvky daného typu značky** jazyka (HTML, XML). Označuje se **názvem značky,** např. selektoru `h1` vyhovují všechny prvky definované značkou `<h1>` v dokumentu. Selektoru `div` odpovídají všechny prvky `<div>`, selektor `PRIJMENI` definuje styl pro všechny prvky `<PRIJMENI>` v dokumentu XML atd.

Příklad

Pravidlu v tabulce stylů:

```
img { border: 1px solid red }
```

vyhoví prvky typu `<img>`, všechny obrázky proto budou mít definovánu vlastnost `border` (styl rámečku).

3.6.2 Rozšířené selektory (s atributy)

Značky HTML i XML mohou kromě názvu obsahovat i další údaje, tzv. *atributy.* Např. ve značkách `<img src="abc.gif" alt="abc">` či `<a href="xyz.html">` jsou atributy `src`, `alt` a `href`. V CSS existují nástroje pro definování stylů i podle těchto atributů. *Základní selektory* mohou být omezeny vlastností nějakého atributu — vyhovují jim potom pouze ty prvky, které mají nejen odpovídající název značky, ale i atribut vyhovující zadání. Možnosti pro použití atributů v selektorech jsou popsány níže, specifikují se ale vždy v hranatých závorkách `[]` připojených za název značky (bez mezery). Atributy je možné použít i s univerzálním selektorem (`*[]`), takový selektor potom popisuje všechny prvky, které mají atribut daných vlastností.

3.6.2.1 Existence atributu

Selektoru `X[atribut]` vyhovují ty prvky typu `X`, které **mají definován uvedený atribut**. Na hodnotě tohoto atributu přitom již nezáleží, podstatná je pouze jeho existence ve značce.

Příklad 1

```
img[title] {...}
```

definuje styl pro všechny prvky `<img>`, které mají definován atribut `title`, např. `<img src="abc.gif" alt="abc" title="Popis obrázku">`. Na prvky `<img>`, které atribut `title` nemají určen, se toto pravidlo nebude vztahovat.

Příklad 2

```
*[alt] {...}
```

popisuje všechny prvky (jakéhokoli typu), které mají (jakkoli) definován atribut `alt`.

3.6.2.2 Atribut dané hodnoty

Pokud je potřeba ověřit nejen existenci atributu, ale i jeho hodnotu, slouží k tomu selektor `X[atr=hodn]`. Vyhovují mu prvky typu `X`, jejichž atribut `atr` **má hodnotu přesně rovnu** `hodn`. Hodnotami atributů jsou buďto *identifikátory* (např. *id* nebo *jméno třídy*), nebo *řetězce.* Hodnoty, které nejsou identifikátory, proto musí být zapsány v uvozovkách (tedy i číselné hodnoty).

Příklad

```
table[border="0"] {...}
```

Tomuto pravidlu vyhoví pouze ty prvky `table`, které mají definován atribut `border` s hodnotou `0`. Prvky s jinou hodnotou `border`, nebo ty, které tento atribut nemají definován vůbec, selektoru nevyhovují a pravidlo se na ně nebude vztahovat.

3.6.2.3 Atribut obsahující danou hodnotu

Selektoru `X[atr~=hodn]` vyhoví ty prvky typu `X`, jejichž atribut `atr` **obsahuje seznam hodnot** (oddělených mezerami) **a jednou z nich je přesně** `hodn`. Protože mezera je zde oddělovačem, `hodn` již mezeru obsahovat nesmí.

Příklad

```
*[class~=nadpis] {...}
```

definuje styl pro všechny prvky, jejichž jednou z hodnot atributu `class` je hodnota `nadpis`. Pravidlu vyhoví např. prvky `<h1 class="nadpis">` a `<h4 class="cerveny nadpis ttl4">`. Prvek `<h1 class="cervenynadpis">` však již tomuto pravidlu nevyhovuje.

3.6.2.4 Atribut obsahující danou podhodnotu

Selektor `X[atr|=hodn]` je definován obdobně jako předchozí varianta. Vyhovují mu však ty prvky, jejichž atribut `atr` obsahuje seznam hodnot oddělených spojovníkem (znak `-`) a **první z nich** je právě hodnota `hodn`. Tento selektor je primárně určen k rozlišování jazykových variant, které jsou specifikovány právě takovým zápisem — např. `en`, `en-US`, `en-cockney`. Selektoru `X[lang|="en"]` pak vyhoví všechny prvky, které mají v atributu `lang` definovánu kteroukoli z výše uvedených variant anglického (`en`) jazyka.

3.6.2.5 Vícenásobné selektory s atributy

Atributy v selektorech mohou být vícenásobné. Poslouží tak k současnému odkazování na více atributů daného prvku či na více možných hodnot jednoho atributu. Např. pravidlu:

```
a[lang="cs"][title] {...}
```

vyhoví ty prvky `a`, které mají `lang="cs"` a současně mají definován atribut `title`.

3.6.2.6 Selektory tříd v HTML

V jazyce HTML (resp. XHTML) se hojně používá atribut `class` pro určení třídy daného prvku. Chceme-li tedy např. definovat pravidlo pro prvky `DIV` s `class="trida"`, použijeme selektor `DIV[class~=trida]`. Protože se však třídy používají opravdu často a takový selektor by nebyl příliš praktický, byl do CSS přidán i zjednodušený zápis `DIV.trida` (tečka následovaná názvem třídy, bez mezery). Oba zápisy jsou však ekvivalentní a vyjadřují totéž.

Příklad

Pravidla v tabulce stylů:

```
h1.nadpis {color:red}
h1[class~=nadpis] {color:red}
```

jsou totožná a obě definují barvu textu pro prvky `<h1 class="nadpis">`.

Pozn.: Chceme-li specifikovat všechny prvky stejné třídy, lze použít *univerzální selektor.* Ve spojení se zkráceným zápisem pro třídu však není nutný a je možné jej vypustit. Např. selektory `*.nadpis` a `.nadpis` jsou totožné.

3.6.3 Selektory ID

Jazyky dokumentů mohou ve svých prvcích používat atributy označené jako *identifikátor prvku* (atribut typu ID). Takový atribut se od ostatních liší především tím, že **musí být unikátní,** tj. že v dokumentu **nesmí existovat dva prvky se stejným identifikátorem** a **jeden prvek nesmí mít více identifikátorů.** V jazyce HTML je identifikátorem vždy atribut `id`, v XML se informace o tom, který atribut je identifikátorem prvků, definuje v příslušném DTD. Pomocí ID je tak možné jednoznačně pojmenovat prvky dokumentu a pomocí CSS jim pak definovat styl.

Obdobně jako u tříd můžeme použít *selektor s atributem* — např. pro prvek `<div id="prvek123">` lze definovat selektor `div[id=prvek123]`. A opět je pro zjednodušení v CSS možný i kratší zápis, tentokrát ve tvaru `div#identifikator` (dvojkřížek následovaný identifikátorem, bez mezery), tedy např. `#prvek123`.

Oba zápisy však již nejsou zcela totožné. I když popisují totéž (prvek s daným ID), druhý, zkrácený zápis má vyšší prioritu z hlediska *kaskádování* [3.7].

Protože identifikátor musí být unikátní pro každý prvek a na stránce se nesmí vyskytovat dva prvky se stejným ID, není nutné uvádět jméno značky a selektor tak může začít rovnou znakem `#`.

Příklad

```
h1#nadpis12 {...}
*#nadpis12 {...}
#nadpis12 {...}
```

Všechna pravidla popisují totéž, ale protože s atributem `id="nadpis12"` smí být na stránce jen jediný prvek, postačí jenom poslední, nejjednodušší zápis — první dva jsou zbytečné.

3.6.4 Kombinování selektorů (operátory)

Selektory je možné vzájemně kombinovat a vytvářet tak selektory složitější. Kombinované selektory umožňují zpřesňovat prvky v tabulce stylů podle jejich kontextu a vazeb na okolí, či naopak rozšiřovat pole jejich působnosti. Selektory se kombinují pomocí **operátorů** `>` , `+` a

mezera, z nichž každý má jiný význam. Takto vzniklé selektory můžeme dále kombinovat a slučovat a vytvářet tak i velmi složité, podrobné selektory.

3.6.4.1 Slučování selektorů

Pomocí operátoru `,` (čárka) je možné do jednoho pravidla sloučit více selektorů, které mají mít shodnou definici stylu. Autorům to může ušetřit mnoho času i námahy, tabulky stylů se díky slučování stávají podstatně přehlednějšími. Tento operátor nijak nemění význam či kontext jednotlivých selektorů, pouze jim přiřazuje jedinou společnou definici.

Příklad

Sadu pravidel pro několik různých selektorů v tabulce stylů:

```
h1 {color:blue}
h2 {color:blue}
h3 {color:blue}
.modrytext {color:blue}
```

lze sloučit do pravidla jediného:

```
h1, h2, h3, .modrytext {color:blue}
```

které má stejný účinek jako předchozí seznam čtyř pravidel.

Pozn.: Tento postup je výhodné použít i tehdy, má-li mít několik selektorů **jen některé** společné vlastnosti a v ostatních se liší. Nejprve nadefinujeme společné vlastnosti a poté dalšími pravidly doplníme další vlastnosti jednotlivým selektorům — např.:

```
h1, h2, h3 {color:blue;margin:1ex 0;background:yellow}
h2 {font-size:180%}
h3 {font-size:150%}
```

Pomocí *předefinování v kaskádě* [3.7] můžeme také libovolnou ze společně nadefinovaných vlastností kterémukoli ze selektorů později změnit.

Pozn.: Před i za operátory může být libovolná *mezera (prázdný prostor).* Selektory v následujících pravidlech jsou proto totožné:

```
h1,h2,h3{...}

h1,
h2,
h3    {...}

      h1,h2 , h3
      {...}
```

3.6.4.2 Následovnické selektory

Pomocí následovnických selektorů dovoluje CSS autorům specifikovat prvky podle kontextu (viz *strom dokumentu* [3.5]). Chceme-li např. definovat vlastnosti „pouze pro ty prvky `strong`, které jsou uvnitř nějakého prvku `h1`“, poslouží k tomu **operátor mezera.** Oddělíme-li dva (či více) selektory A a B mezerou (resp. *prázdným prostorem*), vytvoříme nový, kombinovaný selektor, který definuje, že **B musí být následovníkem A** (neboli A je *předkem* B). Např.:

```
h1 strong {color: red}
```

Tento selektor popisuje výše uvedený příklad, vyhoví mu pouze ty prvky `strong`, které jsou následovníky prvku `h1` (jsou kdekoli uvnitř `h1`).

Dílčích selektorů lze použít více za sebou a popsat tak i složitější struktury v dokumentu — např. selektoru `div p em a` vyhoví jen takové prvky `a`, které jsou uvnitř nějakého prvku `em`, který je uvnitř prvku `p`, a ten musí být uvnitř nějakého prvku `div`.

Můžeme použít všechny typy selektorů, nejen *selektory typu.* Například selektoru `#prvek123 * .odstavec` vyhoví pouze takové prvky třídy `odstavec`, které jsou umístěny uvnitř dalšího prvku v prvku s `id="prvek123"` (jsou to *následovníci* nějakého *následovníka* prvku `#prvek123`; nejsou tedy *potomky* prvku `#prvek123`).

Příklady

```
#prvek123 input {...}
h1 a.externi {...}
div.kapitola p a[lang|="en"] {...}
```

Pozn.: Je důležité dávat si pozor mezery v selektorech, především při použití tříd. Selektoru `p.odstavec` odpovídají prvky `<p class="odstavec">`, zatímco selektor `p .odstavec` popisuje libovolný prvek třídy `odstavec`, který je následovníkem nějakého prvku `p`.

3.6.4.3 Selektory potomků

Obdobně jako následovnické selektory jsou definovány i selektory potomků. Pomocí operátoru `>` mezi selektory A a B určujeme, že **B musí být potomkem** (nikoli jen následovníkem) A. Prvek A tedy musí být *rodičem* prvku B.

Příklad

Selektoru `p > em` vyhoví pouze takové prvky `em`, které jsou (přímým) *potomkem* nějakého prvku `p` (tedy takové prvky `em`, jejichž *rodičem* je prvek `p`). Např. v případě konstrukce HTML:

```
<p>Nějaký odstavec se <em>zvýrazněným</em> slovem.</p>
<p>Druhý odstavec, <i>kde je <em>zvýraznění</em> hlouběji.</i></p>
```

vyhoví tomuto selektoru pouze prvek `em` v prvním odstavci, protože v odstavci druhém již není *potomkem* prvku `p` (je potomkem prvku `i`). *Následovnickému selektoru* `p em` by však už vyhovovaly oba prvky `em`.

Selektory i operátory je možné vícenásobně kombinovat a používat všechny druhy dílčích selektorů.

Příklady

```
div > p > a {...}
div > p a {...}
#prvek123>input {...}
div.kapitola p>a strong {...}
```

3.6.4.4 Selektory sousedních sourozenců

Pomocí operátoru `+` (plus) se definují selektory pro sousedící prvky. Selektoru `A + B` vyhovuje prvek B, pokud je **následujícím sousedním sourozencem** prvku A. Jinými slovy, prvky A i B musí mít stejného *rodiče* a prvek A musí *přímo předcházet* prvku B.

Příklad

V konstrukci HTML:

```
<body>
      <h1>Nadpis 1</h1>
      <p>Text ... text</p>
      <p>Text ... text</p>
      <p>Text ... text</p>
      <h1>Nadpis 2</h1>
      <div>
            <p>Text ... text</p>
      </div>
</body>
```

vyhovuje selektoru `h1+p` první prvek `p`, ostatní již nejsou sousedními sourozenci `h1`. Poslední prvek `p` dokonce vůbec není sourozencem `h1`. Selektoru `p+p` by v uvedeném příkladu vyhovovaly druhé dva prvky `p`, protože oba jsou *následujícími sourozenci* nějakého předchozího prvku `p`.

Pomocí tohoto selektoru je možné vytvářet např. oddělovače **mezi** stejnými prvky — v libovolném seznamu `ul`/`ol` zajistí pravidlo

```
li+li {border-top: 1px solid black}
```

linku *mezi* položkami seznamu — definujeme zde linku *nad* každým prvkem `li`, pokud mu předchází jiný prvek `li` na stejné úrovni. Nad první položkou seznamu tedy linka nebude, neboť jí žádný už prvek `li` nepředchází.

Příklady

Stejně jako v předchozích případech je možné operátory navzájem kombinovat:

```
h1.nadpis+h2 {...}
div.kapitola>h1+p {...}
h1+h2+h3 {...}
#prvek123 p+p {...}
```

3.6.5 Pseudo-třídy a pseudo-prvky

Kromě výše uvedených selektorů, které vycházejí z informací o uspořádání prvků ve *stromu dokumentu,* existuje v CSS ještě skupina selektorů. která jde nad rámec těchto informací. Jsou to selektory, které se v CSS nazývají **pseudo-třídy** a **pseudo-prvky.**

Předpona *pseudo-* značí jistou *zdánlivost, ne-skutečnost* a tyto selektory také vytvářejí abstraktní, ve skutečnosti neexistující třídy a prvky, které nejsou přímo odvozeny ze stromu dokumentu.

Pseudo-prvky rozšiřují strom dokumentů o prvky, které nelze popsat prostředky samotného jazyka. V jazyce HTML např. neexistují informace definující první řádek textu či první znak odstavce, selektory CSS je však dokáží popsat. Dávají také autorům možnost přidávat obsah, který se v samotném dokumentu vůbec nevyskytuje (pseudo-prvky `:before` a `:after`).

Pseudo-třídy umožňují rozlišit prvky i podle dalších charakteristik, než je jejich jméno a atributy, v podstatě jde opět o informace, které většinou nelze přímo vyčíst ze struktury dokumentu. Pseudo-třídy mohou být i dynamické, prvky se do takových tříd dostávají či z nich vystupují během prohlížení stránky, třeba v závislosti na činnosti uživatele.

Pozn.: Pseudo-třídy se mohou vyskytovat jako samostatné selektory, zatímco pseudo-prvky lze použít pouze jako doplněk jiného selektoru (typu, třídy, ID).

3.6.5.1 Pseudo-třídy

3.6.5.2 :first-child

Prvek má automaticky přiřazenu pseudo-třídu `:first-child`, pokud je **prvním potomkem** jiného prvku. Pokud má např. nějaký prvek na stránce více *potomků* `<p>`, selektoru `p:first-child` bude vyhovovat pouze první z nich.

Příklady

Chceme-li zobrazit první položku v seznamech `UL` tučně, můžeme použít:

```
ul>li:first-child {font-weight:bold}
```

První odkaz v každém oddíle `<div class="oddil">` můžeme popsat takto:

```
div.oddil a:first-child {...}
```

3.6.5.3 Pseudo-třídy odkazů :link a :visited

Prohlížeče obvykle umožňují odlišit formátování odkazů podle toho, zda již adresa, na niž vedou, byly navštívena, či nikoli. V CSS je tato informace dostupná pomocí *pseudo-tříd odkazů.* Prvek `a` je v pseudo-třídě `:link`, pokud **adresa jeho odkazu ještě nebyla navštívena.** V opačném případě se nachází v pseudo-třídě `:visited`. Jak je vidět, každý odkaz je vždy v jedné z těchto dvou tříd — jinou možnost nemá.

Příklad

Na stránkách budeme rozlišovat odkazy v rámci našeho webu a odkazy externí (ty budou označeny pomocí `class="extern"`). U „vnitřních“ odkazů nechceme nijak vizuálně odlišovat navštívené a nenavštívené adresy (třeba proto, že všechny stránky jsou generovány pomocí skriptu a prohlížeč by navštívené adresy nemusel rozpoznat). U externích odkazů to ale rozlišit chceme a budeme požadovat, aby navštívené odkazy byly zobrazeny přeškrtnuté. Interní odkazy mají být červené, externí zelené.

```
a {color:red}
a.extern {color:green}
a.extern:visited { text-decoration:line-through}
```

Pozn.: Selektor bez uvedení třídy popisuje dané prvky všech tříd (tedy i pseudo-tříd). První pravidlo proto definuje červenou barvu textu **všech** odkazů `a` (tudíž i odkazů `a.extern` a `a.extern:visited`). Teprve následující pravidla dodatečně mění vlastnosti pro další, již zúžené skupiny prvků `a`.

3.6.5.4 Dynamické pseudo-třídy :hover, :active, :focus

Některé prohlížeče umožňují měnit **formátování prvků v závislosti na činnosti uživatele.** CSS pro zpracování tří nejběžnějších uživatelských akcí poskytuje dynamické pseudo-třídy, neinteraktivní klient (prohlížeč) je bude ignorovat.

- pseudo-třída `:hover` se přiřadí prvku, když na něj uživatel *ukáže* (nějakým polohovacím zařízením), ale neaktivuje jej. V běžných prohlížečích na počítači je to typicky ve chvíli, kdy se kurzor myši dostane nad plochu daného prvku. Když ukazatel plochu prvku opustí, prvek ztratí i třídu `:hover`.
- pseudo-třídu `:focus` má prvek, který získal *zaměření*, tj. jsou do něj směřovány události klávesnice či jiného vstupu. V běžných prohlížečích obvykle získávají *zaměření* pouze prvky formulářů, případně i odkazy (podle nastavení a typu prohlížeče). *Zaměření* se v prohlížečích obvykle přesouvá tabulátorem, klávesovou zkratkou či klepnutím na prvek `label` ve formulářích.
- pseudo-třída `:active` se přiřadí prvku, když byl aktivován — např. uživatel na něj *klepne* myší nebo stiskne *enter,* pokud měl prvek *zaměření* atd.

Prvky přitom mohou být ve více z uvedených stavů současně. CSS však nespecifikuje, jak se tyto stavy mají aktivovat či které prvky v nich mohou být. To je již v kompetenci použitého prohlížeče. Při návrhu stránek je proto potřeba počítat s tím, že uživatelův prohlížeč může podporovat dynamické pseudo-třídy jiným způsobem (či vůbec ne) než prohlížeč náš. Uživatel také může mít jiné návyky a preference, než máme my — například se po stránce pohybuje zásadně pomocí klávesnice nebo úplně jiného zařízení, třeba vůbec nepoužívá myš. **Není proto vhodné pomocí dynamických pseudo-tříd definovat nějaké zásadní informace důležité pro použití stránky** (např. není dobře zvýraznit jinak neznatelné odkazy až pomocí `:hover`). Dokumenty musí být stejně dobře použitelné i bez nich.

Prohlížeče také nejsou povinny přeformátovat dokument kvůli změnám definovaným pomocí dynamických pseudo-tříd. Předepíšeme-li např. odkazům ve stavu `:hover` větší písmo, klient může takové definice ignorovat, pokud by kvůli nim musel změnit formátování ostatních prvků — větší písmo by mohlo odsunout následující text a způsobit tak *překreslení* celé stránky.

Zvláště při definování pseudo-tříd pro odkazy `a` je důležité pamatovat na *pravidla kaskády* [3.7.3] a uvádět pravidla ve vhodném pořadí. Používáme-li současně např. `a:link`, `a:hover` i `a:active`, musíme počítat s tím, že jejich špatné pořadí v tabulce stylů může mít nechtěné následky. Každý odkaz je totiž buďto `:link`, nebo `:visited` (navštívený, či nikoli), ale jen některé z nich mohou mít **také** stav `:hover`. A zase jen některé z nich mohou být `:active` — a aktivujeme-li odkaz, je často také ve stavu `:hover` (ukazujeme na něj myší) nebo `:focus` (byl označen pomocí klávesnice), nebo dokonce v obou; přitom má navíc ještě také `:link`, nebo `:visited`.

Pravidla s pseudo-třídami by se proto měla uvádět v pořadí od nejobecnějších po konkrétnější, tedy v pořadí `:link` a `:visited`, teprve za nimi `:hover`, `:focus` a nakonec `:active`.

Příklad

```
a:link      {color:red }        /* nenavštívené odkazy */
a:visited   {color:brown}       /* navštívené odkazy */
a:hover     {color:green}       /* po najetí kurzoru */
a:active    {color:black}       /* aktivovaný odkaz */
```

Protože prvky mohou být současně ve více stavech — např. najedeme kurzorem nad vstupní pole formuláře, které už má *zaměření* — je možné kombinovat i příslušné pseudo-třídy. Pro uvedený případ to tedy může být:

```
input:focus:hover {...}
```

3.6.5.5 Jazyková pseudo-třída :lang

Dokumenty HTML či XML umožňují specifikovat použitý jazyk. V XML se tak děje např. pomocí atributu `xml:lang`, v HTML 4 pomocí kombinace atributů `lang` v prvcích, údaji v značkách META a dat z hlavičky protokolu HTTP. V CSS je možné s touto informací pracovat pomocí pseudo-třídy `:lang`.

Zapisuje se jako funkce, s parametrem kódu jazyka ve tvaru `:lang(jaz)` (parametr `jaz` se chová obdobně jako u operátoru `|=` v atributech). Tomuto selektoru pak vyhovují všechny prvky, které mají definován jazyk `jaz`.

Příklad

Pomocí jazykové pseudo-třídy můžeme např. definovat různé typy uvozovek pro různé jazyky použité v dokumentu:

```
Q:lang(en) { quotes: '“' '”' }
Q:lang(cs) { quotes: '„' '“' }
Q:lang(fr) { quotes: '«' '»' }
```

3.6.5.6 Pseudo-prvky

3.6.5.7 Pseudo-prvek :first-line

Pseudo-prvek `:first-line` umožňuje definovat **styl pro první řádek textu.** Takový prvek se ve stromu dokumentu vůbec nevyskytuje, protože jeho podoba závisí na výsledném zformátování dokumentu. Mohli bychom sice vytvořit podobný prvek:

```
<p>
<span class="prvniradek">Toto je první řádek odstavce</span>
Další řádek a další text ... text text text ...
Další řádek a další text ... text text text ...
Další řádek a další text ... text text text ...
</p>
```

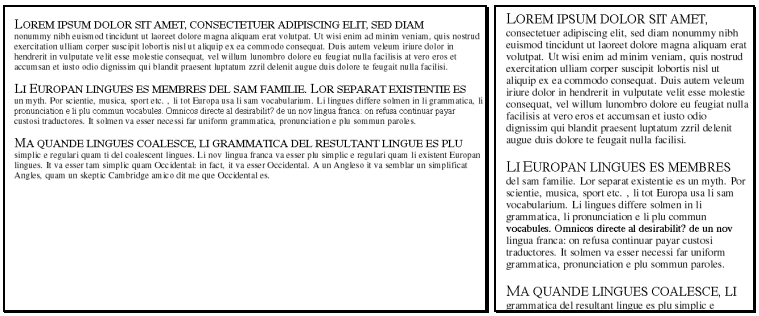
jenže to určitě není to pravé — pokud bude stránka zobrazena ve velmi širokém okně či malým písmem, může se stát, že se celý odstavec vejde na jediný řádek. Na hodně úzkém prostoru nebo s velkým písmem se může naopak na jeden řádek vejít pouze jedno slovo. K vyřešení tohoto dilematu slouží právě pseudo-prvek `:first-line`. Vyhoví mu vždy první řádek textu podle aktuálního formátování — pokud se stránka zformátuje jinak, změní se i obsah pseudo-prvku `:first-line`.

Příklad

Chceme zobrazit první řádky v odstavcích větším písmem s použitím kapitálek. V tabulce stylů proto použijeme pravidlo:

```
p:first-line { font-size: 150%; font-variant: small-caps }
```

Prohlížeč, který `:first-line` podporuje, zobrazí odlišné formátování pro první řádky všech odstavců. Pokud se změní rozměry zobrazovací plochy nebo velikost písma, změní se i první řádky textu a pseudo-prvky `:first-line` se upraví podle nich:



Obr. 19 a 20 — Formátování :first-line

3.6.5.8 Pseudo-prvek :first-letter

Obdobně lze samostatně zformátovat i první znak textu pomocí pseudo-prvku `:first-letter`. Tento pseudo-prvek akceptuje i *obtékání* pomocí vlastnosti `float` [4.3.5.5], čímž můžeme vytvářet i *zapuštěné iniciály.*

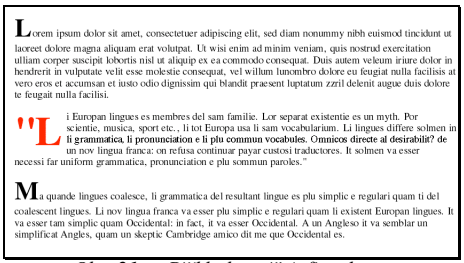
Prohlížeče by měly respektovat typografické zvyklosti. Pokud je proto prvním znakem nějaké interpunkční znaménko (např. uvozovka), měly by do `:first-letter` přidat i znak následující. Začíná-li tedy text např. `"Slovo`, měl by pseudo-prvek `:first-letter` obsahovat `"S`. Týká se to i slitků či grafických spřežek (např. písmeno „Ch“ v češtině).

Příklad

V odstavcích `<p class="p1">` zformátujeme první znak tučně, ve velikosti `250%`; v odstavcích `<p class="p2">` pak použijeme červenou *zapuštěnou iniciálu* ve větší velikosti:

```
p.p1:first-letter {font-size:250%;font-weight:bold}
p.p2:first-letter {
      float:left;
      margin-right:0.1ex;
      font-size:400%;
      font-weight:bold;
      color:red }
```

Odstavce by potom měly být zobrazeny takto:

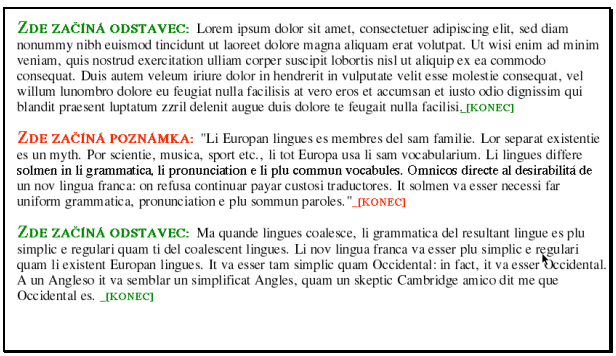


Obr. 21 — Příklad použití :first:letter

Pozn.: Pseudo-prvky `:first-line` a `:first-letter` lze použít pouze s *blokovými prvky* (viz dále), tedy např. pro `p`, `div`, `h1` atd. V prvcích *řádkových* (`a`, `span`, `strong`) nemají žádný význam. V obou pseudo-prvcích lze použít pouze vlastnosti písma, barev a pozadí a vlastnosti `text-decoration`, `text-transform`, `line-height`, `vertical-align`, `text-shadow` a `clear`. Ve `:first-line` pak navíc `word-spacing` a `letter-spacing`; ve `:first-letter` zase `margin`, `padding`, `border` a `float` (`vertical-align` lze použít, jen pokud je `float:none`).

3.6.5.9 Pseudo-prvky :before a :after

Tyto pseudo-prvky slouží ke vkládání nového obsahu, který se v dokumentu vůbec nevyskytuje. Pomocí `:before` se vkládají nová data před prvek, pomocí `:after` za prvek (obsah se přidává pomocí vlastnosti `content`, která je popsána dále). V ukázce je naznačeno, jak můžeme pomocí `:before` a `:after` přidat různý text před a za odstavce:



Obr. 22 — Příklad použití :before a :after

Podrobně je použití těchto pseudo-prvků vysvětleno v kapitole věnované generovanému obsahu [3.9].

3.6.6 Syntaxe selektorů

Jazyk CSS obecně nerozlišuje velká a malá písmena, je však třeba respektovat zvyklosti předepsané použitým jazykem dokumentu. Zatímco jazyk HTML není *citlivý na velikost* a lze v něm libovolně použít zápisy `<body>`, `<BODY>` i `<bOdY>`, pro jazyk XML platí pravý opak a uvedené značky představují různé prvky. V jazyce XHTML se zase musí značky i atributy povinně zapisovat malými písmeny. Všechna tato pravidla musíme respektovat i v tabulkách stylů — podle toho, jaký jazyk je použit v dokumentu.

Pro snazší přechod k jazyku XHTML je proto dobré zvyknout si zapisovat (nejen v dokumentech HTML, ale i v selektorech CSS) názvy značek, tříd i identifikátorů malými písmeny. V aplikacích XML to závisí jen na uvážení autora. Musí jen počítat s tím, že selektory `#prvek` a `#PRVEK` zde neznamenají totéž.

Je také důležité respektovat pravidla pro názvy tříd a identifikátorů. V CSS smějí obsahovat pouze písmena latinské abecedy (`a-z`, `A-Z`), číslice (`0-9`), spojovník („-“) a podtržítko („_“). Název musí začínat písmenem. Ačkoli jsou zde povoleny ještě znaky 161 a vyšší podle ISO 10646, v českém prostředí s nejednotným kódováním o nich nemá smysl příliš uvažovat. Je proto vhodné pojmenovávat třídy a id prvků jen názvy začínajícími písmenem, vše psát malými písmeny a z interpunkce používat pouze spojovník a podtržítko.

Pozn.: Zdaleka ne všechny možné selektory již můžeme v praxi použít. Některé jsou zcela běžné, jiné začínají být podporovány pouze sporadicky, další jsou v různých prohlížečích interpretovány rozdílně. Podrobnosti jsou uvedeny v přehledu kompatibility CSS na konci knihy.

3.7 Dědičnost a kaskáda CSS

Pomocí pravidel v tabulkách stylů definujeme vlastnosti prvkům stránky. Je ale zřejmé, že prvky mohou vyhovovat více pravidlům současně, ke stránce můžeme připojit i několik různých tabulek se styly, nebo také žádnou. Libovolný prvek tak může mít definovanou jednu vlastnost několika způsoby na více místech a vlastnosti jiné zase nebude mít definovány vůbec. Důležitou součástí CSS jsou proto pravidla a postupy pro korektní zpracovávání definic stylů.

3.7.1 Postup přiřazování hodnot vlastnostem

Během zpracovávání dokumentu klient vytváří *strom dokumentu* a každému prvku přitom přiřazuje hodnoty všech vlastností, které lze na daném médiu (obrazovka prohlížeče, tiskárna, zvukový výstup atd.) právě použít. Tyto hodnoty přitom postupně procházejí několika fázemi.

3.7.1.1 Definované hodnoty

V prvním kroku klient (prohlížeč) nejprve přiřadí všem vlastnostem jejich **definované hodnoty,** a to podle následujícího postupu:

1. **Je hodnota někde určena?** Nejprve se zpracují všechny známé definice stylů v dokumentu pomocí mechanismu *kaskády* (viz dále) a pokud *kaskáda* pro danou vlastnost vrátí nějakou hodnotu, použije se.
2. **Není-li určena, může se zdědit?** Pokud *kaskádou* není vrácena žádná hodnota a daná vlastnost je *dědičná,* použije se hodnota z rodičovského prvku. Obvykle se používá *vypočítaná hodnota* (viz dále).
3. **Použij výchozí hodnotu.** Pokud hodnotu nelze ani zdědit (tj. nemá-li již prvek žádného rodiče, nebo se vlastnost *nedědí,*), použije se *výchozí hodnota* definovaná pro tuto vlastnost.

3.7.1.2 Výchozí hodnota

Každá vlastnost má určenu jednu hodnotu, která se používá, pokud danému prvku není během procesu zpracování dokumentu přiřazena hodnota jiná. Tato hodnota se nazývá **výchozí** a každá vlastnost ji má dánu specifikací CSS (viz popis jednotlivých vlastností dále). U některých vlastností ponechává CSS přesnou *výchozí hodnotu* na použitém prohlížeči a ta pak v každém z nich může být odlišná.

3.7.1.3 Vypočítané hodnoty

Nyní již mají všechny prvky všem vlastnostem přiřazeny *definované hodnoty.* V dalším kroku se tyto hodnoty upřesňují. Hodnoty zadané *absolutně* (`1pt`, `12cm` atd.) obvykle žádnou úpravu nepotřebují, ale v ostatních případech, jako jsou hodnoty *relativní* (`60%`, `1.5em`) nebo hodnoty

určené *klíčovými slovy* (`x-large`, `MenuText`), je potřeba spočítat jejich skutečnou, absolutní hodnotu — např. výpočtem procentní části (podle toho, k čemu se procenta vztahují) či nahrazením klíčového slova za odpovídající hodnotu (`MenuText` = `rgb(0,0,0)`). Takto převedenou hodnotu nazýváme **vypočítaná hodnota.**

Ve většině případů se u dědičných vlastností dědí právě tato *vypočítaná hodnota.* Např. velikost písma (vlastnost `font-size`) je dědičná a má-li nějaký prvek definováno třeba `font-size:120%`, prohlížeč z toho *vypočítá* hodnotu řekněme `13.2pt`. Všechny prvky, které jsou *potomky* tohoto prvku a nemají vlastnost `font-size` definovánu, zdědí právě tuto *vypočítanou hodnotu* (tedy nikoli původně zadaných `120%`, ale již přepočítaných `13.2pt`). Pouze výjimečně (např. u vlastnosti `line-height`) se nedědí vypočítaná, ale *definovaná hodnota.* V takovém případě je to v popisu vlastnosti výslovně uvedeno.

3.7.1.4 Skutečné hodnoty

Vypočítané hodnoty by již prohlížeč mohl použít pro zformátování dokumentu. V praxi však ještě některé hodnoty upravuje podle možností svých či výstupního zařízení. Např. na monitoru je klient schopen zobrazovat čáry s rozměry pouze v násobcích bodu obrazovky a proto všechny hodnoty v `px` zaokrouhlí na celá čísla; tiskárna dokáže zobrazit pouze omezené barevné spektrum, vypočítané barvy jsou tedy přizpůsobeny možnostem tiskárny atd. Teprve takto upravené a přizpůsobené hodnoty klient ve skutečnosti použije. Nijak s nimi ale dále nepracuje. I když se tedy vypočítaná šířka rámečku `3.379px` třeba zaokrouhlí na `3px` a v tomto rozměru se vykreslí, prvek má stále přiřazenu původní *vypočítanou hodnotu*. Má-li nějaký jeho potomek zdědit šířku rámečku, zdědí hodnotu `3.379px`.

3.7.2 Dědičnost

Jak již bylo řečeno, některé vlastnosti se mohou dědit z *rodičů* na jejich *potomky.* U každé vlastnosti je přitom výslovně uvedeno, zda **je dědičná, či nikoli.** Pokud dědičná je a prvku tuto vlastnost nedefinujeme, hodnotu *zdědí:* použije se hodnota jeho rodičovského prvku (typicky *hodnota vypočítaná,* viz výše). Pokud ale vlastnost není dědičná, nebo je prvek *kořenem dokumentu*, použije se *hodnota výchozí* (v případě dokumentů HTML lze za *kořen dokumentu* považovat prvky `html` i `body`).

Pokud dokumentu nepřipojíme žádné styly, v kořenovém prvku se všem vlastnostem přiřadí jejich *výchozí hodnoty.* Všechny ostatní prvky tyto hodnoty zdědí (u dědičných vlastností), nebo použijí rovněž výchozí hodnotu (u vlastností, které dědičné nejsou).

Příklad 1:

```
h1 { color:red }
...
<h1>Toto je <em>důležitý</em> nadpis</h1>
```

Prvky `h1` mají definovánu červenou barvu textu, prvky `em` nemají vlastnost `color` určenu. Tato vlastnost je dědičná a `em` je zde potomek prvku `h1`, hodnotu vlastnosti `color` proto zdědí. Celý text bude zobrazen v červené barvě.

Příklad 2:

```
body { color: white; border: 1px solid black }
```

Vlastnost `color` je dědičná. Všechny prvky na stránce, kterým neurčíme jinou hodnotu, zdědí hodnotu z `<body>` a použijí bílou barvu textu. Vlastnost `border` dědičná není, černý tenký rámeček se proto zobrazí jen kolem celého dokumentu. Všechny ostatní prvky použijí výchozí hodnotu, kterou je `none` (žádný rámeček).

3.7.2.1 Hodnota inherit

Každá vlastnost může mít hodnotu `inherit`. Ta říká, že vlastnost pro tento prvek má mít **stejnou vypočítanou hodnotu** jako jeho rodičovský prvek. Jde tedy o jakési vynucené dědění vlastností, hodnota se zdědí i když vlastnost jinak dědičná není.

Příklad:

```
div.priklad { padding-left: 10% }
p { padding-left: inherit }
...
<div class="priklad">
      <p>Text text ... text</p>
      <p>Text text ... text</p>
      <p>Text text ... text</p>
</div>
```

Vlastnost `padding-left` (levý okraj) není dědičná, na *potomky* se proto nevztahuje. Pokud by odstavce `p` v tomto příkladu neměly vlastnost `padding-left` definovánu, byly by zobrazeny bez okraje (`padding-left` má výchozí hodnotu `0`). Hodnotou `inherit` zde ale říkáme, že prvky `p` **mají zdědit vypočítanou hodnotu z rodičovského prvku.** Tím je zde prvek `div.priklad`, jehož *vypočítanou hodnotou* levého okraje bude např. `32.8px` (hodnota spočtená z relativního zadání `10%`). Všechny odstavce `p` uvnitř tohoto prvku tedy budou mít stejnou vypočítanou hodnotu levého okraje: `32.8px`.

Kdybychom pro `p` použili stejnou **definovanou hodnotu** jako pro vnější `div` a zapsali tedy `p { padding-left: 10% }`, okraje už stejné nebudou. Procentní hodnota se u vlastnosti `padding-left` vztahuje k šířce prvku. Protože vnější prvek `div` a prvky `p` uvnitř něj jsou jinak široké, budou se lišit i hodnoty vypočítané ze zadaných `10%`. Vnější `div` bude mít vypočítaný levý okraj např. oněch `32.8px`, zatímco vnitřní odstavce už třeba jen `29.5px`.

3.7.3 Kaskáda

Kaskáda je jednou z klíčových funkcí CSS (připomeňme, že CSS jsou Tabulky *kaskádových* stylů). Popisuje postup, kterým se prohlížeč musí řídit při zpracování všech existujících definic stylů platných pro zobrazovaný dokument a pro použité *médium.* Na základě tohoto postupu seřadí všechny existující tabulky stylů podle jejich **váhy,** zohlední použité **příkazy !important,** zařadí **importované tabulky** na příslušné místo a nakonec setřídí shodné definice podle jejich **specifičnosti** a podle **pořadí.** Cílem tohoto postupu je zjištění, zda má daný prvek pro konkrétní vlastnost definovánu nějakou hodnotu. A pokud takových hodnot existuje více, pak také výběr té správné.

3.7.3.1 Váhy tabulek stylů

V klientu (prohlížeči) se při formátování dokumentu scházejí tabulky stylů ze tří různých zdrojů:

- **od autora** — tabulky, které vytváří autor stránky a připojuje je k dokumentu
- **od uživatele** — prohlížeč může umožnit, aby uživatel používal soubory s vlastními tabulkami stylů, případně program poskytuje nějaké rozhraní s nastavením stylu, které takové tabulky vygeneruje (nebo se chová, jako by je vygeneroval)
- **od klienta** — každý klient musí povinně používat vlastní *výchozí tabulku stylů* (nebo se alespoň chovat, jako by existovala). Ta popisuje typické formátování dokumentů daného jazyka. Vlastně tím definuje prezentaci dokumentů, které nepoužívají vlastní styly. V příloze na konci knihy najdete ukázku takové výchozí tabulky stylů pro jazyk HTML 4.0.

Kaskáda CSS přiřazuje každé tabulce příslušnou **váhu,** na níž potom závisí *priorita* definic stylů. **Tabulky autora mají větší váhu než tabulky uživatelské;** obě pak mají přednost před tabulkami klienta. Pro pravidla s příkazy `!important` ale platí pořadí jiné.

3.7.3.2 Příkaz !important

U libovolné definice vlastnosti je možné uvést příkaz `!important` (angl. „důležitý“), který jí **dává přednost před všemi ostatními definicemi** bez tohoto příkazu.

Protože styly definované autorem stránek mají obecně větší váhu než styly uživatelské, CSS tímto dává jistou možnost uživateli, aby si některé (pro něj důležité) vlastnosti nastavil podle svého přání či potřeby. Zatímco v CSS1 ještě měly autorovy definice s `!important` přednost před `!important` definicemi uživatele, v CSS2 už byl tento nedostatek odstraněn. Příkaz `!important` mění *váhu* definic a dává tak uživateli jisté „právo veta“. Při použití `!important` tedy platí opačné pořadí, **nejvyšší váhu mají definice uživatele** a teprve pak se případně mohou použít definice autorovy. Tabulky klienta mají vždy váhu nejnižší a příkaz `!important` se v nich nepoužívá.

Je to důležité především pro uživatele s handicapem — ať již zdravotním či technickým. Pokud má např. problém s barevným viděním a dává přednost černému textu na bílém pozadí, může si ve svém prohlížeči vytvořit uživatelskou tabulku stylů, do níž napíše pravidla:

```
body { background:white !important; color:black !important }
```

I kdyby autor v dokumentu pro `body` definoval jinou barvu pozadí a textu, uživatelovy definice budou mít vyšší váhu a „přebijí“ deklarace ve stránce. A to i kdyby autor také použil `!important` — v CSS2 má uživatel přednost.

Příkaz `!important` může mít význam i pro autory. Ačkoli s CSS1 se ještě nedoporučovalo jej příliš používat (právě proto, že tím autoři příliš omezovali uživatele), díky elegantnímu řešení v CSS2 se opět vrací do hry. Dobře navržené tabulky stylů se bez něj sice většinou obejdou, ale výjimečně může nastat situace, kdy v rámci *kaskády* již nemáme dostatek prostředků, jak prosadit určité vlastnosti potřebnou hodnotu a jiná pravidla mají vždy vyšší prioritu (viz *pořadí kaskády* dále). Pomocí příkazu `!important` pak můžeme tento problém snadno vyřešit. Jedná se však vždy o nestandardní situace a složité konstrukce — při běžném použití by se měl každý autor bez tohoto příkazu obejít.

3.7.3.3 Pravidlo @import

Toto pravidlo už bylo několikrát zmíněno dříve. Pomocí **@import** můžeme připojit k tabulce stylů jiné tabulky. Definice z importovaných tabulek mají menší prioritu než definice zapsané přímo v tabulce, které je mohou „přebít“. To je patrně jeden z důvodů, proč se pravidla *@import* musí uvádět vždy na začátku tabulky, dříve než napíšeme jakékoli jiné pravidlo. Některé prohlížeče to sice nerespektují (viz tabulku kompatibility na konci knihy), což ale nic nemění na tom, že my jako autoři bychom tento požadavek dodržovat měli. Když už ne kvůli respektování normy CSS, tak určitě proto, abychom nezapomněli na zmíněnou prioritu pravidel. Pro jednoduchost si můžeme představovat, že na místo pravidla *@import* se vloží celý obsah importované tabulky.

Pravidlo @import se zapisuje ve tvaru `@import URL [seznam_médií];` (středník na konci je nezbytný). `URL` je adresa (URI) souboru CSS s importovanou tabulkou stylů a seznam médií je volitelný. Možné jsou dvě syntaxe zápisu adresy: `<uri>` podle zvyklostí CSS [3.3.5], tedy ve tvaru `url(adresa)`, `url('adresa')` či `url("adresa")`, nebo pouze uvedením adresy v uvozovkách. Následující příklady jsou tedy shodné a oba zajistí načtení tabulky `"mujstyl.css"` na začátek aktuální tabulky stylů:

```
@import url('mujstyl.css');
@import "mujstyl.css";
```

Za URL souboru s tabulkou stylů můžeme volitelně doplnit seznam *médií* [3.4] oddělených čárkou, na něž se mají styly z tabulky vztahovat — např.:

```
@import url(mujstyl.css) screen, handheld;
@import 'mujstyl_tisk.css' print;
```

Pozn.: Faktu, že některé prohlížeče nepodporují obě možné syntaxe URL (podporují pouze jednu z nich, či dokonce @*import* „neznají“ vůbec), se často používá pro jejich „odstřižení“ od stylů — viz [5.1.4].

3.7.3.4 Pořadí kaskády

Pro stanovení *priority* definic z tabulek stylů se používá přesný postup, který klient musí korektně dodržet. Je dobré jej pochopit a důkladně si jej zažít. **Kaskáda CSS** je mohutný nástroj, pomocí nějž můžeme své tabulky stylů velmi zefektivnit. A na druhé straně si také můžeme způsobit nepříjemné problémy, pokud na tato pravidla zapomeneme — a budeme pak dlouze hledat, kde jsme udělali chybu a proč prohlížeč zobrazuje stránku úplně jinak, než jsme zamýšleli.

Postup při zpracování pravidel stylů je následující:

1. Najdou se všechny definice pro daný prvek, platné pro použité médium (obrazovka, tiskárna atd.). Definice se na prvek vztahují, pokud vyhovuje použitému *selektoru*.
2. Tyto definice se setřídí **podle váhy** jejich původu. Uvedené pořadí je od nejnižší váhy po nejvyšší:
    a. pravidla z tabulek klientu
    b. pravidla z tabulek uživatele
    c. pravidla z tabulek autora (včetně *přímých stylů*)
    d. pravidla s !important z tabulek autora
    e. pravidla s !important z tabulek uživatele
3. Druhé setřídění je **podle specifičnosti** (viz dále). Mají-li dvě v definice stejnou *váhu,* porovnají se jejich selektory. Specifičtější mají přednost před selektory obecnějšími. Pseudo-třídy se zde považují za běžné třídy.
4. Poslední setřídění je **podle pořadí definování.** Mají-li dvě definice stejnou *váhu původu* i stejnou *specifičnost,* přednost má ta, která byla definována později. Importované styly se považují za definované dříve než styly uvedené přímo v tabulce.

3.7.3.5 Výpočet specifičnosti selektorů

Ve třetím kroku kaskády se v případě shody třídí definice podle **specifičnost selektorů.** Pomocí přesného algoritmu se určuje, jak podrobně jsou v selektoru popsány prvky, na něž se má pravidlo vztahovat. Nejobecnější selektory (např. jen jméno značky HTML) mají nejnižší prioritu. Čím podrobnější (specifičtější) popis v selektoru je, tím stoupá i jeho priorita. Algoritmus pro výpočet specifičnosti je následující:

- a = 1, pokud definice pochází z *přímého stylu* (z atributu `style` přímo ve značce). Pokud pochází z tabulky stylů, je a = 0.

- b = počet id v selektoru (`#jmeno`)
- c = počet atributů (včetně tříd) a pseudo-tříd v selektoru
- d = počet názvů značek v selektoru (`p`, `body`, `h1` atd.)
- pseudo-prvky se ignorují

Specifičnost se určí spojením (ne součtem) těchto čtyř čísel a-b-c-d. Např. pro a=0, b=1, c=3 a d=4 je specifičnost 0134 = 134.

Příklad výpočtu specifičnosti

```
*{...}              /* a=0 b=0 c=0 d=0 -> specifičnost = 0000 = 0 */
p {...}             /* a=0 b=0 c=0 d=1 -> specifičnost = 0001 = 1 */
div p {...}         /* a=0 b=0 c=0 d=2 -> specifičnost = 0002 = 2 */
div ul li+li {...}  /* a=0 b=0 c=0 d=4 -> specifičnost = 0004 = 4 */
*[id=xyz123]        /* a=0 b=0 c=1 d=1 -> specifičnost = 0010 = 10 */
h1 > *[title]{...}  /* a=0 b=0 c=1 d=1 -> specifičnost = 0011 = 11 */
div p em.info {...} /* a=0 b=0 c=1 d=3 -> specifičnost = 0013 = 13 */
li.red.level {...}  /* a=0 b=0 c=2 d=1 -> specifičnost = 0021 = 21 */
#xyz123 {...}       /* a=0 b=1 c=0 d=0 -> specifičnost = 0100 = 100 */
body #xyz123 {...}  /* a=0 b=1 c=0 d=1 -> specifičnost = 0101 = 101 */
#abc> #xyz {...}    /* a=0 b=2 c=0 d=0 -> specifičnost = 0200 = 200 */
style="..."         /* a=1 b=0 c=0 d=0 -> specifičnost = 1000 = 1000 */
```

Při výpočtu specifičnosti se nehledí na význam selektorů, ale pouze na formu jejich zápisu. Ačkoli selektory `*[id=jmeno]` a `#jmeno` vyjadřují totéž, z hlediska kaskády se v prvním případě jedná o selektor s atributem (spec=0010) a v druhém případě o selektor s ID (spec=0100). Zápis `#jmeno` tak má vždycky vyšší prioritu. *Přímé styly* (definované ve značkách HTML pomocí atributu `style`) mají pokaždé prioritu nejvyšší (spec=1000).

3.7.3.6 Použití ne-kaskádových stylů

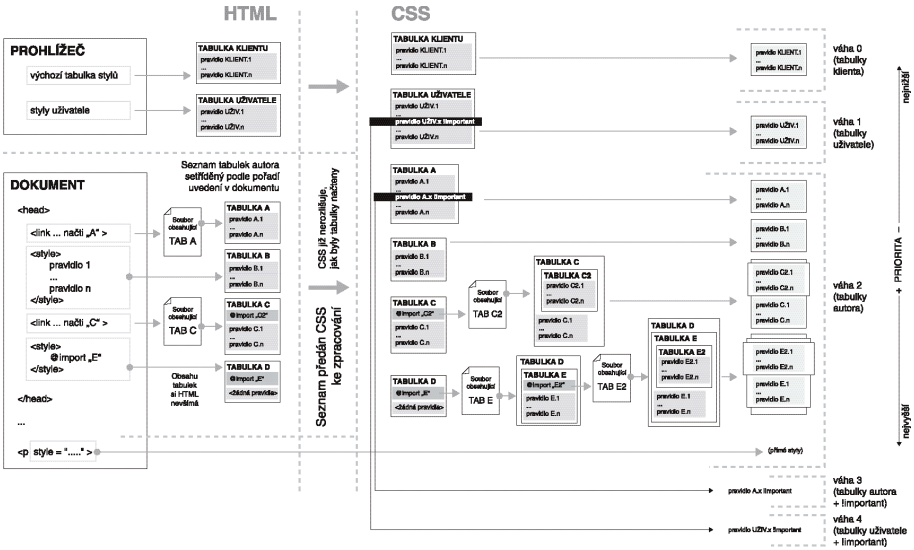
Prohlížeč může používat i další definice vzhledu prvků, které nepocházejí z CSS. Jedná se především o *zastaralé* značky HTML, jako `font`, `big`, `small` apod. V rámci *přechodové fáze* HTML (dokumenty typu *Transitional*) lze takto kombinovat oba způsoby formátování, aby se autorům usnadnil přechod k novým standardům. V tomto případě ale musí prohlížeč takové „styly“ nejprve skrytě převést do CSS definic a přiřadit jim stejnou váhu, jako má *výchozí tabulka klienta.* Díky tomu je možné formátování těchto prvků v autorských i uživatelských stylech změnit.

Pozn.: V CSS 1 ještě měly tyto „styly“ váhu a prioritu vyšší, v CSS 2 byla poněkud snížena a v CSS 2.1 klesla až na tuto úroveň, která nejvíce odpovídá skutečné situaci v prohlížečích.

3.7.3.7 Shrnutí procesu přiřazování hodnot v CSS

Na prvním obrázku je znázorněno, jak se o zpracování stylů dělí nástroje zpracovávající dokument (jazyk HTML) a nástroje kaskádových stylů (jazyk CSS). HTML najde všechny

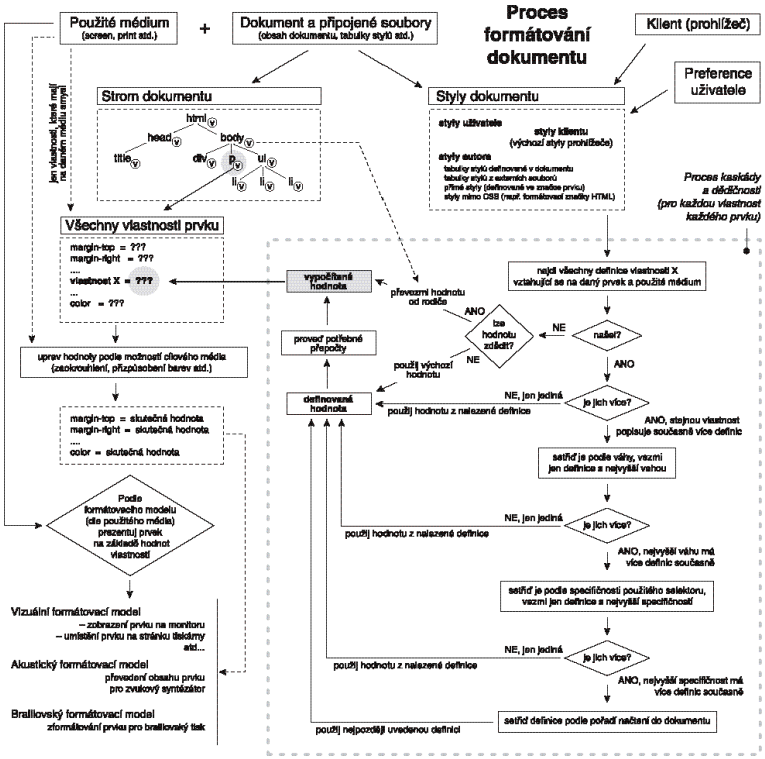
tabulky stylů přiřazené k dokumentu a předá je ke zpracování CSS v pořadí, v němž jsou v dokumentu uvedeny. O jejich obsah se přitom nestará, ani nenačítá importované tabulky. CSS se naopak nestará, jak byly styly k dokumentu připojeny (zda byly v externím souboru či zapsané přímo v dokumentu), respektuje pouze pořadí, v němž tabulky obdržel. Teprve nyní se načtou případné importované tabulky a pravidla se seřadí podle pravidel kaskády.



Obr. 23 — Proces formátování dokumentu

—> Využívání různých způsobů připojování stylů [5.1.4]

Následující schéma zobrazuje postupy, pomocí nichž se přiřadí hodnoty všem vlastnostem každého prvku ve stromu dokumentu a ty se následně použijí pro prezentaci prvku.



Obr. 24 — Proces kasády

—> Výhodné používání pravidel kaskády a dědičnosti [5.1.8]

3.8 Vizuální formátovací model CSS

Dokumenty jsou zpracovávány klienty různým způsobem podle použitého média. Jedním z hlavních rozdělení těchto médií je na *vizuální, zvuková* a *dotyková* [3.4.2]. Druhé dva typy se využívají spíše výjimečně na specifických výstupních zařízeních, nejčastěji jsou používána **média vizuální.**

Vizuální formátovací model popisuje postup, kterým klient zpracovává *strom dokumentu* na *vizuálních médiích.* Tato metodika je jednou z nejdůležitějších součástí technologie CSS.

Připomeňme, že již během vytváření *stromu dokumentu* klient přiřadí hodnoty všem vlastnostem všech prvků podle *pravidel kaskády* [3.7.3]. Nyní má tedy k dispozici všechny prvky, které se mají na stránce prezentovat, uspořádané ve *stromu dokumentu* a rovněž všechny vlastnosti popisující jejich podobu. Dalším krokem je jejich zobrazení, které klient provádí podle postupů určených formátovacím modelem.

3.8.1 Zobrazování dokumentu

Dokument se vždy vykresluje na nějaké **zobrazovací ploše.** U *stránkovaných médií* (tiskárna, projektor) je touto plochou sada jednotlivých stránek, mezi něž je obsah dokumentu rozdělen. S každou stránkou se pak pracuje jako s celkem, samostatnou plochou, která je celá k dispozici pro zobrazení dokumentu.

U *plynulých médií* (obrazovka, okno prohlížeče), je oproti tomu k dispozici jediná *zobrazovací plocha* neomezené délky a šířky. Klient potom uživateli poskytuje **průzor** na tuto plochu, jakési okno, skrze nějž uživatel vidí alespoň část dokumentu. Pokud v *průzoru* není vidět dokument celý, musí dát klient k dispozici nějaký posuvný mechanismus (např. posuvníky v okně prohlížeče), pomocí nějž lze *průzor* nad dokumentem přesouvat.

3.8.2 Zobrazování prvků

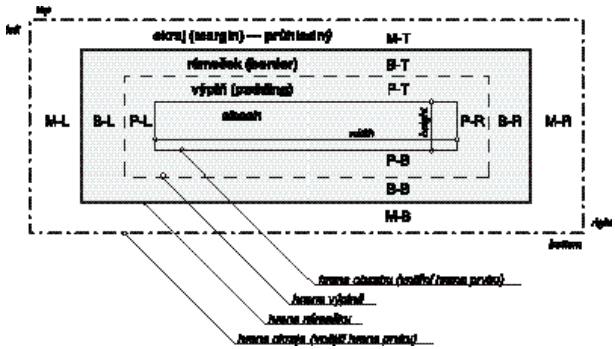
Ve *vizuálním formátovacím modelu* se pro každý prvek stromu dokumentů generuje **rám** *(box)* (případně i více *rámů,* nebo naopak žádný), podle *rámového modelu CSS* [3.8.3 a dále]. Jejich vzhled a umístění na *zobrazovací ploše* přitom závisí na **rozměrech rámu** [3.8.3], použitém **pozičním schématu** (normální, plovoucí, absolutní) [3.8.6], na vztazích mezi prvky ve **stromu dokumentu** [3.5] a externích informacích (rozměry *průzoru* do dokumentu, *skutečné rozměry* objektů atd.)

Různým kombinacím těchto údajů odpovídá určitý **typ formátování,** které CSS jmenovitě popisuje.

3.8.3 Rámový model, rozměry rámů

Rámový model CSS popisuje **obdélníkové rámy,** které klient generuje pro prvky dokumentu. Základem každého *rámu* je **oblast obsahu** (text, obrázek atd.). Kolem ní mohou být volitelně další oblasti: **výplňová oblast** *(padding),* **oblast rámečku** *(border)* a **oblast okraje** *(margin).* Jejich rozměry popisují příslušné vlastnosti CSS a také odpovídající *typ formátování.*

Pro rozměry všech oblastí kolem obsahu (výplň, rámeček, okraj) existují samostatné vlastnosti, takže jejich rozměr je možné určit na každé straně jiný. Na následujícím obrázku jsou zobrazeny všechny oblasti rámu vygenerovaného pro prvek.



Obr. 25 — Rám prvku, jeho oblasti a rozměry

Na obrázku jsou symbolicky označeny vlastnosti, které CSS používá pro definování rozměrů těchto oblastí. Pro každou oblast — P (padding), B (border), M (margin) — existují vždy čtyři vlastnosti pro rozměr v každém směru — T (top, horní), R (right, pravý), B (bottom, dolní) a L (left, levý). Např. M-L značí vlastnost `margin-left` (velikost levého okraje). Šířka a výška obsahu tvoří **vnitřní rozměry** prvku. Tyto rozměry také závisí na dalších faktorech — zda a jak jsou definovány vlastnosti `width` a `height`, zda prvek obsahuje text či jiný obsah, zda je prvek tabulkou atd. Pravidla pro výpočet rozměrů rámů jsou popsána dále [3.8.7], stejně jako příslušné vlastnosti.

—> Vlastnosti width a height [4.3.4]
—> Vlastnosti margin [4.3.1], padding [4.3.2], border [4.3.3]

Rozměry ostatních oblastí se k vnitřním rozměrům připočítávají a součet tvoří **vnější rozměry** prvku:

vnější šířka = M-L + B-L + P-L + width + P-R + B-R + M-R
vnější výška = M-T + B-T + P-T + height + P-B + B-B + M-B

Hrany jednotlivých oblastí jsou často zmíněny v dalším textu. **Hrana okraje** tvoří vnější hranu prvku. Zde začíná pole jeho působnosti, vzdálenost této hrany se měří při *pozicování* prvků. **Hrana rámečku** tvoří viditelnou plochu prvku (oblast okraje je vždy průhledná), až po tuto hranu se zobrazuje pozadí prvku. **Hrana výplně** omezuje odstup rámečku od obsahu, na této hraně začíná rámeček. **Hrana obsahu** určuje rozměry prvku, udává jeho šířku a výšku, na této hraně již přímo leží obsah prvku. Pokud je některý z rozměrů některé oblasti nulový, odpovídající hrany sousedních oblastí splynou.

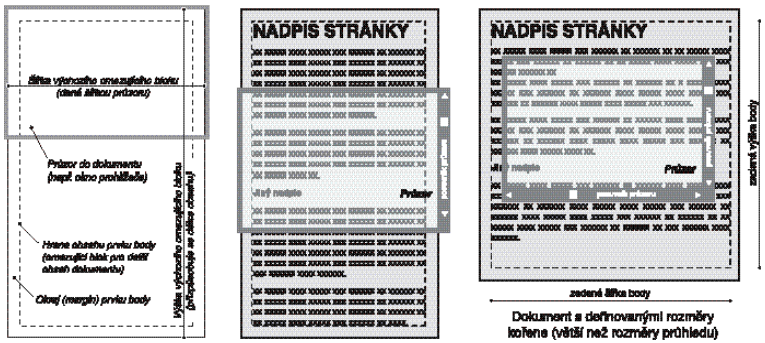
Pokud mají okraje **zápornou hodnotu,** *hrana okraje* se posune až dovnitř oblasti rámečku, výplně či obsahu. Ostatní oblasti tak přesahují *vnější hranu* prvku. Umístění prvku se však stále řídí touto *vnější hranou* — záporné okraje tak způsobí, že obsah prvku může přesahovat svůj *omezující blok* (viz též *přetékání* [3.8.8.1]). Prohlížeče je ale nemusejí podporovat jednotně.

3.8.4 Omezující bloky

Hrana obsahu vytváří tzv. **omezující blok** pro všechny potomky prvku, závisí na něm jejich rozměry i umístění. Každý prvek tak respektuje rozměry a umístění svého *omezujícího bloku* a současně sám definuje *omezující blok* svým potomkům.

Kořen dokumentu určuje **výchozí omezující blok** pro celý dokument. Jeho rozměry můžeme určit definováním vlastností `width` a `height` prvku tvořícího *kořen dokumentu* (v HTML je to prvek `<html>` i `<body>`). Pokud je sami neurčíme, použijí se výchozí rozměry. U *stránkovaných médií* určuje výchozí rozměry formát výstupního média (např. rozměry papíru).

V případě *médií plynulých* závisí výchozí rozměry *výchozího omezujícího bloku* na prohlížeči. Většinou se jako výchozí **šířka** pro dokument používá aktuální šířka *průzoru* [3.8.1]. Není-li určena **výška,** *omezující blok* se bude neomezeně natahovat na výšku, aby pojal celý obsah dokumentu. V praxi to tedy znamená, že pokud pomocí CSS sami nezadáme rozměry prvků `html` nebo `body`, bude výchozí šířka dokumentu nastavena podle rozměrů okna prohlížeče a výška se přizpůsobí délce obsahu. Pokud se rozměr okna prohlížeče změní, může se změnit i šířka *výchozího omezujícího bloku* a celý dokument tak se přeformátuje. Pozor, jakkoli je to typické chování většiny prohlížečů, není to nezbytně chování povinné.



Obr. 26 — Dokumenty zobrazené na plynulých médiích

Pozn.: *Absolutně pozicované* prvky se z běžného formátování vyjímají a zobrazují se samostatně. Jejich omezující blok určuje nejbližší absolutně pozicovaný *předek*. Pokud žádný z jeho předků pozicovaný není, omezujícím blokem je *výchozí omezující blok* dokumentu. Pozor, u absolutně pozicovaných prvků netvoří omezující blok *hrana obsahu,* ale *hrana výplně!*

—> Absolutní pozicování [3.8.6.6]

3.8.5 Generování rámů

Uvnitř *omezujícího bloku* jsou vytvářeny rámy prvků (za určitých okolností jej mohou i přesahovat [3.8.8.1]).Ve vizuálním formátovacím modelu jsou rámy dvojího typu: **blokové** a **řádkové.** Typ použitého rámu závisí především na hodnotě vlastnosti `display` [4.3.5.1].

3.8.5.1 Blokové prvky a blokové rámy

Blokové prvky jsou takové prvky, které se zobrazují jako souvislé obdélníkové bloky (např. odstavce). Prvek je blokový, pokud je hodnota jeho vlastnosti `display` např. `block`, `list-item`, `table` (částečně i `run-in`). Blokové prvky generují **blokové rámy.**

Blokový prvek generuje jeden *hlavní rám,* který tvoří *omezující blok* pro jeho obsah. *Hlavní rám* může obsahovat jen rámy stejného typu — jen *blokové,* nebo jen *řádkové.* Některé typy bloků mohou generovat i další dodatečné rámy (např. `list-item`), jejich pozice je vztažena k *hlavnímu rámu* [3.9.2].

3.8.5.2 Anonymní blokové rámy

Pokud je obsah blokového prvku smíšený, tj. jsou v něm jak bloky, tak řádkový obsah, sjednotí se tak, aby byl obsah tvořen pouze bloky. Z každého úseku řádkového obsahu se proto vytvoří **anonymní blok**.

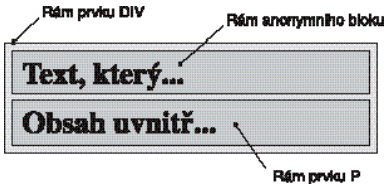
Příklad:

```
<div>
      Text, který není blokovým prvkem
      <p>Obsah uvnitř blokového prvku</p>
</div>
```

Protože obsah bloku `div` musí být stejného typu, řádkový obsah se virtuálně uzavře do *anonymního bloku,* tedy jako by konstrukce byla takováto:

```
<div>
      <blok>Text, který není blokovým prvkem</blok>
      <p>Obsah uvnitř blokového prvku</p>
</div>
```

Tím se docílilo toho, že prvek `div` již obsahuje pouze bloky. *Anonymní bloky* dědí hodnoty vlastností od nadřazeného prvku. Vlastnosti, které nejsou *dědičné,* nabývají výchozích hodnot (tedy např. `font-size` se zdědí, ale `margin` či `padding` budou mít hodnotu `0`). Rámy pro prvek `div` tedy budou vytvořeny takto:



Obr. 27 — Anonymní blokové rámy

3.8.5.3 Řádkové prvky a řádkové rámy

Řádkové (inline) prvky jsou prvky, které nevytvářejí samostatné bloky, typicky se jedná o úseky textu (např. prvky `em` či `strong`). Jejich obsah se zobrazuje v řádcích. Řádkový typ prvku určuje několik hodnot vlastnosti `display` — např. `inline`, `inline-table` a částečně i `run-in`. Řádkové prvky generují **řádkové rámy.**

3.8.5.4 Anonymní řádkové rámy

V konstrukci např.:

```
<p>Toto je <em>zvýrazněný</em> text.</p>
```

blokový prvek `p` vytváří *blokový rám,* jehož obsahem je několik *řádkových rámů.* Rám pro úsek „zvýrazněný“ je vytvořen prvkem `em`, pro úseky textu před ním a za ním jsou rámy vytvořeny automaticky, jako *anonymní řádkové rámy.* Anonymní prvky stejně jako u bloků dědí vlastnosti od svého rodičovského prvku, nedědičné vlastnosti nabývají výchozích hodnot.

Obr. 28 — Anonymní řádkové rámy

3.8.5.5 „Zatahované“ (run-in) rámy

Prvky s vlastností `display:run-in` mají specifické chování, na rozhraní mezi blokovým a řádkovým. Takovýto prvek se zobrazuje v závislosti na následujícím obsahu:

- následuje-li za ním blok (který není plovoucí ani absolutně pozicovaný), prvek vytvoří první *řádkový rám* následujícího bloku
- v opačném případě se prvek chová jako blok

Typickým použitím je vytváření „zatahovaných“ nadpisů — např.:

```
h4 { display: run-in }
...
<h4>Toto je nadpis</h4>
<p>A zde pokračuje text v následujím odstavci, který je blokem...</p>
```

se zobrazí např. takto:

Toto je nadpis A zde pokračuje text v následujícím odstavci, který je blokem ...

Pozn.: I když takto „zatažený“ blok vytvoří řádkový rám, dědí stále vlastnosti v rámci *stromu dokumentu* a nikoli od prvku, do nějž byl „vsunut“.

3.8.6 Poziční schémata

Umístění (i způsob výpočtu rozměrů) vygenerovaných rámů závisí na použitém **pozičním schématu.** Každý prvek je má určeno samostatně pomocí vlastnosti `position` a případně je ovlivněno i vlastností `float`. V CSS se používají tři různá poziční schémata, každé z nich nakládá s prvky odlišným způsobem:

- **normální tok**; hodnoty `position:static` nebo `position:relative` a současně `float:none`
- **plovoucí prvky**; hodnoty `position:static` nebo `position:relative` a současně `float:left` nebo `float:right`
- **absolutní pozicování**; hodnoty `position:absolute` a `position:fixed`

—> Vlastnost position [4.3.5.2]

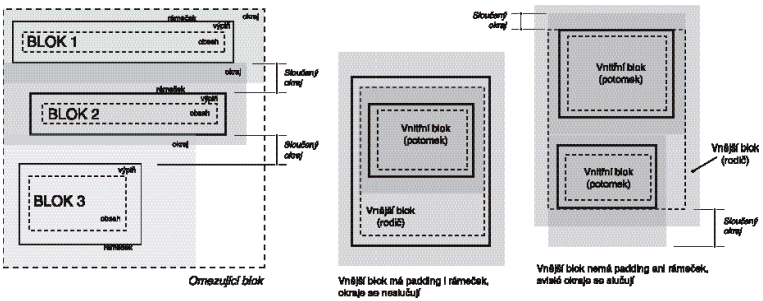
3.8.6.1 Normální tok dokumentu

Normální tok je výchozím zobrazovacím schématem pro dokumenty (prvky mají vesměs výchozí hodnotu `position:static`). Dokud autor neurčí jinak, všechny prvky jsou zobrazovány tímto způsobem.

3.8.6.2 Blokové rámy v normálním toku, slučování okrajů

Blokové rámy se umísťují **pod sebou,** od levé horní hrany jejich *omezujícího bloku.* Horní *vnější hrana* (*hrana okraje,* [3.8.3]) prvního rámu se dotýká horní hrany *omezujícího bloku,* svislé vzdálenosti mezi rámy se řídí velikostí *okrajů* (vlastnosti `margin-top` a `margin-bottom`). Platí přitom pravidlo, že **okraje sousedních bloků se slučují.** Bloky se tedy navzájem nedotýkají svými vnějšími hranami, ale mezera mezi nimi se spojí a odpovídá **většímu z obou sousedních okrajů.** Při záporných hodnotách se při slučování použijí jen existující kladné okraje. Pokud mají všechny dotčené okraje záporné hodnoty, sloučený okraj bude nulový.

Pokud prvek nemá horní, resp. dolní výplň, ani rámeček (nulový `padding` a `border`), obdobným způsobem se *slučuje* i jeho horní okraj s horním okrajem prvního blokového potomka, resp. dolní okraj s dolním okrajem posledního blokového potomka.



Obr. 29 — Blokové rámy v normálním toku a slučování okrajů

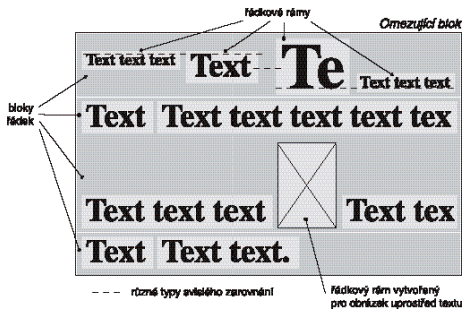
Levá hrana bloků se vždy dotýká levé hrany jejich omezujícího bloku. A to i v případě, že se zde vyskytuje nějaký *plovoucí prvek,* který tyto prvky *obtékají.* O šířku plovoucího prvku se zmenší pouze jejich obsah, vnější hrana i rámeček zůstávají na původním místě, jako by zde obtékaný prvek nebyl (viz ukázky v [3.8.6.5]).

Pozn.: V případě opačného směru textu (jazyky píšící zprava doleva, vlastnost `direction:rtl`) jsou strany prohozeny — bloky se dotýkají pravé hrany omezujícího bloku, text plyne zprava doleva atd. Nadále ale budeme uvažovat pouze u nás běžný zápis textu zleva doprava.

3.8.6.3 Řádkové rámy v normálním toku

Řádkové rámy se umísťují **vedle sebe** od horní hrany omezujícího bloku. Svisle mohou být navzájem zarovnány různým způsobem — ve stejné úrovni mohou být horní hrany textu, spodní hrany, střed atd. (to ovlivňuje jejich vlastnost `vertical-align`).

Obdélník, který obsahuje rámy tvořící jednu řádku, se nazývá **blok řádky.** Jeho šířka je rovná šířce *omezujícího bloku.* Jeho výška je přinejmenším taková, aby pojal všechny rámy, které jej tvoří (v závislosti na jejich výšce a vzájemném svislém zarovnání). Výška *bloku řádky* však může být i vyšší, její výpočet je uveden dále [3.8.7.11]. Pokud se všechny *řádkové rámy* nevejdou do jediného *bloku řádky,* jsou rozděleny do několika bloků řádek. Ty se pak řadí svisle přímo pod sebe (bez mezer). Odstavec textu je tvořen takovou posloupností *bloků řádek.*



Obr. 30 — Řádkové rámy v normálním toku

Bloky řádek se vždy dotýkají krajních hran *omezujícího bloku* a jejich šířka je v rámci celého bloku stejná. Pouze je-li dostupný prostor omezen přítomností nějakého *plovoucího prvku,* mohou se šířky jednotlivých bloků řádek lišit — dotčené řádky jsou zúženy na zbylý prostor vedle obtékaného prvku. Výšky jednotlivých řádek se mohou lišit kdykoli — např. obsahuje-li některá řádka vložený obrázek, který je vyšší než okolní text.

Pokud je celková šířka rámů na jedné řádce menší než šířka *omezujícího bloku,* jejich vodorovné rozmístění se řídí vlastností `text-align`. Má-li hodnotu `justify`, může klient roztáhnout i jednotlivé řádkové rámy, aby vyplnily celý prostor.

—> Výpočet výšky řádek [3.8.7.11]
—> Vlastnosti text-align [4.4.2.4] a vertical-align [4.4.2.6]

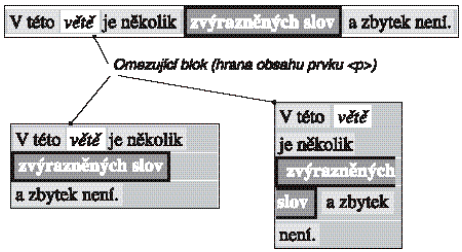
Jelikož řádkové rámy nemohou přesáhnout šířku řádky, delší rámy jsou podle potřeby rozděleny na několik dílčích rámů, které jsou distribuovány na samostatné řádky. U takto „rozříznutého“ rámu se v místě řezu nezobrazují jeho okrajové oblasti (výplň, rámeček a okraj). Ty jsou zobrazeny pouze na skutečných okrajích rámu (viz příklad).

Příklad:

Následující blok `p` obsahuje několik řádkových prvků:

```
<p>V této <em>větě</em> je několik <strong>zvýrazněných slov</strong> a
zbytek není.</p>
```

Bude zde vytvořeno pět *řádkových rámů* — dva pocházejí z (řádkových) prvků `em` a `strong`, ostatní jsou vytvořeny automaticky jako *anonymní rámy*. Pokud bude mít prvek `strong` definován v CSS nějaký styl pro rámeček, pozadí atd., budou vytvořeny řádkové rámy a jednotlivé řádky např. takto (podle aktuální šířky omezujícího bloku):



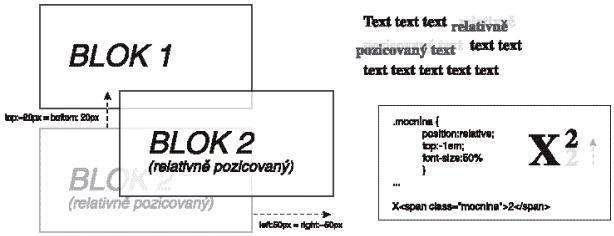
Obr. 31 — Řádkové rámy a jejich rozdělování na více řádek

3.8.6.4 Relativní pozicování

Prvek v normálním toku (či prvek *plovoucí*), který má vlastnost `position:relative`, může být posunut z své pozice. Velikost a směr posunu určují vlastnosti `left`, `right`, `top` a `bottom`. Vlastnost `left` určuje posun vpravo, `right` posun vlevo; vlastnost `top` posouvá prvek dolů, `bottom` nahoru. Platí přitom, že `left=-right` a `top=-bottom`. Pokud zadáme obě protilehlé vlastnosti a nebudou odpovídat této rovnosti, jedna z nich bude ignorována.

—> Vlastnosti top, right, bottom a left [4.3.5.3]

Prvek, který je **relativně pozicován,** je nejprve zformátován v dokumentu jako by pozicován nebyl a teprve nakonec je posunut ze své původní pozice o zadanou vzdálenost. Znamená to tedy, že relativní pozicování nemá žádný vliv na vzhled a rozměry prvku. Posun také nijak neovlivňuje formátování okolních prvků — takto pozicované prvky mohou způsobit, že se rámy prvků budou překrývat.



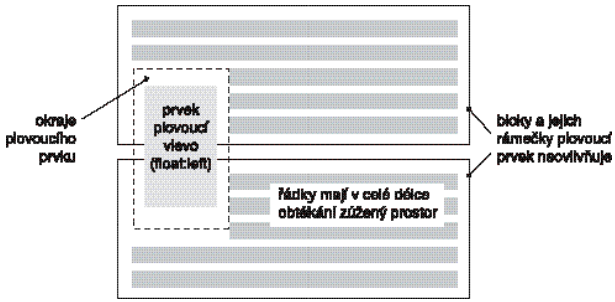
Obr. 32 — Ukázky relativního pozicování

3.8.6.5 Plovoucí prvky

Prvky v normálním toku se stávají **plovoucími,** pokud mají vlastnost `float:left`, resp. `float:right` (u absolutně pozicovaných prvků se tato vlastnost ignoruje). Prvek je z místa, kde by byl normálně zobrazen, posunut k levé (resp. pravé) hraně omezujícího bloku, z toku dokumentu je vyňat a ostatní obsah jej *obtéká.* To znamená, že (pokud zbývá vedle odsunutého *plovoucího prvku* ještě nějaké místo) následující prvky jsou zobrazeny ve zbývajícím prostoru podél něj, prostor pro jejich obsah je zúžen a v plné šíři jsou zobrazeny zase až za tímto plovoucím prvkem. Pokud má však následující obsah nastavenu příslušnou hodnotu vlastnosti `clear`, plovoucí prvek obtékat nebude a zobrazí se až pod ním.

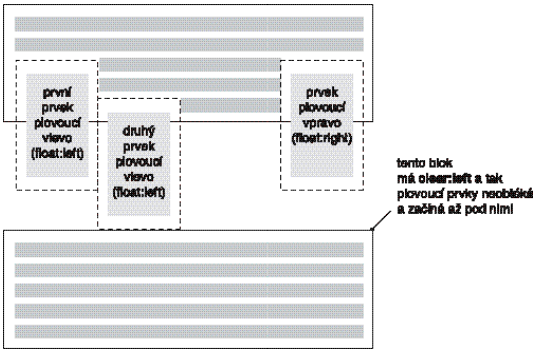
—> Vlastnosti float [4.3.5.5] a clear [4.3.5.6]

Protože je *plovoucí prvek* vyňat z toku dokumentu, bloky před i za ním jsou zobrazeny, jako by zde vůbec nebyl. Z cesty mu „uhýbá“ pouze jejich obsah (řádkové rámy), bloky samotné nejsou plovoucími prvky ovlivněny.



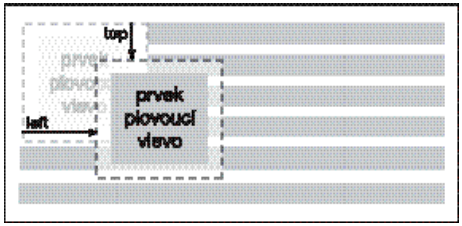
Obr. 33 — Ukázka plovoucího prvku

Každý plovoucí prvek **musí mít udánu šířku** — vlastností `width` nebo svými *skutečnými rozměry* (např. rozměry obrázku). Plovoucí prvek se automaticky stává *blokem.* Jeho horní hrana je zarovnána s horní hranou aktuální řádky, (případně s dolní hranou předchozího bloku, pokud součástí žádné řádky nebyl). Horní a dolní okraje plovoucích prvků se již se sousedícími okraji ostatních rámů *neslučují* [3.8.6.2]. U prvků plovoucích vlevo se dotýká levá hrana levé hrany omezujícího bloku; pokud je zde již jiný (předchozí) plovoucí prvek, dotýká se jeho pravé hrany. U prvků plovoucích vpravo je tomu analogicky naopak.



Obr. 34 — Ukázka plovoucích prvků a ukončení obtékaní vlastností clear

Pokud je plovoucí prvek *relativně pozicován,* stejně jako v normálním toku dokumentu jsou prvek i jeho okolí nejprve zformátovány bez posunutí a teprve následně se prvek odsune podle zadaných hodnot. Pokud je tedy relativně posunut do prostoru ostatních prvků, bude je překrývat.



Obr. 35 — Posunutí plovoucího prvku relativním pozicováním

3.8.6.6 Absolutní pozicování

S hodnotami `position:absolute` a `position:fixed` je prvek **absolutně pozicovaný.** Automaticky se stává *blokem* a je zcela vyňat z toku dokumentu (nijak neovlivňuje formátování následujících prvků). Vytváří nové *omezující bloky* pro své *potomky* a pro všechny své absolutně pozicované *následovníky.* Prvek již neobtéká ostatní obsah ani sám není ostatními prvky obtékán, absolutně pozicované prvky se s ostatním obsahem mohou navzájem překrývat (viz *vrstvy* dále). Okraje pozicovaných prvků se *neslučují,* jako tomu je v normálním toku [3.8.6.2].

Pozice prvku se odvozuje od polohy a rozměru jeho *omezujícího bloku.* Ten je určen nejbližším nadřazeným absolutně pozicovaným prvkem, omezující blok tvoří jeho *hrana výplně* (oblast uvnitř rámečku, [3.8.3]). Pozor, v normálním toku dokumentu tvoří omezující blok až *hrana obsahu;* při absolutním pozicování se však výplň ignoruje. Pokud neexistuje žádný pozicovaný předchůdce prvku, jeho omezující blok tvoří *výchozí omezující blok* dokumentu.

Vlastnosti `top`, `right`, `bottom` a `left` prvku pak udávají absolutní souřadnice vzhledem k hranám omezujícího bloku.

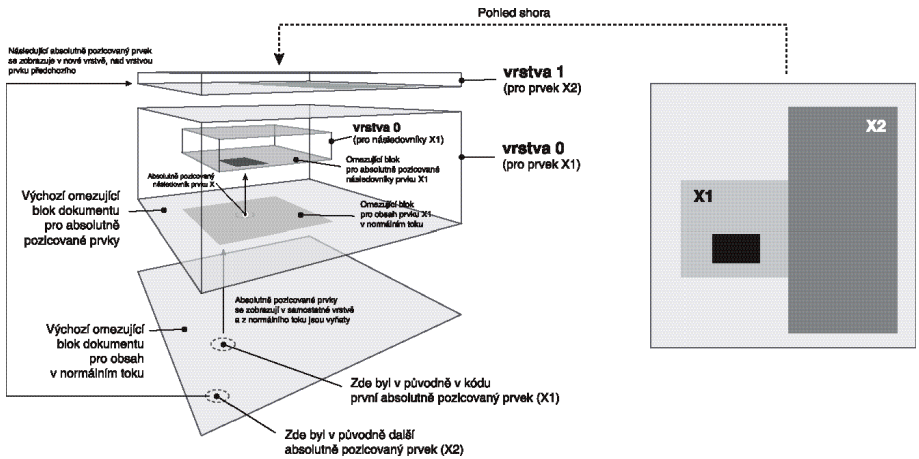
—> Vlastnosti top, right, bottom a left [4.3.5.3]

3.8.6.7 Vrstvy dokumentu

Tím, že se *absolutně pozicované* prvky vyjímají z ostatního toku dokumentu, vznikají vlastně dvě paralelní, na sobě nezávislá zobrazení obsahu dokumentu. Normální tok dokumentu tvoří **základní vrstvu,** nad ní se zakládají **další vrstvy** pro každý absolutně pozicovaný prvek.

Do vrstvy vytvořené pro pozicovaný prvek se umísťuje jeho obsah v normálním toku. Pro jeho *následovníky,* které jsou také absolutně pozicované, se opět vytvářejí další **podvrstvy.** Všechny podvrstvy vytvořené pro následovníky jsou vytvořeny v rámci původní vrstvy prvku — vrstva dalšího prvku je umístěna až nad nimi.

Tyto vrstvy si můžeme představit jako skleněné krabice. Na jejich dno se umísťuje obsah v normálním toku, pro každý absolutně pozicovaný prvek vznikne uvnitř původní krabice krabice nová, překrývající obsah pod ní.



Obr. 36 — Vrstvy dokumentu

Prvky v CSS tak vlastně mají trojrozměrné souřadnice — k plošnému udání pozice je přidána ještě „výšková“ souřadnice. Prvky umístěné „výše“ překrývají prvky „pod nimi“. Hodnotu této souřadnice ovlivňuje vlastnost `z-index`. Její výchozí hodnotou je vždy 0, můžeme ji ale změnit — přiřazením hodnoty `z-index` změníme pořadí vrstvy vytvořené pro daný prvek (hodnota může být i záporná). Pokud mají dva prvky stejnou hodnotu `z-index`, pořadí jejich vrstev se řídí jejich pořadím ve stromu dokumentu — pozdější prvky jsou umístěny výše.

3.8.6.8 Fixní pozicování

Prvky s `position:fixed` jsou podkategorií absolutně pozicovaných prvků. Jediným rozdílem je, že jejich výchozí *omezující blok* je určen *průzorem* do dokumentu. Znamená to, že fixně pozicované prvky jsou umístěny relativně k okrajům tohoto *průzoru* (např. okna dokumentu v prohlížeči). Posun průzoru po dokumentu neovlivňuje jejich pozici a při posouvání tak zůstávají stále na původním místě.

Na *stránkovaných médiích* se fixně pozicované prvky zobrazí na stejném místě na každé stránce.

3.8.7 Výpočet rozměrů rámů

Jak již bylo několikrát řečeno, rozměry rámů závisí na mnoha faktorech a jejich vzájemných vztazích. Zde všechny tyto varianty popíšeme.

Pro potřeby výpočtu rozměrů používá CSS rozdělení prvků na **nahrazované** a **nenahrazované.** Každý typ je potom formátován odlišným způsobem. **Nahrazované prvky** jsou takové, které jsou ve výsledku nahrazeny obsahem, který nepochází přímo ze stromu dokumentu. V HTML jsou typickými příklady prvky `object` nebo `img` — např. prvek `img` je nahrazován obrázkem uvedeným v atributu `src`. U těchto prvků jsou dostupné jejich *skutečné rozměry,* neboli rozměry definované prvkem samotným (např. rozměry obrázku).

Ostatní prvky jsou **nenahrazované.**

Vypočítané rozměry prvků záleží také na definovaných hodnotách vlastností `width` (šířka) a `height` (výška).

—> Vlastnosti width a height [4.3.4]

3.8.7.1 Výpočet šířky, výšky a okrajů

Vypočítané hodnoty vlastností `width`, `margin-left`, `margin-right`, `left`, `right` a `height`, `margin-top`, `margin-bottom`, `top`, `bottom` obvykle odpovídají definovaným hodnotám, s několika výjimkami a níže uvedenými upřesněními. Jedná se především o pravidla pro výpočet hodnoty ze zadaných hodnot `auto`.

3.8.7.2 Řádkové nenahrazované prvky

Vlastnosti `width` a `height` se zde vůbec nepoužívají, vypočítaná hodnota `auto` v `margin-left`, `margin-right`, `top`, `bottom`, `left` a `right` je 0.

Výška obsahu závisí pouze na použitém písmu a jeho velikosti. Výsledná výška *bloku řádky* se potom počítá algoritmem vycházejícím z výšky jednotlivých řádkových rámů, vlastností `line-height` a `vertical-align` [3.8.7.11]. Oblasti výplně, rámečku a okrajů u řádkových prvků se do výšky řádky nepočítají. Vertikálně se řádky formátují, jakoby zde tyto oblasti nebyly

a ty jsou vykreslovány kolem prvků dodatečně. Tyto oblasti tedy mohou přesahovat do sousedních řádek (viz příklad v [3.8.6.3]).

3.8.7.3 Řádkové nahrazované prvky

Vypočítaná hodnota `auto` v `top`, `bottom`, `left` a `right` a všech vlastnostech `margin` je 0. Pokud `width`, resp. `height` je `auto`, záleží na hodnotě druhé vlastnosti — je-li také `auto`, za vypočítané hodnoty `width` a `height` se dosadí *skutečné rozměry* prvku; v opačném případě se `width`, resp. `height` vypočte z hodnoty druhé vlastnosti podle poměru stran daného *vnitřními rozměry*, aby byly zachovány proporce prvku.

3.8.7.4 Blokové nenahrazované prvky v normálním toku

Vypočítaná hodnota `auto` v `top`, `bottom`, `left`, `right`, `margin-top` a `margin-bottom` je 0.

Při výpočtu **šířky** se ostatní hodnoty `auto` vypočítají tak, aby byla zachována rovnost:

```
margin-left + border-left-width + padding-left + width + padding-right
+ border-right-width + margin-right = šířka omezujícího bloku
```

Pokud všechny uvedené hodnoty mají definovánu hodnotu jinou než `auto` (příliš mnoho definic) hodnota `margin-right` se ignoruje. Pokud je `width:auto`, všechny ostatní hodnoty `auto` získají hodnotu 0 a `width` se vypočítá z rovnice. Pokud jsou oba okraje (`margin`) shodně `auto`, jejich vypočítané hodnoty budou shodné (což zajistí vystředění obsahu uvnitř omezujícího bloku).

Je-li `height:auto`, **výšku** prvku tvoří výška jeho obsahu. Obsahuje-li tedy prvek jen řádky, je jeho výškou vzdálenost horní hrany první řádky a spodní hrany poslední řádky. Obsahuje-li prvek bloky (včetně *anonymních bloků*), je jeho výška tvořena součtem výšek jejich *vnějších hran,* přičemž se respektují všechny sloučené okraje [3.8.6.2].

3.8.7.5 Blokové nahrazované prvky v normálním toku

Vypočítaná hodnota `auto` v `top`, `bottom`, `left` a `right` je 0. Hodnota `auto` u `width` a `height` se vypočítává proporcionálně stejně jako u řádkových nahrazovaných prvků [3.8.7.3]. Pokud je jeden z okrajů `auto`, jeho hodnota se vypočítá rovněž z těchto proporcí. Pokud jsou `auto` oba protější okraje, jejich vypočítaná hodnota bude shodná.

3.8.7.6 Plovoucí nenahrazované prvky

Vypočítaná hodnota `auto` ve `width`, `top`, `bottom`, `left` a `right` a všech vlastnostech `margin` je 0. Výška se počítá stejně jako v normálním toku (součet výšek obsahu).

3.8.7.7 Plovoucí nahrazované prvky

Postup je shodný jako u řádkových nahrazovaných prvků.

3.8.7.8 Absolutně pozicované nenahrazované prvky

Pro výpočet **horizontálních rozměrů** se vychází z rovnosti:

```
left + margin-left + border-left-width + padding-left + width +
padding-right + border-right-width + margin-right + right = šířka
omezujícího bloku
```

Pokud `left`, `right` ani `width` nejsou `auto`, vyřeší se rovnice pro `margin-left`, resp. `margin-right`. Jsou-li oba `auto`, musí být shodné; není-li `auto` ani jeden z nich, ignoruje se hodnota `right`.

Pokud jsou `left`, `right` i `width` všechny shodně `auto`, za `left` se použije pozice, kterou by měl prvek v normálním toku dokumentu a provede se výpočet podle bodu 2 níže.

Ve všech ostatních případech se za `auto` v `margin-left` a `margin-right` dosadí 0 a provede se jeden z následujících kroků:

- `left` a `width` jsou `auto`, `right` není `auto` — šířka obsahu se přizpůsobí dostupnému prostoru (viz níže) a `left` se dopočítá
- `right` a `width` jsou `auto`, `left` není `auto` — šířka obsahu se přizpůsobí dostupnému prostoru (viz níže) a `right` se dopočítá
- `left` a `right` jsou `auto`, `width` není `auto` — pro `left` se použije pozice z normálního toku dokumentu (viz výše) a `right` se dopočítá
- má-li `auto` jen jeden z trojice `left`, `width` a `right`, jeho hodnota se dopočítá z rovnice

Pokud je řečeno, že šířka obsahu se má „přizpůsobit dostupnému prostoru“, prohlížeč by měl použít algoritmus podobný tomu, který se používá pro výpočet rozměrů buněk tabulky. Zjednodušeně řečeno, zjistí nejprve *optimální šířku O* (zalomí se řádky pouze tam, kde je to určeno), potom *minimální šířku M* (zalomí se řádky všude, kde je to možné) a nakonec *dostupnou šířku D* (dosadí za `left`, resp. `right` 0 a šířku vypočítá z rovnice). Vypočítaná šířka je potom min(D, max(M, O)).

Obdobně se postupuje při výpočtu **vertikálních rozměrů,** vychází se přitom z rovnosti:

```
top + margin-top + border-top-width + padding-top + height + padding-
bottom + border-bottom-width + margin-bottom + bottom = výška
omezujícího bloku
```

a vlastnostem `left`, `right`, resp. `width` ve výše uvedeném algoritmu odpovídají vlastnosti `top`, `bottom`, resp. `height`.

3.8.7.9 Absolutně pozicované nahrazované prvky

Postup je stejný jako u prvků nenahrazovaných. Jediný rozdíl je v tom, že hodnotu `auto` pro `width` a `height` lze vypočítat ze *skutečných rozměrů* [3.8.7.3], takže se výše uvedený algoritmus provádí s již vypočítanými hodnotami `width`, resp. `height`.

3.8.7.10 Minimální a maximální rozměry prvků

Pokud je definována minimální a/nebo maximální šířka či výška (`min-width`, `max-width`, `min-height`, `max-height`), ve všech uvedených případech se na ně při výpočtu hodnoty `width` a `height` bere zřetel. Neodpovídá-li vypočítaný rozměr zadanému rozsahu, hodnota se mu přizpůsobí a příslušný výpočet se provede znova.

—> Vlastnosti min-width, max-width, min-height, max-height [4.3.4]

3.8.7.11 Výpočet výšky řádek

Řádkové rámy jsou řazeny v posloupnosti *bloků řádek* [3.8.6.3]. **Výška jednoho bloku řádky** se přitom počítá následujícím postupem:

- Je vypočítána výška každého rámu na řádce — výše uvedeným výpočtem [3.8.7.2] a pomocí vlastnosti `line-height` (viz dále)
- Jednotlivé rámy na řádce jsou zarovnány podle svých vlastností `vertical-align`
- Výška řádky je vzdálenost mezi horní hranou nejvyššího rámu a spodní hranou rámu nejnižšího

—> Vlastnosti line-height a vertical-align [4.3.4]

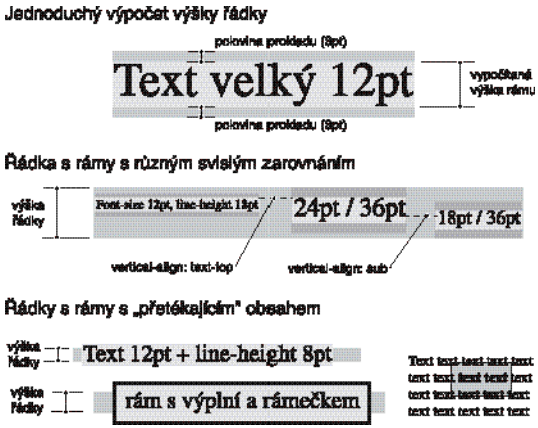
Vlastnost `line-height` definovaná pro *řádkové prvky* specifikuje jejich výšku; definovaná pro blokové prvky definuje **minimální výšku** pro všechny řádkové prvky, které se v bloku budou vyskytovat.

Hodnota vlastnosti `line-height` může být jiná než vypočítaná výška prvku. Rozdíl mezi nimi se nazývá **proklad.** Text se umísťuje na osu řádkového rámu a nahoru i dolů se připočítává polovina *prokladu.* Má-li např. v prvek velikost písma `font-size:12pt` a `line-height` je definována `18pt`, je proklad 18-12 = 6pt. Nad i pod text prvku se proto přidají ještě 3pt.

`Line-height` ale může být naopak i menší, *proklad* je potom záporný. Polovina prokladu se pak od výšky rámu odečítá, výška řádky může být menší než výška prvku a znaky textu mohou dokonce přesahovat do sousedních řádek. Do výšky řádky se také nezapočítávají oblasti *padding, border* a *margin* (vychází se pouze z výšky obsahu prvku). Tyto oblasti jsou však stále kolem prvku vykresleny — může se tedy opět stát, že budou výšku řádky přesahovat. Klient je sice oprávněn je „ořízнout“ podle výšky řádky a přesahující části oblastí nezobrazit. Většinou to ale prohlížeče nedělají.

Pokud tedy řádek tvoří jediný prvek obsahující pouze text, výška všech řádek (vzdálenost mezi účařím textu) je stejná a odpovídá hodnotě `line-height`. Pokud se však na jedné řádce

vyskytuje více prvků s různou výškou, s různým vertikálním zarovnáním či obsahuje *nahrazované prvky* (např. obrázky), výška řádek se může lišit.



Obr. 37 — Výpočet výšky řádek

3.8.8 Další vizuální efekty

3.8.8.1 Přetékání a ořezávání

Za normálních okolností je obsah prvku omezen rozměry svého *omezujícího bloku.* V určitých případech jej však může přesahovat. Obsah **přetéká,** jeho část či celý leží vně omezujícího bloku. Taková situace může nastat např. v těchto případech:

- Nelze rozdělit řádek, řádkový rám je širší než blok řádky
- Blok je příliš široký, než aby se vešel do omezujícího bloku — např. má zadánu velkou hodnotu `width`, obsahuje příliš široký obrázek atd.
- Vnější blok má zadanou konkrétní výšku vlastností `height`, ale jeho obsah je vyšší
- Rám je pozicovaný absolutně, zčásti nebo zcela vně omezujícího bloku
- Rám má záporné okraje (margin), které posouvají část obsahu mimo omezující blok
- atd.

V tom případě nabývá účinnosti mechanismus **přetékání** a **ořezávání obsahu.** Řídí se vlastnostmi `overflow`, případně i `clip`.

Přetékání řídí vlastnost `overflow`. Její výchozí hodnotou je `visible`, což znamená, že případný přetékající obsah je zobrazen i vně *omezujícího bloku.* Toto chování můžeme změnit jinými hodnotami `overflow`. Pomocí hodnoty `hidden` tento obsah skryjeme, přetékající části obsahu nebudou zobrazeny. Hodnota `scroll` zajistí, že klient navíc použije posouvací

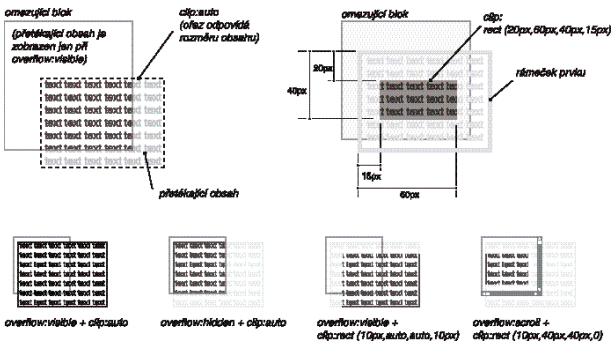
mechanismus, aby uživatel mohl případné nezobrazené části posouvat v rámci viditelné plochy (obdobně jako se posouvá obsah dokumentu v okně prohlížeče).

—> Vlastnost overflow [4.3.6.2]

Dalším doplňkem pro omezení viditelnosti části obsahu je vlastnost `clip`, která řídí jeho **ořezávání.** Výchozí hodnotou je `auto`, prvek není oříznut — jeho viditelná oblast se rovná velikosti obsahu. To můžeme změnit a určit jinou viditelnou oblast. V současnosti jsou v CSS dostupné pouze obdélníkové oblasti, další verze CSS však mohou přidat další možnosti. *Obdélník ořezu* se definuje hodnotou `rect(top,right,bottom,left)`, udávající vzdálenost hran ořezového obdélníku od levého horního rohu *hrany rámečku* prvku (oblast okraje se tedy při ořezu ignoruje — je vždy průhledná a tedy neviditelná).

Prohlížeč vždy nejprve zformátuje prvek celý, jak by byl zobrazen bez použití těchto vlastností, ale zobrazí jen plochu zadanou v `clip`. A pokud není `overflow:visible`, zobrazí navíc jen tu část, která je uvnitř *omezujícího bloku* prvku.

—> Vlastnost clip [4.3.6.3]



Obr. 38 — Přetékání a ořezávání obsahu

3.8.8.2 Viditelnost

Zcela skrýt prvek můžeme vlastností `visibility`. Její výchozí hodnotou je `visible`, tj. prvek je viditelný. Hodnota `hidden` způsobí, že prvek zobrazen nebude. Je důležité si ale pamatovat, že je stále v dokumentu zformátován a umístěn. Ovlivňuje nadále formátování okolních prvků úplně stejně, jako by byl vidět —vlastně ve stránce zobrazen je, ale jako zcela průhledný.

—> Vlastnost visibility [4.3.6.1]

K podobnému účelu se využívá také hodnota `display:none`, ta však způsobí, že prvek zmizí zcela. Neovlivní tedy okolní prvky a ty jsou zformátovány tak, jako by v dokumentu vůbec nebyl.

—> Vlastnost display [4.3.5.1]

Protože se neviditelnost prvků využívá především na dynamických stránkách (skrývání a zobrazování prvků pomocí skriptů), je obzvlášť důležité pamatovat na tento rozdíl v definování viditelnosti. Pokud tedy dynamicky (např. *Javascriptem*) změníme viditelnost vlastností `visibility`, prvek se zobrazí či zmizí, aniž by to jakkoli ovlivnilo jeho okolí. Pokud uděláme totéž pomocí vlastnosti `display`, způsobíme přeformátování stránky, což při rychlejších akcích a především v animacích může přinést nevídané potíže. Nicméně každá z obou možností má své místo k použití a obecně nelze dávat přednost jen jedné z nich. Je pouze na autorovi, aby zvážil všechny důsledky podobné operace.

3.8.8.3 Formátování textu

Text nese hlavní informační hodnotu dokumentů a obvykle tvoří naprostou většinu obsahu webových stránek. Definování jeho vzhledu je také jednou z nejvyužívanějších funkcí CSS. K formátování textů slouží dvě skupiny vlastností.

Vlastnosti písma popisují vzhled jednotlivých znaků (typ písma, velikost, sklon či sílu). Obvykle tuto část formátování obstarává *operační systém* počítače. Prohlížeč mu předá požadavek na zobrazení znaku písmem daných vlastností a od operačního systém dostane zpět grafickou prezentaci, která požadovanému znaku odpovídá.

Vlastnosti textu k takto zformátovaným znakům přidávají další efekty. *Nastylovaný text* může prohlížeč doplnit o různé dekorace (podtržení, přeškrtnutí), změnit mezery mezi znaky či slovy, odsazení prvního řádku odstavce, převést velké znaky na malé (a naopak) či změnit způsob *zalamování* a *zarovnávání* řádek.

3.8.8.4 Vzhled písma

Vzhled písma v CSS definují vlastnosti `font-family` (typ, rodina písma), `font-size` (velikost), `font-style` (styl, sklon), `font-weight` (síla, tučnost) a `font-variant` (varianta písma, kapitálky). Z typografického hlediska se k nim řadí i `line-height` (výška řádek). *Sdruženou vlastností* `font` můžeme určit všechny tyto vlastnosti současně. Vlastnost `font` také umožňuje použít pro text některé z písem, které je použito v *grafickém prostředí uživatele (GUI).*

—> Vlastnosti písma [4.4.1]

Dosud neexistuje žádná všeobecná norma pro definování vzhledu písma. CSS jej ale ze své podstaty musí popisovat co nejjednodušeji, obecně a snadno pochopitelně. Např. pro skloněné písmo používá obecný název *italic*, ve skutečnosti se však skloněné řezy písem označují poměrně libovolně — *Italic, Oblique, Cursive, Kurziv, Skloněné, Incline, Slanted* atd. Proto CSS alespoň definuje postup pro nalezení *co možná nejvhodnějšího* písma pro každý znak textu.

1. Prohlížeč si nejprve v databázi shromáždí všechny výše uvedené vlastnosti CSS pro každé použitelné písmo (písma nainstalovaná v operačním systému, stažená z Internetu či jinak dostupná). Pokud má k dispozici dvě písma stejných vlastností, jedno z nich ignoruje. CSS neurčuje přesný postup, jakým prohlížeč vlastnosti písem zjistí.

2. Pro každý znak textu v daném prvku shromáždí všechny vlastnosti písma, která lze na prvek použít. Podle hodnoty vlastnosti `font-family` zkusmo vybere vhodnou *rodinu písma*. Na této rodině otestuje ostatní vlastnosti. Pokud **všechny** vyhoví, příslušný řez písma se použije pro celý prvek. Jednotlivé vlastnosti se testují v tomto pořadí:
    a. Nejprve se zkouší vlastnost `font-style` (nastavení sklonu). Hodnotě *italic* vyhoví řez písma označený prohlížečem jako *italic* nebo *oblique*. V ostatních případech musí hodnota odpovídat přesně, jinak `font-style` nevyhoví. Skloněné písmo však může být vytvořeno i uměle, elektronickým skloněním základního písma.
    b. Jako další se testuje `font-variant` (nastavení kapitálek). Hodnotě *normal* vyhoví řez písma, který není označen jako *small-caps*. Hodnotě *small-caps (kapitálky)* vyhoví řez označený jako *small-caps*. Pokud takový není, kapitálky mohou být vytvořeny uměle — zmenšením velkých znaků (*verzálek*), případně jen nahrazením malých znaků (*mínusek*) za verzálky.
    c. Následně se testuje `font-weight` (síla písma). Této vlastnosti písmo vyhoví vždy (viz `font-weight`).
    d. Nakonec se vybere vhodná velikost písma (`font-size`). Zde se musí zohlednit i možnosti cílového média (především zaokrouhlení na celé obrazové body) a také tolerance prohlížeče (např. při výběru nejbližší existující velikosti bitmapového písma).
3. Pokud vybraná *rodina písma* testu vlastností nevyhoví a je-li v seznamu písem ve `font-family` uvedena další alternativa, opakuje krok 2 s další položkou.
4. Neobsahuje-li nalezené písmo symbol pro některý znak, opakuje se krok 2 s další alternativou uvedenou ve `font-family` a tento znak se zobrazí náhradním písmem.
5. Pokud opakování kroků 2 a 3 není žádné vhodné písmo nalezeno, použije se krok 2 pro výchozí písmo prohlížeče a najde se písmo nejvíce odpovídající daným vlastnostem. Pokud není možné nějaký znak zobrazit ani výchozím písmem, prohlížeč místo něj zobrazí nějaký symbol s významem „chybějící znak“.

Uživatel má obvykle možnost zakázat prohlížeči použití jiných písem. V tom případě se ignoruje pouze hodnota vlastnosti `font-family`, ostatní vlastnosti se nadále používají. Výše uvedený postup začne až krokem 5.

3.8.8.5 Vzhled textu

Vzhled textu definují vlastnosti `text-indent` (odsazení prvního řádku), `text-align` (vodorovné zarovnání textu), `text-decoration` (dekorace textu), `letter-spacing` a `word-spacing` (meziznakové a mezislovní mezery, *prostrkání*), `text-transform` (převod malých/velkých znaků, *mínusky—verzálky*) a lze k nim řadit i vlastnost `white-space` (řízení lámání řádek).

—> Vlastnosti textu [4.4.1], vlastnost white-space [4.3.6.4]

Výslednou podobu textu ovlivňují i *vlastnosti barev*.

3.8.8.6 Barva textu a styl pozadí

Barvu textu určuje vlastnost `color`. Její výchozí hodnota závisí na prohlížeči, případně i na předvolbách uživatele.

—> Vlastnost color [4.5.1]

Pro styl pozadí prvků slouží skupina vlastností **background.** Pozadí je zobrazeno na celé ploše *oblastí obsahu, výplně a rámečku* (*okraje* jsou vždy průhledné). Rámeček a samotný obsah prvku jsou vykresleny nad touto plochou (překrývají ji). Pozadí *kořenového prvku* vyplňuje celou zobrazovací plochu dokumentu (doporučuje se používat raději prvek `body` než `html`). Pokud není v CSS specifikováno, styl pozadí dokumentu závisí na prohlížeči.

Vlastnost `background-color` specifikuje barvu pozadí. Výchozí hodnotou je `transparent`, tj. průhledné pozadí — dokud prvkům pozadí neurčíme, budou průhledné a pod nimi bude prosvítat pozadí jejich rodičovského prvku (případně i obsah prvků, které překrývají).

Vlastností `background-image` můžeme na pozadí umístit obrázek. Další vlastnosti řídí jeho umístění a opakování. Opakování obrázku se definuje vlastností `background-repeat` — obrázek může být na pozadí umístěn pouze jednou, nebo se může dlaždicově opakovat ve svislém, vodorovném či obou směrech. V místech, kde obrázek není, a pod jeho průhlednými oblastmi je zobrazena plocha daná hodnotou `background-color` (případně nic, pokud je hodnota `transparent`). Umístění obrázku se řídí vlastností `background-position`. Obrázek se může dotýkat hrany prvku, lze jej zarovnat na osu nebo posunout v libovolném směru.

Vlastnost `background-attachment` definuje vazbu mezi pozadím a obsahem prvku v situacích, kdy se obsah posouvá při *přetékání* [3.8.8.1]. Obvykle se pozadí posouvá synchronně s obsahem, hodnotou `fixed` však můžeme nařídit, aby pozadí zůstalo na místě a posouval se pouze obsah. Nejčastěji se *fixované pozadí* používá v prvku `body`. Nastavíme tím pozadí celého dokumentu v rámci *průzoru dokumentu* a obsah se bude posouvat nad ním — pozadí zůstane na místě.

—> Vlastnosti pozadí [4.5.2]

3.8.8.7 Uživatelské rozhraní

Interaktivních vizuálních médií (tedy především prohlížeče v počítači) se týká i několik dalších dílčích vlastností CSS, jejichž podpora v prohlížečích však ještě dost kolísá.

Vlastnost `cursor` určuje, jaký ukazatel (kurzor myši) se má zobrazit, když uživatel ukáže na prvek. Pomocí dynamických *pseudo-tříd* (`:hover`) lze definovat i vzhled prvku samotného. Většina prohlížečů však dynamické pseudo-třídy aplikuje pouze na odkazy (`<a>`), u ostatních prvků je nepodporuje. Lze také definovat, jakým způsobem se prvek označí při změně *zaměření*

klávesnicí; styl orámování prvku v tomto případě určuje vlastnost `outline`. Tato vlastnost je podporována ještě sporadičtěji.

—> Vlastnosti cursor [4.7.1] a outline [4.7.2]

Z aktuálního uživatelského rozhraní také vycházejí některé další informace, které CSS používá — např. barvy nebo písma používané operačním systémem (viz výše).

Často řešeným problém je i otázka zvětšování (*zoom*) dokumentu. CSS ji přímo neřeší, jedinou podmínkou, kterou zmiňuje, je nutnost zachování poměrů velikostí a vzdáleností u absolutně pozicovaných prvků. Závisí tak na prohlížečích, jak jejich tvůrci možnost zvětšení vyřeší. Většina umožňuje pouze zvětšit písmo, ale MSIE ve Windows přitom ignoruje velikosti zadané pomocí `px`. Jiné prohlížeče (např. Opera) zvětší obsah stránky kompletně, jako zvětšovacím sklem. Je potřeba s tím počítat a každý autor by měl předpokládat i to, že si jeho stránku může uživatel prohlížet např. s písmem zvětšeným na 200 %.

3.9 Obsah generovaný CSS, seznamy

CSS umožňuje autorům přidávat do dokumentů obsah, který v samotném dokumentu neexistuje. Nejvíce je tato funkce spojena se *seznamy.* Můžeme chtít, aby byly jejich položky číslované, aby se před nimi zobrazovaly odrážky atd. Ale netýká se to pouze seznamů — před názvy kapitol můžeme chtít zobrazit text "Kapitola", automaticky vytvořit popisky obrázku "Obrázek: <popiska>" atd. To vše v CSS umožňuje **generovaný obsah.**

Obsah do dokumentů přidávají dva mechanismy CSS:

- vlastnost `content` v pseudo-prvcích `:before` a `:after`
- položky seznamů, definované pomocí vlastnosti `display:list-item`

3.9.1 Obsah generovaný pseudo-prvky :before a :after

Pseudo-prvky :before a :after se používají výhradně ve spojení s nějakým prvkem a definují obsah před (*before*) a za (*after*) ním. Obsah, který se má před/za prvek vložit, se definuje vlastností `content`, která se používá výhradně v těchto pseudo-prvcích.

—> Selektory s :before a :after [3.6.5.9], vlastnost content [4.6.1]

Obsah generovaný vlastností `content` je především text. V popisu CSS 2 sice můžeme najít i další možnosti, jako vložení obrázku pomocí jeho URL, automatická počítadla atd., ale tyto funkce zatím nejsou příliš podporovány a specifikací CSS 2.1 byly opět odstraněny — připravovaná verze CSS 3 bude tuto problematiku řešit zcela jinak a mnohem komplexněji.

3.9.1.1 Textový obsah

Generovaný text můžeme definovat přímo jako `<řetězec>`. Např.:

```
p.poznamka:before { content: "Poznamka: " }
```

Na začátek prvku `<p class="poznamka">` se vloží text „Poznámka: “ a teprve za ním bude zobrazen samotný obsah prvku.

Pozn.: Hodnotou vlastnosti `content` může být pouze textový řetězec — značky a entity HTML se zde ignorují a zobrazí se stejně jako byly zapsány. Pokud potřebujeme vložit speciální znak, např. *odřádkování,* musíme jej zapsat pomocí příslušné *escape-sekvence*, tedy např. `\A` (viz [3.3.7]). Znak `\A` vynutí v textu odřádkování stejně jako značka `<br>` v HTML.

3.9.1.2 Uvozovky

Klíčovými slovy `open-quote`, `close-quote` umístíme kolem textu **uvozovky.** CSS se již postará o korektní typ uvozovky podle použitého jazyka a úrovně vnoření. Např.:

```
p.citat:before { content: open-quote }
p.citat:after  { content: close-quote }
```

Uvozovky se definují vlastností `quotes`. Její hodnotou je seznam párů řetězců. První z dvojice se použije jako *otevírací* uvozovka, druhý jako *uzavírací.* Další páry definují uvozovky pro další úrovně vnoření (citát v citátu). Výhodně se zde používá pseudo-třída `:lang`, s jejíž pomocí můžeme definovat uvozovky podle různých zvyklostí jednotlivých jazyků. Protože vlastnost `quotes` je dědičná, stačí nastavit uvozovky pro kořen dokumentu a ostatní prvky tuto definic zdědí. Např.:

```
body:lang(en) { quotes: '“' '”' '‘' '’' '>' '<' }
body:lang(cs) { quotes: '„' '“' ',' '‘' '»' '«' }
body:lang(fr) { quotes: '«' '»' '„' '“' ',' '‘' }

q:before { content: open-quote }
q:after  { content: close-quote }
```

S takto definovanými uvozovkami se vnořené uvozovky:

```
<q>Podle prof. Nováka jsou <q>vnořené uvozovky <q>běžnou
záležitostí</q> každého dne,</q> jak řekl na tiskové konferenci.</q>
```

zobrazí (s nastaveným českým jazykem) takto:

„Podle prof. Nováka jsou ‚vnořené uvozovky »běžnou záležitostí« každého dne,‘ jak řekl na tiskové konferenci.“

—> Vlastnost quotes [4.6.1]

3.9.1.3 Vložení hodnoty atributu

Třetí možnou hodnotou vlastnosti `content` je `attr(X)`, která vloží do textu hodnotu atributu X. Např.:

```
img:before { content: attr(alt) }
```

Vloží za každý obrázek text zapsaný v atributu `alt`.

Pozn.: S výhodou můžeme všechny možné hodnoty `content` kombinovat. Pokud např. každý obrázek v dokumentu obsahuje popis v atributu `title`, můžeme vytvořit textové popisky pod obrázky:

```
img:after { content: "\AObrázek: " attr(title) "(URL: " attr(src) ")"
```

Pod obrázkem např. `<img src="simpanz.jpg" alt="Simpanz Bobo" title="Šimpanz Bobo při krmení.">` bude zobrazen text:

Obrázek: Šimpanz Bobo při krmení (URL: simpanz.jpg)

3.9.1.4 Formátování generovaného obsahu

Pseudo-prvky `:before` a `:after` a potažmo jimi vložený obsah můžeme formátovat jako běžné prvky. Generovaný obsah *dědí* všechny dědičné vlastnosti od mateřského prvku. Pokud tedy neurčíme jinak, bude zobrazen stejným písmem a stylem jako text prvku, pro nějž obsah generujeme.

Je-li mateřský prvek *blokový,* můžeme pseudo-prvkům `:before` a `:after` definovat také `display:block`. Generovaný obsah se pak chová jako *blokový prvek* a bude zformátován jako samostatný blok před, resp. za mateřským prvkem. V ostatních případech se generovaný obsah formátuje jako *řádkový prvek.*

Vlastnosti `position`, `float`, `list` a vlastnosti tabulek se ignorují, jinak můžeme použít všechny dostupné vlastnosti a vygenerovaný obsah zformátovat dle libosti. Např.:

```
body:after {
      content: '--- konec stránky ---';
      margin-top: 2em;
      font-size: smaller;
      font-weight: bold;
      color: red;
      text-align: right
      }
```

Pozn.: Jakkoli jsou generované seznamy užitečnou funkcí, nejsou dosud implementovány v nejrozšířenějším prohlížeči MSIE ve Windows. Jiné prohlížeče (Mozilla, Opera, MSIE/Mac) je však podporují.

3.9.2 Seznamy

Seznamy jsou specifickým případem generování obsahu. Zde již nedefinujeme obsah zcela libovolně, pouze si můžeme vybrat z několika přednastavených možností. Navíc je zde ještě jedna odlišnost. Položky seznamů (prvky s `display:list-item`) negenerují pouze jediný *hlavní rám* [3.8.5], ale mohou generovat i *rám doplňkový,* do nějž se umístí značka uvozující položky seznamu.

Umístění *doplňkového rámu* se definuje vlastností `list-style-position`. CSS neumožňuje určit jeho přesnou polohu, pouze určuje, zda tento rám má být **vnější** či **vnitřní** (konkrétní formátování závisí na prohlížeči). *Vnitřní rám* (hodnota `inside`) se formátuje jako první *řádkový rám* prvku. Uvozující značka se tedy umístí *dovnitř* prvku, na místo, kde by jinak začínal samotný obsah, který se tím odsune o kus dál. *Vnější rám* (hodnota `outside`) se umisťuje vně prvku (za *hranu rámečku*) a formátování samotného obsahu prvku nijak neovlivňuje. Pozadí prvku se zobrazuje pouze v *hlavním rámu*, *vnější rám* je vždy průhledný. Např.:

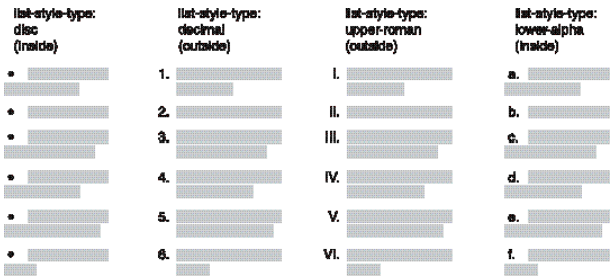
```
li.typ1 { list-style-position: inside }
li.typ2 { list-style-position: outside }
```

```
<ul>
      <li class="typ1">Položka seznamu
      formátovaná s vnitřní značkou</li>
      <li class="typ2">Položka seznamu
      formátovaná s vnější značkou</li>
</ul>
```

- Položka seznamu formátovaná s vnitřní značkou
- Položka seznamu formátovaná s vnější značkou

Jako značku, která se má u položek seznamu zobrazit, můžeme použít vlastní obrázek. Jeho adresu zadáme jako hodnotu vlastnosti `list-style-image`. Pokud obrázek nezadáme (hodnota `none`), nebo není dostupný, použije se druhá vlastnost — `list-style-type`. Její hodnotou je klíčové slovo popisující některou z *předdefinovaných* typů značek. První skupinou značek jsou symboly — `disc` (výchozí hodnota), `circle` a `square`. Druhou tvoří číslované seznamy a třetí skupinu seznamy označované znaky některé z „abeced“. U druhé a třetí skupiny jsou položky postupně „číslovány“ — značkou u první položky je číslo 1, resp. první písmeno abecedy, u dalších položek je značka postupně o jednu zvyšována.

—> Vlastnosti seznamů [4.6.2]



Obr. 39 — Příklady značek u seznamů

3.10 Tabulky

Tabulky slouží k prezentaci závislostí mezi daty. Připomeňme, že jejich účelem rozhodně není realizovat vzhled celé stránky či jejích částí — takový přístup je kontraproduktivní (viz úvod této knihy). Pro svůj účel jsou však tabulky navrženy velmi dobře a jejich funkce jsou stále rozšiřovány.

3.10.1 Tabulkový model CSS

Tabulkový model CSS vychází ze obdobného modelu v HTML 4. Jeho rozsah je však mnohem širší. Tabulku tvoří z obdélníková matice buněk (prvky uspořádané do řádků a sloupců), přičemž model je *primárně řádkový* (autoři specifikují řádky tabulek, sloupce jsou odvozeny automaticky). Součástí tabulky je i volitelné *záhlaví* (caption).

Všechny prvky tabulek jsou uspořádány hierarchicky do několika skupin, každé z nich odpovídá příslušný typ prvku. Tabulkový model CSS nezpracovává pouze prvky, které příslušný jazyk označí jako *tabulkové* (např. prvky `table`, `tr`, `td` v HTML), jeho přístup je obecnější. Prvky jsou zpracovány podle hodnoty vlastnosti `display`. Touto vlastností je možné definovat jako *tabulkový* kterýkoli prvek, což je potřebné především v jazycích, které nemají značky pro tabulky předdefinovány (např. XML).

3.10.1.1 Prvky tabulek

Následující seznam uvádí všechny typy tabulkových prvků a odpovídající hodnotu vlastnosti `display`.

- **Tabulka** (`table`, `inline-table`). Prvek je tabulkou, pokud má jeho vlastnost `display` hodnotu `table` (tabulka se formátuje jako *blokový prvek*) nebo `inline-table` (tabulka se formátuje jako *řádkový prvek*).
- **Nadpis** (`table-caption`). Prvek tvoří nadpis tabulky, může se zobrazovat nad tabulkou, pod ní, ale i vlevo či vpravo od ní.
- **Řádek** (`table-row`). Obsahem prvku je skupina buněk tvořících jeden řádek tabulky.
- **Sloupec** (`table-column`). Prvek popisuje buňky tvořící jeden sloupec tabulky.
- **Skupina řádků** (`table-row-group`). Prvek seskupuje jeden nebo více řádků tabulky.
- **Skupina sloupců** (`table-column-group`). Prvek seskupuje jeden nebo více sloupců tabulky.
- **Skupina záhlaví** (`table-header-group`). Stejně jako `table-row-group`, při vizuálním formátování se však vždy zobrazuje před všemi ostatními řádky a skupinami řádků (ale až za nadpisy umístěnými nahoře). Při tisku je klient může zobrazit na začátku každé stránky pokud je tabulka rozdělena na více stran.

- **Skupina zápatí** (`table-footer-group`). Stejně jako `table-header-group`, zobrazuje se ale až za všemi ostatními řádky (před nadpisy umístěnými dole).
- **Buňka** (`table-cell`). Prvek tvoří jednu buňku tabulky.

Prvky typu `table-row` a `table-row-group` se nezobrazují (stejně jako by měly `display:none`), ale mohou definovat současně vlastnosti pro skupinu buněk tvořících sloupec či více sloupců tabulky.

Tabulkové prvky v HTML již mají hodnoty vlastnosti `display` předdefinovány ve své *výchozí tabulce stylů* (ukázku pro HTML 4 najdete na konci této knihy):

```
table      { display: table }
tr         { display: table-row }
thead      { display: table-header-group }
tbody      { display: table-row-group }
tfoot      { display: table-footer-group }
col        { display: table-column }
colgroup   { display: table-column-group }
td, th     { display: table-cell }
caption    { display: table-caption }
```

Klient však může v HTML vlastnost `display` u těchto prvků ignorovat, protože tabulky v HTML mohou být vykresleny pomocí odlišného algoritmu (kvůli zajištění zpětné kompatibility). Neznamená to ale, že by měl ignorovat použití `display:table` u jiných prvků.

3.10.1.2 Anonymní prvky tabulek

V jiných jazycích než HTML (např. v XML) nemusí dokument obsahovat kompletní strukturu tabulky. Model CSS automaticky doplní chybějící prvky, aby tabulka mohla být zobrazena.

Pokud rodič prvku označeného jako `table-cell` není typu `table-row`, je mezi ně přidán *anonymní prvek* typu `table-row`, který sdruží do řádku všechny buňky v rodičovském prvku. Obdobně vznikne anonymní prvek typu `table`, pokud neexistuje takový rodičovský prvek dalších tabulkových prvků; vytvoří se anonymní prvky typu `table-cell`, pokud nějaký prvek `table-row` není takto označen atd. Klient však není povinen tyto anonymní bloky vytvářet, autoři by proto měli strukturu tabulek definovat co nejpřesněji.

3.10.2 Formátování sloupců

Každá buňka tabulky patří současně do dvou kontextů — do řádku a do sloupce. Protože je však tabulkový model *řádkový* (viz výše), je buňka ve *stromu dokumentu* potomkem nějakého řádku, nikdy ne sloupce. Přesto je však možné definovat vlastnosti pro sloupce a ty se pak použijí na všechny buňky, které jej tvoří. Protože se sloupcový kontext poněkud vymyká ze standardní struktury dokumentu, jsou pro prvky typu sloupec (resp. skupina sloupců) povoleny jen následující vlastnosti, navíc s poněkud pozměněným významem:

- `border` — vlastnosti rámečků jsou povoleny pouze když má tabulka definován `border-collapse:collapse`, navíc zde vstupuje do hry algoritmus pro řešení konfliktů (viz dále).
- `background` — pozadí je nastaveno všem buňkám ve sloupci, ale jen pokud mají všechny buňky i řádky `background:transparent`.
- `width` — definuje *minimální šířku* buněk ve sloupci
- `visibility` — povolena je pouze hodnota `collapse`. S touto hodnotou nebude žádná buňka sloupce vykreslena a šířka tabulky se zmenší o šířku tohoto sloupce. Používá se především pro dynamické efekty tabulek [3.10.4.3].

Jiné vlastnosti zde nemají žádný efekt.

Příklad:

```
col { background: white }
col.zahlavi { background: yellow }
col.vysledek { background: blue; border-left: 4px solid black }
...
<table>
<col class="zahlavi"><col span="2"><col class="vysledek">
<tr> <th>2000</th> <td>10,2</td> <td>8,6</td> <td>19,0</td> </tr>
<tr> <th>2001</th> <td>12,0</td> <td>8,4</td> <td>20,4</td> </tr>
<tr> <th>2002</th> <td>14,6</td> <td>9,9</td> <td>24,5</td> </tr>
</table>
```

První sloupec (`class="zahlavi"`) bude zobrazen se žlutým pozadím, poslední sloupec (`class="vysledek"`) s pozadím modrým a silnějším rámečkem vlevo. Ostatní sloupce budou mít pozadí bílé.

3.10.3 Tabulky ve vizuálním formátovacím modelu

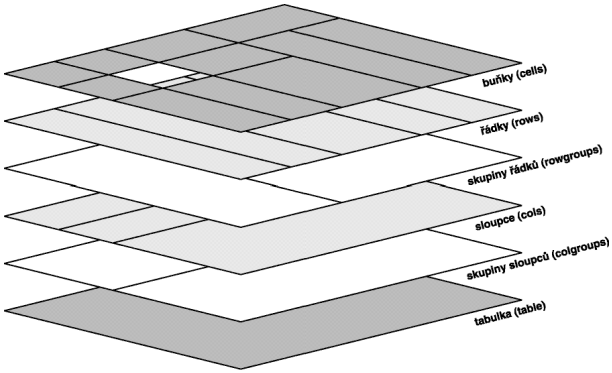
Z hlediska *vizuálního formátovacího modelu* [3.8] se tabulky chovají jako *blokové prvky*, resp. jako *nahrazované řádkové prvky* (`inline-table`). V obou případech se však nejprve vygeneruje *anonymní blokový rám,* který teprve obsahuje rám samotné tabulky a také rám jejího nadpisu (caption). Umístění nadpisu se řídí vlastností `caption-side` [4.8.1].

3.10.3.1 Vzhled tabulek

Vnitřní prvky tabulek generují rámy stejně jako kterékoli jiné prvky. Mají oblasti obsahu, výplně i rámeček, narozdíl od ostatních prvků **nemají okraje** (margin).

Buňky tabulek sice mohou být i plovoucí či pozicované, specifikace to ale nedoporučuje. Takový prvek je vyňat z tabulky, což ovlivní její rozměry a může přinést nepředvídatelné výsledky.

Pro určení stylu pozadí jednotlivých částí tabulky si lze představit, že její prvky jsou umístěny v šesti vrstvách. Skrze průhledné pozadí jedné vrstvy lze vidět pozadí vrstev nižších, plné pozadí nižší vrstvy překrývá. Pořadí prvků a jejich vrstev je na následujícím obrázku:



Obr. 40 — Vrstvy v tabulkách

Jak z obrázku vyplývá, např. pozadí řádku překrývá pozadí tabulky; pozadí jedné buňky má přednost před všemi ostatními prvky atd. Pokud je v některém místě tabulky průhledné pozadí ve všech prvcích (včetně tabulky samotná), bude zde prosvítat pozadí rodičovského prvku. Má-li tabulka `border-collapse:separate`, prázdná místa kolem buněk daná vlastností `border-spacing` jsou vždy zobrazena barvou pozadí tabulky (pozadí prvku `table`).

3.10.3.2 Výpočet šířky tabulky

CSS nespecifikuje, jak má klient zobrazovat tabulku, ale pouze omezení, která při tom musí dodržovat. Klient může použít jakýkoli algoritmus, může i upřednostňovat rychlost formátování tabulek na úkor jejich přesnosti — ovšem vyjma případů, kdy má být použit *fixní model* formátování (viz dále).

Je třeba připomenout, že pravidla v této kapitole nahrazují obecná pravidla CSS pro výpočet rozměrů [3.8.7]. Např. má-li tabulka danou šířku `auto` a její okraje (`margin`) mají hodnotu `0`, tabulka nemusí vyplnit celou šířku omezujícího bloku. Na druhé straně, je-li níže uvedeným postupem (či jiným postupem, který používá klient) vypočtena její šířka, použijí se obecná pravidla pro výpočet rozměrů — tabulka tak *může* být zarovnána na osu, má-li okraje s hodnotou `auto`.

Model, který se použije pro formátování tabulky, se definuje vlastností `table-layout`. S hodnotou `fixed` se použije **fixní model,** s hodnotou `auto` pak **automatický model.**

–> Vlastnost table-layout [4.8.1]

3.10.3.3 Fixní model

Tento (rychlejší) formátovací model nezohledňuje obsah buněk tabulky. Její vodorovné rozměry závisí jen na udané šířce tabulky, šířce sloupců a šířkách rámečků a výplní buněk. Šířka tabulky musí být určena vlastností `width` — hodnota `auto` způsobí, že se použije *automatický model* formátování.

Ve fixním modelu se šířka sloupců počítá takto:

- Má-li sloupcový prvek hodnotu `width` jinou než `auto`, určuje šířku tohoto sloupce.
- V opačném případě, má-li první buňka ve sloupci hodnotu `width` jinou než `auto`, určuje šířku pro celý sloupec. Přesahuje-li tato buňka přes více sloupců, její šířka je rovnoměrně rozdělena mezi tyto sloupce.
- Všechny ostatní sloupce si stejnoměrně rozdělí zbývající vodorovný prostor (po odečtení rámečků a výplní).

Je-li součet šířek sloupců větší než šířka definovaná pro tabulku vlastností `width`, tabulka bude širší, než udává tato hodnota. Pokud je součet šířek sloupců naopak menší, zbývající prostor se mezi ně rovnoměrně rozdělí. Případné přetékání obsahu buněk se řídí jejich vlastností `overflow`.

Při použití tohoto modelu může prohlížeč začít vykreslovat tabulku ihned, jak obdrží první řádek — obsah další řádků již rozměry tabulky neovlivňují.

3.10.3.4 Automatický model

Tento model zohledňuje algoritmy používané v nejznámějších prohlížečích. Žádný klient však není povinen jej používat — pokud má tabulka definován `table-layout:auto`, může použít jakýkoli jiný algoritmus.

Algoritmus pro tento model je méně efektivní než ve fixním modelu, neboť před zahájením formátování vyžaduje načtení celého obsahu tabulky (a může vyžadovat více než jeden průchod tabulkou). Šířka tabulky je dána šířkou jejích sloupců a šířkou rámečků.

V automatickém modelu se šířka sloupců počítá takto:

- Nejprve spočítej minimální a maximální šířky:
    - Spočítej minimální a maximální šířku každé buňky. *Minimální* je taková šířka, aby obsah buňky nepřetékal, když se zalomí řádky všude, kde je to možné. *Maximální* je taková šířka, kdy se řádky v buňce zalomí pouze na autorem určených místech. Pokud má buňka šířku definovánu (`width` není `auto`) a je větší než vypočítaná minimální/maximální šířka, použij vždy větší z obou hodnot.
    - Najdi minimální a maximální šířku každého sloupce. Určuje je buňka s největší minimální, resp. maximální šířkou.

    - Pokud některá z buněk přesahuje přes více sloupců, zvětši minimální a maximální šířku těchto sloupců tak, aby byly alespoň stejně široké jako tato buňka. Pokud je to možné, zvětšuj všechny sloupce stejnoměrně.

- Tyto hodnoty použij pro zformátování tabulky:
    - Má-li tabulka zadanou šířku (`width` není `auto`), její vypočítanou šířkou bude tato hodnota `width`, nebo součet *minimálních šířek* všech sloupců (plus šířka rámečků a výplní) — podle toho, která z hodnot je větší. Je-li větší zadaná šířka, zbývající prostor se rozdělí stejnoměrně mezi všechny sloupce. Je-li menší, tabulka bude širší, než definoval autor.
    - Má-li tabulka `width:auto`, její vypočítanou šířkou bude šířka jejího *omezujícího bloku,* nebo součet *minimálních šířek* všech sloupců (plus šířka rámečků a výplní) — podle toho, která z hodnot je větší. Pokud je však součet *maximálních šířek* sloupců menší než šířka omezujícího bloku, použije se tato hodnota (tabulka nevyplní celou dostupnou šířku).

Procentní šířky buněk a sloupců jsou relativní k šířce tabulky. Pokud má tabulka `width:auto`, procentní šířka představuje omezující podmínku pro výpočet šířky sloupce, kterou by se měl prohlížeč pokusit dodržet — ne vždy to je ale možné.

Při tomto algoritmu se navzájem ovlivňují šířky sloupců a výšky řádků — zadá-li autor explicitně výšku řádku či buňky, promítne se to do výpočtu minimálních šířek sloupců, a naopak.

3.10.4 Výpočet výšky tabulky

Výšku tabulky určuje vlastnost `height`. Má-li hodnotu `auto`, výšku tvoří součet výšek všech řádků (plus rámečky a výplně). Jiná hodnota určuje exaktní výšku. CSS však nespecifikuje, jak má prohlížeč tabulku formátovat, když se udaná výška liší od součtu výšek řádků.

Výška řádku (prvku `table-row`) se počítá ve chvíli, kdy jsou načteny všechny buňky na řádku. Je to maximum z udané výšky řádku (vlastností `height`) a minimálních výšek, které vyžadují jednotlivé buňky. CSS neurčuje, k čemu se vztahují procentní výšky udané pro řádky a skupiny řádků.

Výška buňky je maximum z udané výšky (`height`) a minimální výšky, kterou vyžaduje její obsah (závisí také na vypočítané šířce sloupce, viz výše). CSS neurčuje, k čemu se vztahují procentní výšky definované pro buňky. Není také definováno, jak výšku řádků ovlivňují buňky, které přesahují přes více řádků. Platí pouze, že výška řádků musí být dostatečná na to, aby takové buňky pojaly.

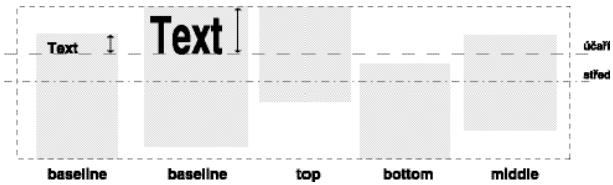
Výšku řádků ovlivňuje také svislé zarovnání buněk.

3.10.4.1 Svislé zarovnání buněk

Svislé zarovnání buňky na řádku určuje její vlastnost `line-height` [4.4.2.7]. V kontextu tabulek má však mírně odlišný význam než u ostatních prvků.

- `baseline` — účaří buňky je zarovnáno na účaří prvního řádku, do nějž buňka zasahuje (účaří viz níže).
- `top` — horní hrana buňky je zarovnána na horní hranu prvního řádku, do nějž buňka zasahuje
- `bottom` — dolní hrana buňky je zarovnána na dolní hranu posledního řádku, do nějž buňka zasahuje
- `middle` — střed buňky je zarovnán na střed všech řádků, do nichž buňka zasahuje
- `sub, super, text-top, text-bottom` — tyto vlastnosti nelze v tabulce použít; interpretují se jako `baseline`.

Účaří buňky je dáno účařím prvního řádkového rámu (účaří prvního textu v buňce). Pokud zde text není, použije se účaří kteréhokoli jiného objektu, případně se za účaří považuje spodní hrana buňky. Účaří řádku určuje největší vzdálenost mezi horní hranou a účařím všech buněk, které mají `vertical-align:baseline` (viz obrázek).



Obr. 41 — Svislé zarovnání buněk

Aby se zabránilo případným konfliktům při zarovnávání buněk, platí tato pravidla:

- Nejprve se zarovnají všechny buňky, které mají zarovnání `baseline`. Tím se vytvoří účaří řádku. Poté se zarovnají buňky s `top`. Tím má řádek definovánu svou horní hranu, případně účaří a má provizorně stanovenou výšku.
- Pokud mají některé z ostatních buněk definovanou výšku, která je větší než provizorní výška řádku, výška řádku se zvětší podle největší z těchto hodnot.
- Nakonec se zarovnají ostatní buňky (`bottom` a `middle`).

Buňky, které jsou nižší než výška řádku, získají dodatečnou horní/dolní výplň (padding).

3.10.4.2 Vodorovné zarovnání buněk

Vodorovné zarovnání obsahu buněk určuje vlastnost `text-align` [4.4.2.4]. Narozdíl od ostatních prvků lze v tabulkách použít i zarovnání na daný řetězec — `text-align:<řetězec>`.

Pokud mají alespoň dvě buňky ve stejném sloupci **zarovnání na stejný řetězec,** zarovná se jejich obsah podle společné svislé osy, začátek zadaného řetězec se dotýká této osy. Takový způsob zarovnání má ale smysl pouze pro jednořádkové buňky. Pokud bude buňka obsahovat více řádek textu, výsledek není možné odhadnout.

Příklad:

```
td { text-align: ',' }
td.email { text-align: '@' }
...
<table>
<tr><td>152,60</td><td class="email">ja@mojefirma.cz</td></tr>
<tr><td>2,-</td><td class="email">jan.novak@novak.cz</td></tr>
<tr><td>17,635</td><td class="email">pixy@pixy.cz</td></tr>
<tr><td>3.123,4567</td><td class="email">a@b.cz</td></tr>
</table>
```

Obsah buněk bude zarovnán takto:

```
       152,60                 x@mojefirma.cz
         2,-           jan.novak@novak.cz
        17,635              pixy@pixy.cz
     3.123,4567                a@b.cz
```

3.10.4.3 Dynamické řádkové a sloupcové efekty

Řádky, skupiny řádků, sloupce a skupiny sloupců mohou používat vlastnost `visibility` s hodnotou `collapse`. Ta způsobí, že část tabulky tvořená těmito řádky nebo sloupci se nezobrazí a o jejich rozměr se celá tabulka zmenší.

Samo o sobě by to nemělo velký význam, ale je zvlášť výhodné nastavovat viditelnost dynamicky, např. Javascriptem. Změna této hodnoty nijak neovlivňuje formátování tabulky, a můžeme tak dosáhnout např. rychlých změn pohledů na rozsáhlejší tabulky bez nutnosti jejich přeformátování.

3.10.5 Rámečky v tabulkách

V CSS existují dva samostatné modely pro formátování rámečků v tabulkách. První z nich orámuje každou buňku tabulky samostatně (**oddělený model**), používá se obvykle jako výchozí formátování tabulek v HTML bez použití CSS. Druhý bere na zřetel i okolní prvky a spojuje sousední rámečky do jediné čáry (**slučovací model**), buňky se rámují obdobně jako v tabulkových procesorech (např. MS Excel). To, který model se použije, určuje vlastnost tabulky `border-collapse` [4.8.3.1] — s hodnotou `separate` se použije *oddělený model,* s hodnotou `collapse` *model slučovací.*

3.10.5.1 Oddělený model rámování

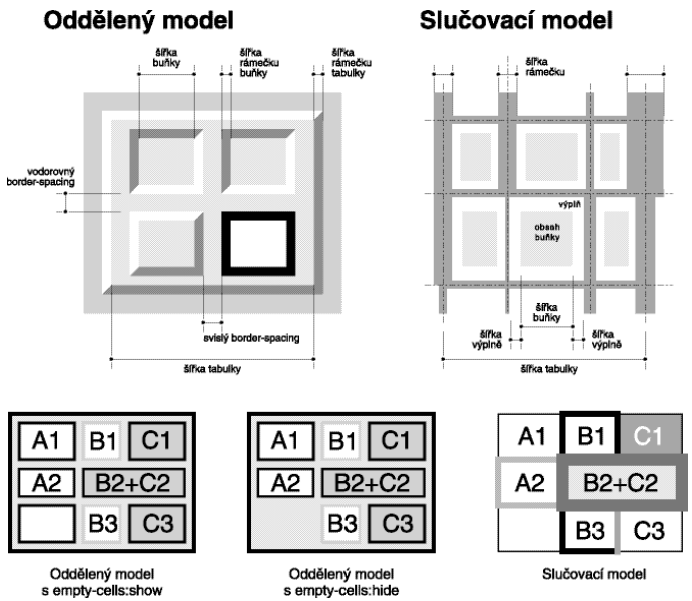
V *odděleném modelu* má každá buňka svůj vlastní rámeček. Řádky, sloupce či jejich skupiny nemohou být orámovány a jejich rámečky se ignorují. Buňky se nemusí navzájem dotýkat, odstup mezi nimi řídí vlastnost `border-spacing` [4.8.3.2], prázdný prostor mezi buňkami je vyplněn pozadím tabulky.

Způsob zobrazování prázdných buněk řídí vlastnost `empty-cells` [4.8.3.3]. Pokud má hodnotu `hide`, v prázdných a neviditelných buňkách se nezobrazí pozadí ani rámeček; s hodnotou `show` jsou formátovány jako ostatní buňky.

3.10.5.2 Slučovací model rámování

Ve *slučovacím modelu* jsou prvky tabulky rámovány v kontextu. Je možné použít rámečky i pro řádky, sloupce a jejich skupiny. Buňky se navzájem dotýkají, mají společné sousední hrany (vlastnosti `border-spacing` a `empty-cells` zde nemají význam).

Hrany buněk tak tvoří síť, na niž se umístí **osy rámečků.** Prohlížeč musí použít algoritmus, který šířky rámečků vhodně zaokrouhlí, aby byly na společné ose a přitom na sebe plynule navazovaly. V tomto modelu se do šířky tabulky započítává polovina šířky jejího rámečku a tabulka nemá výplň (pouze rámeček a okraje).



Obr. 42 — Dva modely rámování v tabulkách

Na každé hraně buňky se může sejít několik rámečků (rámeček buňky, řádku, sloupce, skupiny řádků či sloupců a rámeček tabulky), které se vždy mají spojit do rámečku jediného. V případech, kdy se rámečky na jedné hraně navzájem liší, řídí se prohlížeč následujícími pravidly:

- Rámeček se stylem `hidden` má vždy přednost. Pokud *kterýkoli prvek* tabulky na dané hraně předepisuje `border-style:hidden`, rámeček zde nebude zobrazen. Rámečky s `border-style:none` mají naopak nejnižší prioritu — rámeček nebude zobrazen, jen pokud mají tuto hodnotu definovány *všechny prvky,* které se hrany dotýkají.
- V ostatních případech mají přednost silnější rámečky před tenčími. Zobrazí se ten rámeček, který má větší hodnotu `border-width`. Má-li více prvků stejně silný rámeček, použije se rámeček se stylem nejvyšší priority. Styly jsou seřazeny od nejvyšší po nejnižší prioritu takto: `double`, `solid`, `dashed`, `dotted`, `ridge`, `outset`, `groove` a `inset`.
- Pokud se více rámečků liší pouze barvou, zobrazí se rámeček konkrétnějšího prvku — prvky jsou seřazeny takto: buňka (nejvyšší priorita), řádek, skupina řádků, sloupec, skupina sloupců a tabulka (nejnižší priorita). Rámečky buněk tak mají přednost rámečky řádku atd.

Pozn.: Styly rámečků `inset` a `outset` se používají v každém z obou modelů odlišně. V *odděleném modelu* mají stejný význam jako u jiných prvků (buňka je zobrazena, jako by byla celá zapuštěna, resp. vystupovala z tabulky). Ve slučovacím modelu se zobrazí stejně jako styly `groove`, resp. `ridge`.

3.11 Formátování stránek

Na *plynulých médiích* (např. na obrazovce počítače) je dokument zobrazen na jediné, neomezeně velké ploše, do níž prohlížeč uživateli poskytuje pohyblivý *průzor*. Na *stránkovaných médiích* (např. při výstupu dokumentu na tiskárnu) je však dostupná plocha omezena velikostí tohoto média (např. rozměrem papíru).

Dokument je zde zformátován na jednu či více samostatných stránek. Obsah je v určitém místě rozdělen a následující obsah je zformátován na nové stránce. CSS předepisuje postupy, kterými by se klient měl při rozmísťování obsahu na stránky řídit. Autor má navíc možnost doplnit tato pravidla o řídicí příkazy zakazující či naopak vynucující odstránkování před či za určitým prvkem dokumentu.

Specifikace CSS 2 původně obsahovala více nástrojů pro správu tisku (direktivu `@page`, pravidla pro okraje stránek atd.). V současné době je však podporuje pouze minimum prohlížečů (z těch nejpoužívanejších jen Opera) a specifikací CSS 2.1 byly opět odstraněny. Zůstaly zde pouze vlastnosti sloužící k řízení stránkových zlomů — teprve připravovaná norma CSS 3 nabídne samostatný modul pro komplexnější řešení správy stránek a tisku.

3.11.1 Řízení stránkování v CSS

Autoři mají k dispozici vlastnosti `page-break-before`, `page-break-after` a `page-break-inside`. Jejich hodnotou upřesňují, zda se může vyskytnout přechod na novou stránku před/za prvkem nebo uvnitř něj. Klient pak při formátování tyto hodnoty zohledňuje v níže uvedených pravidlech pro stránkování.

–> Vlastnosti page-break [4.9]

3.11.2 Pravidla pro stránkování

CSS nenařizuje žádná pravidla pro stránkování, ani nezakazuje, aby klient stránkoval kdekoli či dokonce nestránkoval vůbec. Pouze *doporučuje* dodržování následujících pravidel.

3.11.2.1 Povolené stránkování

V normálním toku stránky se může odstránkování vyskytnout na těchto místech:

- Ve svislém okraji mezi bloky. V tom případě se vypočítaná hodnota ovlivněných okrajů `margin-top` a `margin-bottom` nastaví na hodnotu 0.
- Mezi řádkami uvnitř bloku.

Platí přitom tato pravidla:

- **Pravidlo A** — zlom mezi bloky je povolen jen pokud to dovolují vlastnosti `page-break-after` a `page-break-before` všech prvků, které se na tomto okraji setkávají (tj. všechny mají hodnotu `auto`, nebo alespoň jeden z nich má hodnotu `always`, `right` nebo `left`).
- **Pravidlo B** — i když mají všechny tyto prvky hodnotu `auto`, ale jejich rodičovský prvek má hodnotu `page-break-inside:avoid`, stránkování je zde zakázáno
- **Pravidlo C** — odstránkování mezi řádky je v bloku povoleno, jen pokud má `page-break-inside:auto`.

Pokud tato pravidla neposkytují dostatek míst k odstránkování, aby se obsah vešel na stránku, klient vypustí pravidla B a C a pokusí se najít další místa vhodná pro zalomení. Pokud to stále nepostačí, vypustí se i pravidlo A.

3.11.2.2 Vynucené stránkování

Stránkový zlom se musí provést mezi bloky, pokud se mezi vlastnostmi `page-break-after` a `page-break-before` všech prvků, které se zde stýkají, vyskytne alespoň jedna hodnota `always`, `left` nebo `right` (je zde prvek, který před či za sebou vyžaduje odstránkování).

3.11.2.3 Optimální stránkování

Klient by měl navíc zohledňovat i následující doporučení:

- Stránkuj co nejméně
- Snaž se, aby stránky (nekončí-li vynuceným odstránkováním) měly přibližně stejnou výšku
- Nestránkuj v blocích, které mají rámeček
- Nestránkuj uvnitř tabulky
- Nestránkuj uvnitř plovoucího prvku

3.12 Zvukové styly

Dokumenty nemusí být formátovány pouze vizuálně, stále častější je také jejich zvuková prezentace. Zvukový výstup vyžívali takřka výhradně zrakově postižení uživatelé, pro něž je obvykle jedinou cestou k informacím na webu. Stále častěji se však můžeme setkat se zvukovou prezentací dokumentů i v mnoha dalších oblastech — ve čtecích zařízeních v automobilech, v telekomunikačních hlasových službách (např. předčítání zpravodajství přes telefon), domácí hlasový výstup (počítač čte obsah vašeho oblíbeného webu, zatímco si chystáte snídani), průmyslové či lékařské informační systémy atd. Lze očekávat, že v budoucnu se bude uplatnění zvukových prezentací stále rozšiřovat.

Nejčastějším a nejjednodušším způsobem zvukového zpracování dokumentů je jejich převod do prostého textu, který následně zpracuje *čtečka obrazovky* — tedy program, který jednoduše „přečte“ všechny znaky na obrazovce. To však může způsobit nekorektní prezentaci, především pokud je obsah dokumentu uspořádán ve více sloupcích.

Kaskádové styly poskytují sofistikovanější nástroje pro formátování dokumentů na zvukovém výstupu: celá stránka i každý její prvek mohou mít definovány vlastnosti CSS, které určují, jak bude prvek a jeho obsah zvukově prezentován.

Z hlediska použití zvukových vlastností CSS představuje dokument trojrozměrný prostor, v němž jsou umístěny *zdroje,* ze kterých vychází zvuk prezentující obsah jednotlivých prvků. Pro každý prvek pak může autor nastavit pozici zdroje zvuku, typ a vlastnosti hlasu, který předčítá textový obsah, jeho rychlost i hlasitost, zvuky, které mají zaznít před začátkem či po skončení čtení textu atd.

—> Přehled zvukových vlastností a jejich popis [4.10]

4 Přehled funkcí a vlastností CSS

4.1 Legenda

4.1.1 Popis vlastností CSS

V této kapitole najdete přehled všech vlastností CSS. Vlastnosti jsou uvedeny v pořadí, v němž jsou zmíněny v předchozím textu. V jednom bloku je vždy společně celá skupina vlastností, které k sobě logicky patří. U každé z nich je uveden příklad použití v *pravidlech* CSS a úroveň podpory v běžných prohlížečích.

4.1.1.1 Legenda k popisu vlastností

Popis každé vlastnosti či skupiny vlastností je uveden v samostatné tabulce.

Ve sloupci **vlastnost** je název vlastnosti. Ve sloupci **hodnota** jsou uvedeny povolené hodnoty, kterých vlastnost může nabývat (význam symbolických zápisů syntaxe byl popsán již dříve [3.2.1]). Špičatými závorkami jsou označeny *typy hodnot CSS* [3.3] — např. pro typ `<číslo>` může být hodnota `10`, `1.35` či `-17`. Typu `<velikost>` odpovídají hodnoty ve tvaru `<číslo>jednotka`, např. `1em`, `10px`, `3.5cm` atd.

Tučně zvýrazněné typy slouží jako *zkratky* pro popis dalších vlastností. Např. možné hodnoty okrajů jsou `<velikost>|<procenta>|auto`. Tento typ je označen jako **<šířka_okraje>** a pozdější zkrácený zápis **<šířka_okraje>**{1,4} znamená totéž jako `[<velikost>|<procenta>|auto]{1,4}`.

Ostatní položky již popisují konkrétní možné hodnoty — klíčová slova (`auto`, `none`), čísla (`0`), řetězce (`'text'`) atd. Kromě uvedených hodnot může každá vlastnost nabývat i hodnoty `inherit` (použije se *vypočítaná hodnota* z *rodičovského prvku*).

Ve sloupci **výchozí hodnota** je hodnota, která se použije, pokud vlastnost není explicitně definována jinak a její hodnotu nelze zdědit. Pole **použitelnost** udává prvky či typ prvků, pro něž daná vlastnost způsobí nějaký efekt. **Dědičnost** určuje, zda se hodnota vlastnosti dědí z rodičovského prvku. Pole **procenta** říká, k čemu jsou vztaženy procentní hodnoty, pokud je lze v této vlastnosti použít. Ve sloupci **média** je uvedena skupina médií, na nichž se daná vlastnost používá.

4.1.2 Kompatibilita CSS v prohlížečích

Protože se podpora různých funkcí CSS v prohlížečích výrazně liší, kromě popisu vlastností zde nejdete i tabulky s přehledem úrovně jejich implementace v nejpoužívanějších či nejznámějších prohlížečích. Ve zvýrazněném řádku je celková podpora dané funkce, následují informace pro jednotlivé dílčí vlastnosti. Tam, kde je třeba podrobnější vysvětlení, je uvedeno číslo poznámky s upřesňujícími informacemi.

V tabulkách jsou pro přehlednost použity zkrácené názvy prohlížečů a symboly označující podporu dané funkce v konkrétním prohlížeči. Následující legenda podává jejich podrobnější popis. Prohlížeče, které v době přípravy této knihy ještě nebyly dostupné ve finální verzi, nebyly plně testovány.

4.1.2.1 Legenda k tabulkám kompatibility

Použité prohlížeče:

MSIE Windows:
- **4** MS Internet Explorer 4.0 pro Windows
- **5** MS Internet Explorer 5.0 pro Windows
- **5.5** MS Internet Explorer 5.5 pro Windows
- **6** MS Internet Explorer 6.0 pro Windows

MSIE Mac:
- **4** MS Internet Explorer 4.5 pro MacOS
- **5** MS Internet Explorer 5.0 pro MacOS

NN/Moz:
- **4** Netscape Navigator 4.x (všechny platformy, případné odlišnosti mohou být upřesněny v poznámkách)
- **6** Netscape 6 / Mozilla 1 (všechny platformy)

Opera:
- **4** Opera 4.02 pro Windows
- **5** Opera 5 pro Windows

Symboly pro úroveň podpory vlastností:

- **++** prohlížeč danou funkci podporuje
- **+** prohlížeč funkci podporuje, ale ne zcela podle standardu (po svém)
- **+/-** prohlížeč funkci podporuje pouze zčásti
- **---** prohlížeč funkci nepodporuje
- **!!!** prohlížeč funkci zpracovává chybně

4.2 Obecné funkce CSS

Vkládání CSS	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
Vložení stylů do HTML											
<link rel="stylesheet"...>	++	++	++	++	++	++	++	++	++	++	> Většina prohlížečů nepodporuje alternativní styly <link rel="alternate stylesheet">. Opera je sice nabízí v menu Links, ale místo použití pouze zobrazí obsah souboru. NN6/Mozilla nalezené altrenativní styly nabízí a používá korektně.
<style>...</style>	++	++	++	++	++	++	++	++	++	++	
@import	+/-	+	+	++	+/-	++	--	++	++	++	> MSIE/Win 4-5 (a MSIE 6 mimo striktní režim) načte soubor, i když u @import není na začátku. MSIE 4 však ignoruje syntaxi @import "..." a podporuje pouze @import url(...) — což se často využívá pro jeho odříznutí od stylů.
@media	--	--	++	++	--	--	--	++	--	++	
<prvek style="...">	++	++	++	++	++	++	++	++	++	++	
Komentáře											
Komentáře CSS	!!!	++	++	++	++	++	++	++	++	++	
Komentáře HTML	++	++	++	++	++	++	++	++	++	++	

Selektory	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
Základní selektory											
Univerzální selektor (*)	--	++	++	++	++	++	--	++	++	++	
Selektory typu (A)	++	++	++	++	++	++	++	++	++	++	
Selektory pro ROW a COLUMN	--	--	--	--	--	--	--	--	--	--	> Formátování řádek a sloupců tabulek zatím není nikde podporováno.
Selektory s atributy											
A[atr]	--	--	--	--	--	--	--	++	++	++	
A[atr=hodn]	--	--	--	--	--	--	--	++	--	--	
A[atr~=hodn]	--	--	--	--	--	--	--	++	++	++	
A[atr|=hodn]	--	--	--	--	--	--	--	++	!!!	!!!	
Selektory tříd a ID											
Selektory tříd (.trida)	+/-	++	++	++	+/-	++	+/-	++	++	++	> IE5.x (Win i Mac) chybně interpretuje vícenásobné třídy a použije styl, i když je v prvku definována jen jedna ze tříd. Např. h1.nadpis.modry chybně vyhovuje i <h1 class="nadpis">. IE4-5/Win chybně akceptuje i názvy tříd začínající číslicemi.
Selektory ID (#prvek)	++	++	++	++	++	++	++	++	++	++	> Chyby jsou spíše v implementaci HTML. IE4 akceptuje i ID začínající číslicí, IE/Win do verze 6 nerozlišuje velikost písmen ve striktním HTML4/XHTML (#prvek = #PRVEK). Všechny prohlížeče chybně dovolují, aby více prvků používalo stejné ID.
Kombinování selektorů											
Následnické selektory (A B C)	++	++	++	++	++	++	!!!	++	++	++	> NN4/Mac chybně zpracuje kontextové selektory obsahující tabulky
Selektor potomka (A>B)	!!!	!!!	--	---	--	++	--	++	++	++	> IE4-5/Win chybně interpretuje A > B i A + B jako A B — „div > p“, „div + p“ a „div p“ je pro něj totéž
Sousední sourozenec (A+B)	!!!	!!!	--	--	---	--	--	++	++	++	
Slučování selektorů (A,B,C)	++	++	++	++	++	++	!!!	++	++	++	
Pseudo-třídy											
:first-child	--	--	--	--	--	++	--	++	--	--	
:link	++	++	++	++	++	++	--	++	++	++	
:visited	++	++	++	++	++	++	--	++	--	++	
:hover	+/-	+/-	+/-	+/-	+/-	+/-	--	++	--	+/-	> Prohlížeče vesměs umí :hover pouze u odkazů, ačkoli jeho použití není v CSS nijak omezeno. Jen NN6/Moz zpracuje :hover i s jinými prvky.
:focus	--	--	--	--	--	++	--	++	--	--	> V IE5/Mac lze měnit background, ale třeba width nebo height ne (což je ale stále korektní chování). V NN6/Moz je velmi podivný :focus na prvku select.
:active	+	+	+	+	++	++	--	++	+/-	++	> IE/Win ponechá :active až do chvíle, než uživatel klepne jinam - přitom by měl tento stav trvat pouze mezi stisknutím a puštěním tlačítka myši.
:lang	--	--	--	--	--	++	--	--	--	--	
Kombinace :pseudo-třída1:pseudo-třída2	--	--	--	--	--	++	--	++	--	--	
Pseudo-prvky											
:first-line	--	--	++	++	--	++	--	++	++	++	
:first-letter	--	--	++	++	--	++	--	++	++	++	
:before / :after	--	--	--	--	--	--	--	++	!!!	++	> Tyto selektory mají smysl pouze tehdy, podporuje-li prohlížeč generovaný obsah, tedy především vlastnost content.

Dědičnost a kaskáda	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
Dědičnost	++	++	++	++	++	++	!!!	++	++	++	> V NN4 je chyba na chybě. Z hlediska kaskády a hlavně dědičnosti je prakticky nepoužitelný. Nejproblematičtějšími jsou především tabulky a seznamy. Je nutné počítat s tím, že jejich obsah nezdědí žádnou hodnotu od rodičovského prvku.
Definice s !important	++	++	++	++	--	++	--	++	++	++	
Pořadí kaskády							!!!				
Třídění podle váhy	++	++	++	++	++	++	!!!	++	++	++	> NN4 — opět chyba na chybě, porušuje prakticky všechna pravidla.
Třídění podle původu	++	++	++	++	++	++	!!!	++	++	++	
Třídění podle specifičnosti	++	++	++	++	++	++	!!!	++	++	++	
Třídění podle pořadí	++	++	++	++	++	++	!!!	++	++	++	

Hodnoty a jednotky	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
<velikost>											> IE/Win chybně akceptují hodnotu bez jednotky, ta je pak interpretována jako by zde bylo px. IE6 ve standardním režimu to již netoleruje.
em	++	++	++	++	++	++	++	++	++	++	
ex	+	+	++	++	+	++	+	++	++	++	> Všechny prohlížeče zřejmě interpretují ex jako polovinu em. Je to možná použitelná náhrada, ale je to špatně.
px	++	++	++	++	++	++	++	++	++	++	
in	++	++	++	++	++	++	++	++	++	++	
cm	++	++	++	++	++	++	++	++	++	++	
mm	++	++	++	++	++	++	++	++	++	++	
pt	++	++	++	++	++	++	++	++	++	++	
pc	++	++	++	++	++	++	++	++	++	++	
<procenta>											
%	++	++	++	++	++	++	++	++	++	++	
<barva>											
#000	++	++	++	++	++	++	++	++	++	++	> IE/Win chybně akceptují barevné hodnoty i bez znaku #.IE6 ve standardním režimu to již netoleruje.
#000000	++	++	++	++	++	++	++	++	++	++	
(RRR,GGG,BBB)	++	++	++	++	++	++	++	++	++	++	
(R%,G%,B%)	++	++	++	++	++	++	++	++	++	++	
<klíčové slovo>	++	++	++	++	++	++	!!!	++	++	++	> NN4 vygeneruje barvu pro libovolné (i neexistující) klíčové slovo. Např. color: samavoda nastaví modrou barvu, NN „přeloží" dokonce color: inherit (jako odstín zelené).
<url>											
<url>	++	++	++	++	++	++	!!!	++	++	++	> NN4 vztahuje chybně relativní URL k umístění dokumentu, namísto k umístění tabulky CSS.
<řetězec>											
<řetězec>	!!!	!!!	!!!	++	!!!	+	++	++	++	++	> IE4-5/Win a IE4/Mac nepodporují korektně escape-sekvence v řetězcích. IE5/Mac pracuje korektně, jen pokud je text uzavřen v "…" — např. url("60\'s.gif") funguje, ale url('60\'s.gif') ne. IE6 už má plnou podporu, včetně kódů Unicode.

4.3 Vlastnosti rámů

4.3.1 Okraje — margin

Skupina vlastností **margin** definuje (vždy průhledné) *okraje* kolem rámu prvku [3.8.3]. V některých případech se okraje sousedních prvků slučují (viz [3.8.6.2]). Procentní hodnoty se vždy vztahují **k šířce** *omezujícího bloku* [3.8.4] (a to i u vertikálních okrajů!). Z hodnoty `auto` se šířka okraje vypočítává příslušným algoritmem [3.8.7.1].

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
border-top-width, border-right-width, border-bottom-width, border-left-width	thin | medium | thick | <velikost> (= **<šířka_rámečku>**)	medium	nelze	ne	všechny	vizuální
border-width (sdružená vlastnost)	**<šířka_rámečku>**{1,4}	viz dílčí vlastnosti border-top-width až border-left-width				
border-top-color, border-right-color, border-bottom-color, border-left-color	<barva> | transparent (= **<barva_rámečku>**)	hodnota vlastn. **color**	nelze	ne	všechny	vizuální
border-color (sdružená vlastnost)	**<barva_rámečku>**{1,4}	viz dílčí vlastnosti border-top-color až border-left-color				
border-top-style, border-right-style, border-bottom-style, border-left-style	none | hidden | solid | double | dotted | dashed | groove | ridge | inset | outset (= **<styl_rámečku>**)	none	nelze	ne	všechny	vizuální
border-style (sdružená vlastnost)	**<styl_rámečku>**{1,4}	viz dílčí vlastnosti border-top-style až border-left-style				
border-top, border-right, border-bottom, border-left, border (sdružené vlastnosti)	**<šířka_rámečku>** || **<barva_rámečku>** || **<styl_rámečku>**	viz dílčí vlastnosti				

Sdružená vlastnost `margin` definuje všechny čtyři okraje současně pomocí jedné až čtyř hodnot v pořadí top—right—bottom—left (shora ve směru hodinových ručiček). Je-li hodnot méně než čtyři, hodnota chybějícího okraje je stejná jako okraje protějšího. Hodnota jediná definuje všechny čtyři okraje stejné.

margin: A B C D = po řadě horní (top), pravý (right), dolní (bottom) a levý (left) okraj
margin: A = margin: A A A A
margin: A B = margin: A B A B
margin: A B C = margin: A B C B

U *nenahrazovaných řádkových prvků* se svislé okraje ignorují. Okraje smí být i **záporné** [3.8.3], ale jejich formátování může být limitováno možnostmi prohlížeče.

```
p { margin-top: 10px }
div { margin: 10mm 10% auto 0 }
```

Vlastnosti rámů — okraje	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
margin-top											
<velikost>	+/-	+/-	++	++	+/-	++	+/-	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	+/-	++	++	++	
auto	+/-	+/-	++	++	+/-	++	+/-	++	++	++	
margin-right							!!!		!!!		> Opera4 *někdy* aplikuje v řádkových prvcích pravý okraj na všechny ostatní strany.
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
auto	--	--	++	++	+/-	++	--	++	++	++	
margin-bottom											
<velikost>	+/-	+/-	++	++	+/-	++	--	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	--	++	++	++	> NN4 spodní okraje nepodporuje, ale nejsou nulové. Mezera mezi prvky je tak zcela nepředvídatelná.
auto	+/-	+/-	++	++	+/-	++	--	++	++	++	
margin-left							!!!		!!!		> Opera4 *někdy* aplikuje v řádkových prvcích pravý okraj na všechny ostatní strany.
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	> Podpora v NN4/Mac je poněkud lepší.
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
auto	--	--	++	++	+/-	++	--	++	++	++	
margin							!!!		!!!		
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	> Všechny vlastnosti okrajů řádkových prvků jsou problematické, IE4-5/Win je nepodporuje vůbec. Podivné problémy Opery viz výše. NN4 pracuje s okraji řádkových a plovoucích prvků velmi nešťastně — u plovoucích prvků mohou zcela „zbořit" stránku. U blokových prvků jsou okraje počítány obvykle korektně (kromě NN4).
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
auto	+/-	+/-	++	++	+/-	++	--	++	++	++	

4.3.2 Výplně — padding

Skupina vlastností **padding** definuje *výplňovou oblast* mezi obsahem prvku a jeho rámečkem [3.8.3]. Stejně jako u okrajů se procentní hodnoty vodorovné i svislé výplně vztahují **k šířce** omezujícího bloku. Sdružená vlastnost `padding` definuje výplň ve všech čtyřech směrech současně (viz `margin` výše). Záporné hodnoty nejsou povoleny.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
padding-top, padding-right, padding-bottom, padding-left	<velikost> | <procenta> (= **<šířka_výplně>**)	0	vždy relativně **k šířce** omezujícího bloku	ne	všechny	vizuální
padding (sdružená vlastnost)	**<šířka_výplně>**{1,4}	viz dílčí vlastnosti padding-top až padding-left				

```
p { padding-top: 1em }
div { padding: 10pt 10% 0 2mm }
```

Vlastnosti rámů — výplň	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
padding-top							!!!				
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
padding-right											
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
padding-bottom							!!!				
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	> Stejně jako u okrajů, nějvětší problémy jsou s výplní řádkových prvků. IE4-5/Win je nepodporuje vůbec. NN4 má potíže stejné jako s okraji — u plovoucích a řádkových prvků mohou se stránkou provést velmi zvláštní věci.
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
padding-left							!!!				
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
padding							!!!				
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<procenta>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	

4.3.3 Rámečky — border

Skupina vlastností **border** definuje šířku, barvu a styl *rámečku* kolem prvku [3.8.3]. Prohlížeče by měly ignorovat rámeček pro prvek `html`. Rámečky některých prvků (především tlačítek a dalších prvků formulářů) mohou být zobrazeny jiným způsobem, podle zvyklostí *uživatelského prostředí* (např. zaoblené, s trojrozměrnými efekty atd.).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
border-top-width, border-right-width, border-bottom-width, border-left-width	`thin` | `medium` | `thick` | `<velikost>` (= **<šířka_rámečku>**)	`medium`	nelze	ne	všechny	vizuální
border-width (sdružená vlastnost)	**<šířka_rámečku>**{1,4}	viz dílčí vlastnosti border-top-width až border-left-width				
border-top-color, border-right-color, border-bottom-color, border-left-color	`<barva>` | `transparent` (= **<barva_rámečku>**)	hodnota vlastn. **color**	nelze	ne	všechny	vizuální
border-color (sdružená vlastnost)	**<barva_rámečku>**{1,4}	viz dílčí vlastnosti border-top-color až border-left-color				
border-top-style, border-right-style, border-bottom-style, border-left-style	`none` | `hidden` | `solid` | `double` | `dotted` | `dashed` | `groove` | `ridge` | `inset` | `outset` (= **<styl_rámečku>**)	`none`	nelze	ne	všechny	vizuální
border-style (sdružená vlastnost)	**<styl_rámečku>**{1,4}	viz dílčí vlastnosti border-top-style až border-left-style				
border-top, border-right, border-bottom, border-left, border (sdružené vlastnosti)	**<šířka_rámečku>** || **<barva_rámečku>** || **<styl_rámečku>**	viz dílčí vlastnosti				

Pro **šířku rámečku** se používají klíčová slova `thin` (tenký rámeček), `medium` (střední) a `thick` (silný). Konkrétní šířka takto zadaného rámečku už závisí na prohlížeči — CSS pouze určuje, že rámeček `thin` nesmí být širší než `medium` a ten zase širší než `thick`. Tyto rozměry musí být stejné v celém dokumentu. Přesnou šířku rámečku určíme hodnotou `<velikost>`. *Sdružená vlastnost* `border-width` definuje současně šířku všech čtyř stran rámečku — stejně jako např. `margin` [4.3.1].

```
p { border-top-width: thick }
div { border-width: thin medium 5px thick }
```

Barvu rámečku určíme hodnotou typu `<barva>`, hodnota `transparent` způsobí, že rámeček bude průhledný (stále ale může mít nenulovou šířku). Není-li barva zadána, jako *vypočítaná hodnota* se použije hodnota vlastnosti `color` [4.5.1]. *Sdružená vlastnost* `border-color` opět definuje současně barvu všech čtyř stran rámečku.

```
p { border-top-color: red }
div { border-color: red #FFC090 transparent rgb(30,64,180) }
```

Pro **styl rámečku** se používají uvedená klíčová slova s následujícím významem:

`none` — žádný rámeček (šířka rámečku je současně nastavena na hodnotu 0).
`hidden` — stejně jako `none`, rozdíl je pouze při řešení konfliktních rámečků u tabulek [3.10.5.2]

`solid` — rámeček je tvořen plnou čarou
`double` — rámeček je tvořen dvěma plnými čarami, šířka rámečku odpovídá součtu jejich šířek a mezery mezi nimi
`dashed` — rámeček tvoří řada krátkých úseků čáry (čárkovaná čára)
`dotted` — rámeček tvoří řada teček (tečkovaná čára)
`groove` — rámeček je zobrazen s 3D efektem, jako by byl vyryt do pozadí prvku
`ridge` — opak `groove`; rámeček vystupuje nad pozadí
`inset` — rámeček je vykreslen tak, že celý prvek vypadá jako zapuštěný do pozadí
`outset` — opak `inset`; rámeček je vykreslen tak, že celý prvek zdánlivě vystupuje nad pozadí

Rámečky typu `solid`, `double`, `dashed` a `dotted` jsou vykresleny jedinou barvou, pro rámečky `groove`, `ridge`, `inset` a `outset` může prohlížeč použít i další barvy, aby dosáhl příslušného 3D efektu. Základem je ale vždy barva definovaná pro rámeček hodnotou `border-color`. Styly `inset` a `outset` mají mírně odlišné chování v tabulkách (viz [3.10.5]). Prohlížeč může podporovat pouze typ `solid` a rámečky všech následujících stylů zobrazovat jen plnou čarou. *Sdružená vlastnost* `border-style` definuje styl všech stran rámečku současně.

```
p { border-top-style: solid }
div { border-style: groove dotted ridge none }
```

Sdružené vlastnosti `border-top` až `border-left` nastavují současně šířku, barvu a styl rámečku. Můžeme zadat všechny tři údaje v libovolném pořadí nebo některý z nich vynechat. Vynechaná podvlastnost se považuje za *nedefinovanou* a nabývá své *výchozí hodnoty*.

```
p { border-top: 1px solid red }
div { border-right: dotted 2pt }
```

je totéž jako:

```
p {
      border-top-width: 1px;
      border-top-style: solid;
      border-top-color: red
}
div {
      border-top-style: dotted;
      border-top-width: 2pt
      /* border-top-color není definována,
      použije se hodnota vlastnosti color */
}
```

Sdruženou vlastností `border` se určuje současně šířka, barva a styl rámečku stejná pro všechny čtyři strany. Vlastností `border` není možné nastavit stranám rámečku různé hodnoty.

```
p { border: outset silver }
```

je totéž jako:

```
p {
      border-top: outset silver;
      border-right: outset silver;
      border-bottom: outset silver;
      border-left: outset silver
}
```

Vlastnosti rámů — rámečky	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
border-top-width, border-right-width, border-bottom-width, border-left-width, border-width											Obdobně jako u vlastností margin a padding nepodporuje IE4-5/Win rámečky řádkových prvků. U blokových je ale zpracování rámečků korektní.
thin	+/-	+/-	++	++	+/-	++	!!!	++	++	++	> NN4 zobrazí všechny rámečky stejné, pokud je nějak definován border-style. Naopak zobrazí rámeček jako solid, i když border-style není definován. Strany rámečku různých šířek na sebe nenavazují. Řádkové prvky s rámečkem jsou zobrazeny jako bloky.
medium	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
thick	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<velikost>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
border-color											> NN4 seznam více hodnot nepoužije jako definice pro jednotlivé strany rámečku, ale „přeloží" si je jako jediné klíčové slovo (viz poznámka u hodnot a jednotek <barva>).
<barva>	++	++	++	++	++	++	+/-	++	++	++	
border-style											
none	++	++	++	++	++	++	++	++	++	++	> NN4 nevynuluje šířku rámečku, je-li border-style:none a použije šířku z border-width.
dotted	--	--	--	--	++	++	--	++	++	++	> IE6/Win umí jen čárkovanou čáru, rámečky dotted zobrazí stejně jako dashed. IE4-5/Win v obou případech použijí čáru plnou.
dashed	--	--	--	++	++	++	--	++	++	++	
solid	++	++	++	++	++	++	++	++	++	++	
double	++	++	++	++	++	++	++	++	++	++	
groove	++	++	++	++	++	++	++	++	++	++	
ridge	++	++	++	++	++	++	++	++	++	++	
inset	++	++	++	++	++	++	++	++	++	++	
outset	++	++	++	++	++	++	++	++	++	++	
border-top, border-right, border-bottom, border-left											
<border-top-width>	+/-	+/-	++	++	+/-	++	--	++	++	++	> IE4-5 nepodporují rámečky u řádkových prvků.
<border-style>	+/-	+/-	++	++	+/-	++	--	++	++	++	
<barva>	+/-	+/-	++	++	+/-	++	--	++	++	++	
border										!!!	> Opera5 někdy (náhodně) špatně vykreslí rámeček prvního řádkového prvku v bloku.
<border-width>	+/-	+/-	++	++	+/-	++	!!!	++	++	++	
<border-style>	+/-	+/-	++	++	++	++	+/-	++	++	++	
<barva>	+/-	+/-	++	++	++	++	++	++	++	++	

4.3.4 Rozměry prvku

4.3.4.1 Šířka prvku — width, min-width, max-width

Vlastností **width** se definuje *šířka obsahu* blokových a nahrazovaných prvků (rozměr ostatních oblastí se do šířky nezapočítává). U nenahrazovaných řádkových prvků nemá `width` žádný význam, formátují se vždy podle dostupné šířky řádky. Nahrazované prvky mají svou *vnitřní šířku,* ale mají-li udánu hodnotu `width` jinou než `auto`, klient je může příslušně zvětšit/zmenšit.

Vlastnosti `min-width` a `max-width` omezí šířku prvku pouze na určité rozmezí.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
width	<velikost> | <procenta> | auto	auto	relativně k šířce omezujícího bloku	ne	všechny kromě řádkových nenahraz., řádek a skupin řádek tabulek	vizuální
min-width	<velikost> | <procenta>	závisí na prohlížeči			dtto + vůbec žádné prvky tabulek	
max-width	<velikost> | <procenta> | none	none				

Záporné hodnoty nejsou povoleny. Hodnota `<velikost>` specifikuje šířku prvku udaným rozměrem s příslušnou jednotkou. Hodnotou `<procenta>` určíme šířku prvku relativně k šířce

jeho *omezujícího bloku.* S hodnotou `auto` se provádí automatický výpočet šířky. Hodnota `none` u `max-width` říká, že maximální šířka není omezena.

–> Výpočty rozměrů prvků [3.8.7]

```
div.menu { width: 240px }
p { width: 80% }
h1 {
      width: auto;
      min-width: 100px;
      max-width: 90%
}
```

4.3.4.2 Výška prvku — height, min-height, max-height

Vlastností **height** se definuje výška obsahu blokových a nahrazovaných prvků (rozměr ostatních prvků se do výšky nezapočítává). U nenahrazovaných řádkových prvků se ignoruje, jejich výška se řídí výškou řádky (vlastnost `line-height`). Nahrazované prvky mají svou *vnitřní výšku,* ale mají-li udánu hodnotu `height` jinou než `auto`, klient je může příslušně zvětšit/zmenšit.

Vlastnosti `min-width` a `max-width` omezí šířku prvku pouze na určité rozmezí.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
height	`<velikost>` | `<procenta>` | auto	auto	relativně k výšce omezujícího bloku (viz text)	ne	všechny kromě řádkových nenahraz., sloupců a skupin sloupců tabulek	vizuální
min-height	`<velikost>` | `<procenta>`	0			dtto + vůbec žádné prvky tabulek	
max-height	`<velikost>` | `<procenta>` | none	none				

Záporné hodnoty nejsou povoleny. Hodnota `<velikost>` specifikuje výšku prvku udaným rozměrem s příslušnou jednotkou. Hodnotou `<procenta>` určíme výšku prvku relativně k výšce jeho *omezujícího bloku.* Pokud není výška tohoto omezujícího bloku explicitně udána (např. se přizpůsobuje velikosti obsahu), interpretuje se hodnota `height` zadaná procenty jako `auto`. Klient může procentní výšku kořenového prvku počítat relativně k výšce *průzoru* (např. okna prohlížeče, viz [3.8.1]), není to však jeho povinností. S hodnotou `auto` se provádí automatický výpočet výšky. Hodnota `none` u `max-height` říká, že maximální výška není omezena.

–> Výpočty rozměrů prvků [3.8.7]

```
div.menu { height: 24px }
p { height: 50% }
p.uvod {
      height: auto;
      min-height: 10em;
      max-height: none
}
```

4.3.4.3 Výpočetní model rozměrů rámů — box-sizing

Kvůli zásadnímu nepochopení specifikace CSS ze strany společnosti Microsoft vzniká při definování rozměrů prvků řada komplikací. Podle CSS totiž vlastnosti `width` a `height` popisují pouze *rozměry obsahu* prvku (tzv. **model W3C**). Prohlížeče MSIE ale do těchto rozměrů započítávají i oblasti `padding` a `border` (tzv. **model MSIE**). Dokumenty se proto zobrazují velmi rozdílně v prohlížečích MSIE ve Windows a ve všech ostatních, které dodržují specifikaci bezchybně (dlužno podotknout, že korektně již rozměry zpracovává i MSIE od verze 6, ovšem pouze ve *striktním režimu* [5.1.2]).

V nově připravované verzi CSS3 však W3C udělalo vstřícný krok a plánuje do CSS přidat novou vlastnost, která autorům umožňuje zvolit model, který se má použít. I když v době přípravy této knihy ještě specifikace CSS3 není dokončena, vlastnost **box-sizing** již teď podporuje MSIE 5+ na MacOS (který jinak zásadně používá *model W3C*). Prohlížeče NN6/Mozilla ji sice nepodporují, ale mají mezi svými interními vlastnostmi podobný nástroj: vlastnost `-moz-box-sizing`.

V obou případech se ale používají stejné hodnoty, které nastaví příslušný výpočetní model — s hodnotou `border-box` se použije *model MSIE*, s hodnotou `content-box` pak *model W3C*. Výchozí hodnotou je vždy `content-box`. Tyto vlastnosti se nedědí.

```
div.menu {
      box-sizing: border-box;
      -moz-box-sizing: border-box;
      padding: 10px;
      border: 5px solid red;
      width: 150px;
/* model MSIE, width udává šířku oblasti rámečku;
   šířka obsahu zde bude 120px */
}
```

Vlastnosti rámů — rozměry	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
width											> IE4-5/Win použijí šířku pro obrázky a tabulky, ale u většiny blokových prvků (p, h1-h4) ji ignorují. NN4 aplikuje hodnotu width jakýmsi nejasným způsobem.
<velikost>	+/-	+/-	++	++	++	++	+/-	++	++	++	> I když jsou rozměry prvkům přiřazeny korektně, je třeba počítat s rozdílným výpočetním modelem v prohlížečích (více viz jinde v této knize).
<procenta>	+/-	+/-	++	++	++	++	+/-	++	++	++	
auto	+/-	+/-	++	++	++	++	+/-	++	++	++	
height											> Podle W3C se za rozměr prvku považuje jen rozměr oblasti obsahu, model IE do něj počítá i výplně a rámeček. Model IE je použit ve všech IE (kromě IE6 ve standardním módu a IE4/Mac[!]), model W3C používá NN4, NN6/Moz, Opera, IE4/Mac a IE6 (std.mód). (viz také box-sizing)
<velikost>	++	++	++	++	++	++	--	++	++	++	
auto	++	++	++	++	++	++	--	++	++	++	
min-width	--	--	--	--	--	--	--	++	--	++	> Minimální rozměry zatím podporují je NN6/Mozz a Opera 5+. V IE6 funguje pouze min-height, a to jen za velmi specifických okolností (jen u tabulkových prvků uvnitř tabulky s table-layout: fixed).
max-width	--	--	--	--	--	--	--	++	--	++	
min-height	--	--	--	+/-	--	--	--	++	--	++	
max-width	--	--	--	--	--	--	--	++	--	++	
box-sizing	--	--	--	--	--	++	--	--	--	--	> V IE5/Mac a NN6/Moz lze vlastnosti box-sizing (převzatou z připravovaného CSS3) změnit počítání rozměrů na model IE. NN6/Moz však používá jiný, vlastní název vlastnosti.
-moz-box-sizing	--	--	--	--	--	--	--	++	--	--	

4.3.5 Typ a pozice prvku

Následující skupina vlastností definuje, jakým způsobem bude prvek formátován a určuje jeho umístění vzhledem k okolním prvkům.

–> Vizuální formátovací model [3.8]

4.3.5.1 Typ prvku — display

Vlastnost **display** určuje, jaký model formátování se pro prvek použije a jaké bude generovat rámy [3.8.5].

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
display	`inline | block | list-item | run-in | inline-block | table | inline-table | table-row-group | table-header-group | table-footer-group | table-row | table-column-group | table-column | table-cell | table-caption | none`	`inline`	nelze	ne	všechny	vizuální

Možné hodnoty vlastnosti `display` mají následující význam:

- `inline` — prvek je řádkový a generuje jeden či více řádkových rámů.
- `block` — prvek je blokový a generuje blokový rám.
- `inline-block` —prvek generuje blokový rám a obsah prvku se formátuje jako blok. Prvek sám se však zpracovává jako řádkový (obdobně jako řádkové nahrazované prvky). Z pohledu okolních prvků se tedy jedná o řádkový prvek (chová se jako jeden znak či obrázek v textu), z pohledu prvku samotného jde ale o blok (může obsahovat další blokové i řádkové prvky).
- `list-item` —prvek generuje jeden hlavní blokový rám a případně i doplňkový rám; formátuje se jako položka seznamu (viz Seznamy [3.9.2]).
- `none` — prvek negeneruje žádné rámy a nemá vliv na formátování dokumentu. Ani potomky tohoto prvku negenerují žádné rámy bez ohledu na jejich hodnotu `display`. Prvek i jeho celý obsah se chová, jako by v dokumentu vůbec nebyl (nejsou tedy generovány ani neviditelné rámy, jako s `visibility:hidden` [4.3.6.1]).
- `run-in` —prvek generuje blokový rám nebo řádkové rámy podle kontextu („zatahované“ rámy [3.8.5.5]). Prvek se pak podle výsledného zformátování chová jako řádkový, nebo jako blokový.
- `table`, `inline-table`, `table-row-group`, `table-column`, `table-column-group`, `table-header-group`, `table-footer-group`,

`table-row`, `table-cell` a `table-caption` — prvek se chová jako tabulkový prvek. Podrobnosti viz Tabulky [3.10].

I když je výchozí hodnotou `display:inline`, nesmíme zapomenout, že klienty povinně používají svou *výchozí tabulku stylů*. V ní má každá značka přiřazeny vlastnosti `display` obvyklé hodnoty — např. prvky `p`, `div`, `h1`, `h2` atd. mají definováno `display:block`; prvky `a`, `strong`, `em` atd. mají `display:inline`; prvky `li` jsou typu `list-item` atd. Ukázkovou *výchozí tabulku stylů* pro HTML 4 najdete na konci této knihy.

Jako autoři však můžeme libovolnému prvku hodnotu `display` změnit a zformátovat jej jinak, než je obvyklé.

```
/* běžné hodnoty */
div { display: block }
strong { display: inline }
li { display: list-item }
td { display: table-cell }

/* změna obvyklého formátování */
a { display: block }   /* odkazy budou zobrazeny jako bloky */
ul.menu li { display: inline }  /* položky seznamu "menu" budou
zobrazeny jako řádkové prvky, na jedné řádce za sebou */
hr { display: none }   /* vodorovné linky se nebudou zobrazovat */
```

4.3.5.2 Poziční schéma — position

Vlastnost **position** určuje, jaké poziční schéma se použije pro formátování prvku. Pozici ovlivňují také následující vlastnosti `top`, `right`, `bottom`, `left`, `z-index` a vlastnosti obtékání `float` a `clear`.

–> Poziční schémata [3.8.6]

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
position	`static` | `relative` | `absolute` | `fixed`	`static`	nelze	ne	všechny kromě generovaného obsahu	vizuální

Hodnoty mají následující význam:

- `static` — prvek se formátuje v normálním toku dokumentu, jeho *souřadnice* (vlastnosti `top`, `right`, `bottom` a `left`) se ignorují.
- `relative` — prvek se formátuje v normálním toku dokumentu (stejně jako `static`), následně je ze své normální pozice posunut o hodnoty souřadnic. Okolních prvků se to netýká, formátují se, jako by prvek posunut nebyl.
- `absolute` — prvek je vyňat z normálního toku a zformátován v rámci svého *omezujícího bloku* pro absolutní pozicování. Pozici určují jeho *souřadnice*. Vrstvu,

v níž je zobrazen, ovlivňuje vlastnost `z-index`. Viz Absolutní pozicování [943.8.6.6].

- `fixed` — stejně jako `absolute`, navíc je prvek *zafixován* na své pozici. Při posunu *průzoru* po dokumentu se nepohybuje, ostatní obsah se posouvá „pod ním“. V případě *stránkovaných médií* se zobrazuje na stejném místě na každé stránce.

4.3.5.3 Souřadnice prvku — top, right, bottom, left

Tato skupina vlastností určuje souřadnice při pozicování. Pokud má prvek `position:static`, nemají žádný význam.

Při *absolutním pozicování* (`static`, `fixed`) určují vzdálenost horní (`top`), pravé (`right`), spodní (`bottom`), resp. levé (`left`) **hrany *okraje*** (*margin*) prvku od horní, pravé, spodní, resp. levé hrany jeho *omezujícího bloku*. Při *relativním pozicování* (`relative`) určují posun prvku z jeho pozice v normálním toku.

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`top` `bottom`	`<velikost>` | `<procenta>` | `auto`	`auto`	relativně k výšce omezujícího bloku	ne	pozicované prvky	vizuální
`left` `right`			relativně k šířce omezujícího bloku			

Hodnota `<velikost>` udává fixní vzdálenost/posun prvku. Hodnota `<procenta>` udává vzdálenost/posun relativně k šířce, resp. výšce *omezujícího bloku*. Pokud omezující blok nemá výšku zadanou (závisí jen na velikosti obsahu), procentní hodnoty `top` a `bottom` se interpretují jako `auto`. S hodnotou `auto` se pozice vypočte automaticky [3.8.7.8].

```
p { position: static }
span.mocnina {
      position: relative;
      top: -0.5em;
      font-size:70%
}
#menu {
      position: absolute;
      right: 0;
      top: 2em;
}

@media screen {
      #logo { position: static }
}
@media print {
      #logo {
             position: static;
             left: 0;
             top: 0;
      }
}
```

4.3.5.4 Vrstvy — z-index

Vlastnost **z-index** určuje *vrstvu*, v níž se zobrazuje prvek, pokud je *absolutně pozicovaný*. V ostatních pozičních schématech nemá žádný význam.

–> Absolutní pozicování a vrstvy [3.8.6.7]

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`z-index`	`<celé číslo>` | `auto`	`auto`	nelze	ne	pozicované prvky	vizuální

Má-li prvek **`z-index:auto`**, je ve stejné *vrstvě* jako jeho rodičovský prvek. Pro své potomky nevytváří nové *podvrstvy*. Hodnota **`<celé_číslo>`** umístí prvek do vrstvy daného čísla a současně zde vytvoří novou *podvrstvu* (s číslem 0) pro své potomky.

Vrstvy s vyšším číslem jsou „výše“, s nižším číslem jsou „níže“. Záporné hodnoty jsou povoleny. Vrstvy s číslem jsou vždy „nad“ vrstvami neočíslovanými (**`auto`**). Prvky ve stejné vrstvě jsou řazeny „na sebe“ v pořadí, v němž za sebou následují ve *stromu dokumentu*.

Vlastnosti rámů — typ a pozice	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
display											
block	+/-	++	++	++	+/-	++	!!!	++	++	++	
inline	--	++	++	++	--	++	--	++	++	++	> NN4 sice display:inline nepodporuje, ale i tak chybně formátuje řádkové prvky. Pokud přidáme např. prvku nebo <a> rámeček, NN4 jej zformátuje jako blok.
inline-block	--	--	--	--	--	++	--	++	--	--	
run-in	--	--	--	--	--	++	--	--	--	++	
list-item	--	--	--	++	+/-	++	!!!	++	++	++	
table	--	--	--	--	--	!!!	---	++	--	++	
inline-table	--	--	--	--	--	--	--	+/-	--	++	> NN6/Moz interpretuje inline-table jako table.
none	++	++	++	++	++	++	!!!	++	++	++	> V určitých situacích NN4 zobrazí prázdný rámeček.
position							!!!				
static	++	++	++	++	++	++	!!!	++	++	++	> NN4 position:static nezná, ale i když position není definováno (static je výchozí hodnota), nakládá jinak s prvky, které mají definováno top či left — i když by zde tyto vlastnosti měl ignorovat.
relative	+/-	++	++	++	+/-	++	+/-	++	++	++	> IE4 a NN4 nepodporují vlastnost right.
absolute	+/-	++	++	++	+/-	++	+/-	++	+/-	+/-	> V IE4 pouze s prvky DIV. IE4, NN4 a Opera nepodporují vlastnost right.
fixed	--	--	--	--	--	++	--	++	--	++	> NN až od verze 6.1
top	+/-	+/-	++	++	+/-	+	!!!	+	--	+	> I v prohlížečích, které tyto vlastnosti podporují, jsou problematická procenta. Opera vztahuje chybně procenta k výšce prvku (místo k výšce omez. bloku), NN6/Moz a IE5/Mac nepoužijí procentní posun top/bottom pokud je již posunut rodičovský prvek.
left	+/-	+/-	++	++	+/-	++	!!!	++	--	++	
bottom	--	--	++	++	--	+	--	+	--	+	
right	--	--	++	++	--	++	--	++	--	++	
z-index	++	++	++	++	++	++	++	++	++	++	

4.3.5.5 Styl obtékání — float

Vlastnost **float** určuje, zda bude prvek obtékán (bude plovoucí vlevo či vpravo), nebo ne. Používá se pouze pro prvky, které nejsou absolutně pozicovány.

–> Plovoucí prvky [3.8.6.5]

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`float`	`left` | `right` | `none`	`none`	nelze	ne	všechny kromě pozicovaných prvků a generovaného obsahu	vizuální

Hodnota **`left`** určuje, že prvek bude posunut k levé hraně svého *omezujícího bloku.* Následující obsah bude plynout podél jeho pravé hrany, počínaje horní hranou tohoto plovoucího prvku. Vlastnost **`display`** se ignoruje. Pro hodnotu **`right`** platí totéž, text plyne podél levé hrany prvku, a ten je posunut k pravé straně. S hodnotou **`none`** prvek není plovoucí.

Plovoucí prvky musí mít definovánu šířku (*vnitřními rozměry* u nahrazovaných prvků nebo vlastností **`width`**).

4.3.5.6 Zrušení obtékání — clear

Vlastnost **clear** řídí, jak obsah obtéká plovoucí prvky.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`clear`	`none` | `left` | `right` | `both`	`none`	nelze	ne	blokové prvky	vizuální

Hodnota **`left`** nařizuje, aby se horní okraj prvku (margin-top) zvětšil tak, aby horní hrana jeho rámečku byla zobrazena až pod spodní vnější hranou (hranou okraje) všech vlevo plovoucích prvků, které jsou v dokumentu uvedeny před ním. Jinými slovy, obsah prvku je

zobrazen až za všemi vlevo plovoucími prvky a je zrušeno jejich obtékání. Hodnota `right` obdobně zakazuje obtékání vpravo plovoucích prvků, hodnota `both` pak ruší všechna obtékání.

S hodnotou `none` nejsou žádná omezení a obsah obtéká všechny plovoucí prvky.

Vlastnost `clear` ruší pouze obtékání plovoucích prvků, které prvku předcházejí. Nijak neomezuje obtékání uvnitř prvku či u prvků následujících.

```
p.pozn {
      float: left;
      width: 30%;
}
h1 { clear: left }
```

Vlastnosti rámů — obtékání	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
float	!!!	!!!					!!!				> Plovoucí prvky jsou poměrně náročné na implementaci. U jednoduchých prvků (hlavně obrázků) je obtékání obvykle v pořádku, složitější konstrukce však mívají nečekané výsledky. Je proto nutné je vždy důkladně testovat.
left	!!!	!!!	++	++	++	++	!!!	++	++	++	> Nejhorší je zpracování obtékání v NN4 a IE4. Opera 4 umísťuje plovoucí prvky poněkud nepřesně. Ve všech prohlížečích jsou větší či menší problémy s plovoucími textovými prvky.
right	!!!	!!!	++	++	++	++	!!!	++	++	++	
none	++	++	++	++	++	++	++	++	++	++	
clear	!!!	!!!					!!!				
none	++	++	++	++	++	++	++	++	++	++	> Také zde je složitá implementace a při hodně složitých konstrukcích se prohlížeč může „zbláznit" a zobrazit stránku poměrně nesmyslně.
left	!!!	!!!	++	++	++	++	!!!	++	++	++	
right	!!!	!!!	++	++	++	++	!!!	++	++	++	
both	++	++	++	++	++	++	++	++	++	++	

4.3.6 Viditelnost a přetékání

4.3.6.1 Viditelnost — visibility

Vlastnost **visibility** určuje, zda jsou zobrazeny rámy generované prvkem.

–> Viditelnost [3.8.8.2]

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
visibility	`visible | hidden | collapse`	inherit	nelze	ne	všechny	vizuální

S hodnotou `visible` je prvek viditelný. Hodnota `hidden` jej učiní neviditelným, resp. zcela průhledným. Narozdíl od vlastnosti `display:none` jsou však rámy pro prvek vygenerovány a ovlivňují formátování okolních prvků, stejně jako když je prvek viditelný. *Potomky* neviditelného prvku, které mají `visibility:visible`, budou zobrazeny.

Hodnota `collapse` se používá pouze v tabulkách [3.10.5.2]. V jiných prvcích, než jsou řádky a sloupce tabulek, má stejný význam jako `hidden`.

Vlastnost `visibility` se často využívá pro dynamické efekty [5.1.11].

```
#menu { visibility:hidden }
#menu h4 { visibility:visible }
```

4.3.6.2 Přetékání obsahu — overflow

Vlastnost **overflow** definuje, zda je obsah blokového prvku oříznut, pokud přesahuje jeho rám.

–> Přetékání a ořezávání [3.8.8.1]

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
overflow	`visible | hidden | scroll | auto`	`visible`	nelze	ne	blokové a nahrazované prvky	vizuální

Výchozí hodnota `visible` zajišťuje, že přetékající obsah prvku bude vykreslen i mimo rám. Formátování okolních prvků se ale stále řídí rozměry a umístěním rámu, přetékající obsah na ně nemá vliv a může do okolních prvků přesahovat.

S hodnotou `hidden` je přetékající obsah skryt (obsah je oříznut podle rámu prvku a cokoli jej přesahuje, již není vidět). Hodnota `scroll` přikazuje klientu, aby na prvku zajistil posuvný mechanismus (např. přidal kolem prvku posuvníky). Na obsah přetékající mimo rám se pak uživatel může přesunout stejně, jako posouvá dokument v *průzoru*. Pokud koncové zařízení posuvníky nedokáže poskytnout (např. tisk dokumentu na papír), skrytý obsah se zobrazí.

Interpretace hodnoty `auto` závisí na konkrétním klientu, měla by ale zajistit zobrazení posuvného mechanismu, pokud obsah přetéká.

```
p.clanek {
      height: 250px;
      overflow: scroll;
}
div.logo { overflow:hidden }
```

4.3.6.3 Viditelná oblast — clip

Vlastnost **clip** určuje viditelnou oblast rámu prvku.

–> Přetékání a ořezávání [3.8.8.1]

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
clip	`rect(top,right,bottom,left) | auto`	`auto`	nelze	ne	blokové a nahrazované prvky	vizuální

Výchozí hodnota `auto` nastaví viditelnou oblast shodně s okraji rámu prvku, bude viditelný celý obsah. Jinou hodnotou je možné nastavit jiný obrazec, který má tvořit viditelnou oblast. V současnosti je k dispozici pouze obdélník, definovaný funkcí `rect()`, v budoucích verzích CSS však být přidány další typy oblastí.

Funkce `rect()` definuje viditelnou oblast ve tvaru obdélníka. Má čtyři parametry oddělené čárkou, po řadě `top`, `right`, `bottom` a `left`. Hodnoty `top`, resp. `bottom` určují vzdálenost horní, resp. dolní hrany viditelné oblasti od horní hrany obsahu prvku. Hodnoty `left` a `right` určují vzdálenost levé a pravé hrany od levé hrany obsahu. Tyto hodnoty si lze představit také jako souřadnice viditelné oblasti počítané od levého horního rohu obsahu prvku. Bude-li mít levý

horní roh viditelné oblasti souřadnice `(x1,y1)` a pravý dolní roh `(x2,y2)`, bude funkce ve tvaru `rect(y1,x2,y2,x1)`.

Parametry mohou být buďto typu `<velikost>` (absolutní vzdálenost s jednotkou, záporné hodnoty jsou povoleny), nebo mohou mít hodnotu auto (příslušná hrana bude shodná s hranou prvku). Definice `clip:auto` je proto totéž jako `clip:rect(auto,auto,auto,auto)`.

```
p { clip: rect(5px,150px,120px,10px) }
#menu { clip: rect(5px,auto,auto,10px) }
h1 { clip: auto }
```

4.3.6.4 Zalamování řádek — white-space

Vlastnost `white-space` řídí způsob, kterým jsou zpracovávány *mezery* (prázdný prostor) v textu. Ačkoli se jedná spíše o vlastnost textu, významně ovlivňuje přetékání a případné ořezání obsahu, proto byla zařazena do této skupiny.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
white-space	`normal` | `pre` | `nowrap`	`normal`	nelze	ano	blokové prvky	vizuální

Výchozí hodnota `normal` ponechává všechny *mezery* v textu k dispozici prohlížeči pro zalomení řádek. Každá *mezera* může být nahrazena odřádkováním, aby se co nejefektivněji vyplnila celá řádka. Dodatečné řádkové zlomy mohou být přidány sekvencemi `\A` v generovaném obsahu (např. v prvcích `br` v HTML atd.).

Hodnota `nowrap` tuto interpretaci *mezer* zakazuje, řádky budou zalomeny jen tam, kde je to v kódu přímo přikázáno (sekvencemi `\A`). Hodnota `pre` funguje obdobně, navíc zalomí řádky také tam, kde se vyskytuje odřádkování v kódu (jako značka `pre` v HTML).

```
p.program { white-space: pre }
blockquote { white-space: nowrap }
```

Vlastnosti rámů — viditelnost a přetékání	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
visibility											
visible	++	++	++	++	++	++	++	++	++	++	
hidden	++	++	++	++	++	++	++	++	++	++	
overflow											
visible	!!!	!!!	!!!	!!!	!!!	++	!!!	++	++	++	> IE/Win4-6, IE4/Mac a NN4 chybně natáhnou rám, aby pojal celý obsah, místo aby jej obsah přetékal.
hidden	++	++	++	++	++	++	+	++	+	+	> NN4 nezobrazí spodní rámeček, Opera nezobrazí rámeček vůbec.
scroll	++	++	++	+	!!!	!!!	!!!	++	!!!	!!!	> IE4/Mac, NN4 a Opera chybně zobrazí prvek jako overflow:hidden. IE6/Win často zbytečně přidá i vodorovný posuvník. V IE/Win funguje kolečko myši až od verze 5.5. IE5/Mac někdy zobrazí obsah korektně teprve po posunutí a návratu zpět.
auto	++	++	++	+	!!!	!!!	!!!	++	!!!	!!!	> Obdobně jako u scroll. V IE5/Macvodorovný posuvník někdy překryje obsah a navíc produkuje velmi zvláštní chyby v tabulkách. Scroll, auto a hidden působí mnohé další komplikace v IE5/Mac, především při použití v DHTML.
clip											
rect(x1 y1 x2 y2)	--	--	++	++	--	++	--	++	--	++	
white-space											
normal	--	--	++	++	--	++	++	++	++	++	
pre	--	--	--	++	--	++	++	++	++	++	
nowrap	--	--	++	++	--	++	--	++	++	++	> Hodnota nowrap může být nekorektně zpracována ve všech prohlížečích v hodně složitých konstrukcích, především pokud obsahují plovoucí prvky.

4.4 Formátování řádkových prvků

4.4.1 Vlastnosti písma

4.4.1.1 Typ písma — font-family

Vlastnost **font-family** definuje typ (rodinu) písma, kterým se má zobrazit text.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
font-family	[<rodina>|<obecná_rodina>] [, <rodina>|<obecná_rodina>]*	závisí na prohlížeči	nelze	ano	všechny	vizuální

Hodnotou této vlastnosti je seznam názvů rodin písma, oddělený čárkami. Názvem rodiny je buďto konkrétní písmo používané v klientu (názvy obsahující mezeru **musí** být uzavřeny v uvozovkách), nebo obecná rodina. CSS používá obecné rodiny `serif`, `sans-serif`, `monospace`, `fantasy` a `cursive`. Postup zpracování těchto seznamů popisujeme na jiném místě.

–> Vzhled písma [3.8.8.4]; CSS v české prostředí [5.2]

```
body { font-family: "Arial CE", "Helvetica CE", Arial, sans-serif }
p { font-family: 'Times CE', 'Times New Roman', serif }
code { font-family: monospace }
```

4.4.1.2 Styl písma — font-style

Vlastnost **font-style** přepíná styl písma mezi základním (neskloněným) a skloněným řezem (italikou).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
font-style	`normal | italic | oblique`	`normal`	nelze	ano	všechny	vizuální

Výchozí hodnota `normal` nastaví základní (neskloněné) písmo, hodnoty `italic` a `oblique` písmo skloněné. S hodnotou `oblique` se použije písmo označené klientem jako *oblique* (obvykle písma s přídomkem *Oblique, Slanted* či *Incline*). S hodnotou `italic` se použije písmo označené jako *italic* (obvykle písma s přídomkem *Italic, Cursive* či *Kurziv*), pokud není dostupné, použije písmo označené jako *oblique* (viz [3.8.8.4]).

```
p.pozn { font-style: italic }
em {
      font-style: normal;
      font-family: cursive;
}
```

4.4.1.3 Kapitálky — font-variant

Vlastnost **font-variant** přepíná mezi základním písmem a kapitálkami (styl písma, kde jsou malé znaky vytvořeny zmenšenými velkými znaky, angl. *small caps*).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
font-variant	`normal | small-caps`	`normal`	nelze	ano	všechny	vizuální

Výchozí hodnota **`normal`** nastaví základní písmo, s hodnotou **`small-caps`** se text zobrazí v kapitálkách. Kapitálky může klient vytvořit uměle, případně místo nich použít pouze verzálky (viz [3.8.8.4]).

```
h1 { font-variant: small-caps }
```

4.4.1.4 Síla písma — font-weight

Vlastnost **font-weight** nastavuje sílu (tučnost) písma.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
font-weight	`normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900`	`normal`	nelze	ano (vypočítaná číselná hodnota)	všechny	vizuální

Hodnoty **`100`** až **`900`** tvoří posloupnost od nejslabšího po nejsilnější písmo. Následující hodnotě musí odpovídat písmo alespoň stejně silné jako hodnotě předcházející. Výchozí hodnota **`normal`** odpovídá hodnotě **`400`**, hodnota **`bold`** je totéž jako **`700`**.

Klient setřídí všechny dostupné řezy použité rodiny písma podle jejich síly a přiřadí jim hodnoty **`100`** až **`900`**, např. v pořadí *Light, Book* (obvykle **`400`**), *Medium* (obvykle **`500`**), *Bold* (**`700`**), *Heavy, ExtraBlack.* Pokud nějaká hodnota zůstane nepřiřazena, přiřadí se jí nejbližší vhodné písmo. Hodnoty **`bolder`** a **`lighter`** použijí nejbližší písmo, které je silnější, resp. slabší než písmo rodičovského prvku. Pokud již žádné takové není k dispozici, písmo se nezmění a prvku se pouze nastaví následující, resp. předchozí hodnota v posloupnosti (pokud ještě není použita krajní hodnota **`900`**, resp. **`100`**).

Klient může písma různé síly vytvořit uměle, má-li k dispozici technologii, která to umožňuje.

```
strong { font-weight: bold } /* 700 */
p { font-weight: 400 }       /* normal */
p span { font-weight: lighter }
```

4.4.1.5 Velikost písma — font-size

Vlastnost **font-size** definuje velikost písma. Tato velikost odpovídá typografickému *čtverčíku*, některé znaky jej mohou přesahovat.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
font-size	xx-small | x-small | small | medium | large | x-large | xx-large | larger | smaller | <velikost> | <procenta>	medium	relativní k velikosti písma (font-size) rodičovského prvku	ano (vypočítaná hodnota)	všechny	vizuální

Hodnota **<velikost>** definuje písmo absolutně nebo relativně. Relativní jednotky (**em**, **ex**) a hodnoty **<procenta>** se vztahují k velikosti písma rodičovského prvku (pozor, u všech ostatních vlastností se **em** a **ex** vztahují k velikosti písma prvku samotného). Např. hodnoty **1.2em** a **120%** shodně definují písmo 1,2krát větší než písmo rodičovského prvku.

W3C doporučuje používat především *absolutní velikosti* písma (zadané klíčovými slovy). Výchozí hodnota **medium** definuje základní (střední) velikost písma (písmo nastavené uživatelem jako základní). Klíčová slova obsahující „small“ popisují menší písmo, hodnoty s „large“ písmo větší.

Specifikace CSS 1 určovala koeficient 1.5 pro převod mezi jednotlivými stupni, ten však byl příliš velký. CSS 2 jej snížila na 1.2, to však stále působilo příliš velké skoky u menších velikostí. CSS 2.1 nakonec uvádí převodní tabulku, která používá různé koeficienty pro různé velikosti, aby rozdíly mezi nimi byly přiměřené.

Absolutní velikosti CSS	xx-small	x-small	small	medium	large	x-large	xx-large	——
Koeficient vzhledem k základní velikosti	3/5	3/4	8/9	**1**	6/5	3/2	2	3
	60%	75%	89%	**100%**	120%	150%	200%	300%
Koeficient vzhledem k sousední velikosti	□ 80% □	□ 125% 84,39% □	□ 119% 89% □	□ **112,5%** **83,33%** □	□ 120% 80% □	□ 125% 75% □	□ 133% 66,67% □	□ 150% □
Příklad	8.4pt	10.5pt	12.44pt	**14pt**	16.8pt	21pt	28pt	42pt
Odpovídající nadpisy HTML	H6	——	H5	H4	H3	H2	H1	——
Odpovídající velikosti staré značky FONT	1	——	2	3	4	5	6	7

Tyto koeficienty mohou být pro různá média odlišné. Klient také může upravit převodní tabulku pro jednotlivá písma, různé typy písma tak mohou mít různé koeficienty. Prohlížeče by však na základě tohoto výpočtu nikdy neměly použít písmo používající na výšku znaku méně než 9 obrazových bodů.

Relativní velikosti **smaller** a **larger** předepisují použít písmo menší, resp. větší, než je písmo rodičovského prvku. Např. pokud rodičovský prvek používá velikost písma **small**, s hodnotou **larger** by se měla použít velikost **medium**. Pokud velikost písma rodičovského prvku není blízko některému z klíčových slov, klient může velikost interpolovat mezi hodnotami z tabulky, nebo vybrat hodnotu nejbližší; u velikostí mimo rozsah tabulky může hodnoty extrapolovat.

```
body { font-size: normal }
p { font-size: larger }
#menu { font-size: 12pt }
#menu strong { font-size: 1.2em }
```

4.4.1.6 Sdružená vlastnost font

Vlastnost **font** slouží k nastavení všech vlastností písma současně.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
font (1. možnost použití)	[**<fst>** || **<fv>** || **<fw>**]? **<fsz>**[/**<lh>**]? **<ff>**	viz dílčí vlastnosti	pro hodnoty font-size a line-height	ano	všechny	vizuální

<fst>=font-style; <fv>=font-variant; <fw>=font-weight; <fsz>=font-size; <lh>=line-height; <ff>=font-family

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
font (2. možnost použití)	caption | icon | menu | message-box | small-caption | status-bar	viz dílčí vlastnosti	nelze	ano	všechny	vizuální

Vlastnost **font** má dvě možné syntaxe.

První možností je použití **font** jako *sdružené vlastnosti.* Lze jí definovat všechny vlastnosti písma současně — ve tvaru, který je velmi podobný předpisům písma používaným v typografii. U jednotlivých podvlastností je třeba dodržet uvedené pořadí:

- Nejprve se uvádí hodnoty vlastností **font-style**, **font-variant** a **font-weight**, v libovolném pořadí, oddělené mezerou. Kterákoli z nich (nebo žádná) nemusí být uvedena. Pokud je zde uvedena alespoň jedna hodnota **normal**, použije se pro všechny chybějící vlastnosti.
- Za nimi následuje hodnota **font-size**. Tato hodnota je zde povinná.
- Volitelně může být hodnota **font-size** následována lomítkem **/** a hodnotou **line-height** (bez mezer). Pokud není tato hodnota uvedena, nesmí zde být ani lomítko.
- Nakonec se povinně uvádí hodnota **font-family** (seznam rodin písma).

Sdružená vlastnost **font** tedy musí minimálně obsahovat dvojici **font-size** a **font-family**, ostatní vlastnosti jsou nepovinné.

Pokud je vlastnost **font** použita, všechny dílčí vlastnosti, které může obsahovat, jsou nejprve nastaveny na svou výchozí hodnotu a následně jsou předefinovány ty podvlastnosti, které jsou zde uvedeny. Vynecháme-li proto některou z nich, musíme vždy počítat s tím, že bude mít výchozí hodnotu — i když jsme ji předtím definovali. Např.:

```
#prvek { font-weight: bold }
#prvek { font: 12pt/1.2 "Helvetica CE",Arial,sans-serif }
```

Vlastnost `font` zde přiřadí chybějícím podvlastnostem `font-style`, `font-variant` a `font-weight` jejich výchozí hodnoty, což u `font-weight` znamená hodnotu `normal`. Písmo tedy nebude tučné!

```
body { font: normal serif }
/* chybějící vlastnosti font-style, font-variant, font-weight a
line-height budou mít přiřazenu výchozí hodnotu */
em { font: normal bold smaller inherit }
/* hodnota normal se použije pro font-style i pro font-variant; písmo
bude tučné, menší velikosti, line-height bude mít výchozí hodnotu a
rodina písma se zdědí z rodičovského prvku (inherit) */
p.pozn { font: normal small-caps bolder 12pt/14pt serif }
```

Druhou možnou syntaxí vlastnosti `font` je její použití jakožto standardní vlastnosti. Její hodnotou pak může být jedno z klíčových slov, odkazujících na písma použitá v systému uživatele:

- `caption` — písmo použité pro ovládací prvky (tlačítka, seznamy atd.)
- `icon` — písmo použité pro popisy ikon
- `menu` — písmo použité pro nabídky
- `message-box` — písmo použité v dialogových rámečcích
- `small-caption` — písmo pro zobrazení malých ovládacích prvků
- `status-bar` — písmo použité ve stavových rámečcích oken

Přiřazení klíčového slova nastaví všechny dílčí vlastnosti podle charakteristiky použitého písma, je ale možné je následně upravit. Pokud v systému není písmo uvedeného typu, klient by je měl nahradit písmem podobným, nebo alespoň svým výchozím písmem.

```
button { font: caption }
#zahlavi { font: menu }
```

Vlastnosti písma	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
font-family											
název_písma	++	++	++	++	++	++	++	++	++	++	
serif	++	++	++	++	++	++	++	++	++	++	
sans-serif	++	++	++	++	++	++	++	++	++	++	
cursive	++	++	++	++	++	++	--	++	++	++	> Opera nerespektuje písmo nastavené uživatelem v předvolbách a použije písmo jiné.
fantasy	++	++	++	++	++	++	--	++	++	++	> NN4/Win fantasy a cursive nepodporuje, NN4/Mac ano.
monospace	++	++	++	++	++	++	++	++	++	++	
font-style											
normal	++	++	++	++	++	++	++	++	++	++	
italic	++	++	++	++	++	++	++	++	++	++	
oblique	++	++	++	++	++	++	--	++	++	++	
font-variant											
normal	++	++	++	++	++	++	--	++	++	++	
small-caps	+	+	+	++	+	++	--	++	++	++	> IE4-5/Win a IE4/Mac místo kapitálek uměle vytvoří jen verzálky (tj. totéž jako text–transform:uppercase). CSS to sice povoluje, ale kapitálky to nejsou.
font-weight											
normal	++	++	++	++	++	++	++	++	++	++	
bold	++	++	++	++	++	++	++	++	++	++	
bolder	++	++	++	++	++	++	+/-	++	++	++	> Nepodporováno v NN4/Mac.
lighter	++	++	++	++	++	++	--	++	++	++	
100 - 900	++	++	++	++	++	++	++	++	++	++	
font-size											
absolutní velikost (xx-small až xx-large)	+	+	+	++	+	++	++	++	++	+	> IE4-5/Win používá základní velikost písma (velikost nestylovaného textu) pro small místo pro medium. Písmo o velikosti medium (a vůbec písmo zadané absolutní velikostí) je tak v IE vždy větší než v jiných prohlížečích.
relativní velikost — larger	++	++	!!!	++	++	++	++	++	++	++	
relativní velikost — smaller	++	++	!!!	++	++	++	++	++	++	++	
<velikost>	++	++	++	++	++	++	++	++	++	++	
<procenta>	++	++	++	++	++	++	++	++	++	++	
font											
<font-family>	++	++	++	++	++	++	+/-	++	++	++	
<font-style>	++	++	++	++	++	++	+/-	++	++	++	
<font-variant>	+	+	+	++	+	++	--	++	++	++	> Podpora v NN4/Mac je lepší než v NN4/Win.
<font-weight>	++	++	++	++	++	++	+/-	++	++	++	
<font-size>	+	+	+/-	++	++	++	++	++	++	++	
<line-height>	++	++	++	++	++	++	!!!	++	++	++	

4.4.2 Vlastnosti textu

–> Vzhled textu [3.8.8.5]

4.4.2.1 Slovní a mezislovní mezery — letter-spacing, word-spacing

Vlastnost **letter-spacing** upravuje velikost mezer mezi znaky textu, **word-spacing** velikost mezislovních mezer.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`letter-spacing`	`normal` | `<velikost>`	`normal`	nelze	ne	všechny	vizuální

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`word-spacing`	`normal` | `<velikost>`	`normal`	nelze	ne	všechny	vizuální

Hodnota `normal` ponechává normální velikost meziznakových, resp. mezislovních mezer definovaných pro použité písmo. Klient může tyto hodnoty upravit pro potřeby vodorovného zarovnání textu (především při `text-align:justify`).

Hodnota `<velikost>` **zvětší** o zadanou hodnotu základní meziznakové, resp. mezislovní mezery (výsledná mezera = běžná mezera + `<velikost>`). Meziznakové mezery již klient nesmí

upravovat. Hodnoty mohou být záporné, ale jejich interpretace může být omezena možnostmi prohlížeče.

```
em.prolozeny { letter-spacing:0.5em }
#menu { word-spacing: 30px }
```

4.4.2.2 Dekorace — text-decoration

Vlastnost **text-decoration** definuje dekorace, které se mají přidat ke zformátovanému textu. Pokud je definována pro blokový prvek, dekorace se aplikuje na všechny jeho řádkové *následovníky*. Pokud je použita pro řádkový prvek, aplikuje se na všechny rámy, které prvek generuje. Prohlížeč tuto vlastnost musí ignorovat, pokud prvek je prvek prázdný, nebo nemá žádný textový obsah (např. obsahuje jen obrázek).

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
text-decoration	none | [underline || overline || line-through || blink]	none	nelze	ne (viz text)	všechny	vizuální

Výchozí hodnota `none` nepřidává žádné dekorace. S hodnotou `underline` je každá řádka podtržena, `overline` přidá linku nad každou řádkou a `line-through` doprostřed každé řádky textu (přeškrtnutí). Hodnota `blink` způsobí blikání textu (střídavé zobrazování a skrývání), prohlížeče by však blikání raději neměly používat.

Barva linky by měla být odvozena od barvy definované vlastností `color`. Tato vlastnost se nedědí, ale každý *následovník* prvku by měl být zobrazen se stejnou dekorací (např. všechny podtržené). Barva linky by měla zůstat stále stejná, i když má *následovník* jinou hodnotu vlastnosti `color` (bude mít tedy jinou barvu textu, ale barva linky by měla být nezměněná v celém prvku).

Všechny dekorace mohou být zobrazeny současně, lze proto definovat zároveň několik hodnot (kromě `none`).

```
a { text-decoration: underline }
span.skrtnuto { text-decoration: line-through }
div.obelinky { text-decoration: underline overline }
```

4.4.2.3 Transformace znaků — text-transform

Vlastnost **text-transform** řídí změnu velikosti malých znaků na velké a naopak.

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
text-transform	capitalize | uppercase | lowercase | none	none	nelze	ano	všechny	vizuální

Při výchozí hodnotě `none` nejsou prováděny žádné transformace. Hodnota `uppercase` převede všechny znaky na velké (*verzálky,* ABC), hodnota `lowercase` převede znaky na malé (*mínusky,* abc). Hodnota `capitalize` převede První Znak Každého Slova Na Velký.

Prohlížeč může považovat hodnotu `text-transform` za `none` u znaků, které nejsou součástí kódování Latin-1, případně u jazyků, v nichž se převod znaků liší od převodních tabulek uvedených v ISO10646 (viz [Reference]).

```
abbr { text-transform: uppercase }
h1.nazevpisne { text-transform: capitalize }
```

4.4.2.4 Vodorovné zarovnání řádek — text-align

Vlastnost **text-align** definuje, jak bude zarovnán řádkový obsah bloku.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
text-align	`left | right | center | justify | <řetězec>`	závisí na prohlížeči a směru textu	nelze	ano	blokové prvky a buňky tabulek	vizuální

Výchozí hodnota záleží na prohlížeči a použitém směru textu, u nás je to obvykle hodnota `left`. Hodnoty `left`, `right`, resp. `center` zarovnají obsah řádek k levé, resp. pravé hraně, resp. na osu bloku.

Hodnota `justify` zarovná obě hrany řádek, klient může kromě mezislovních mezer zvětšit případně i meziznakové mezery (viz `letter-spacing` výše). Protože prohlížeče vesměs nepodporují dělení slov na konci řádků, tento způsob zarovnání by se měl používat jen v dostatečně širokých blocích. Klient není povinen tento způsob zarovnání podporovat.

Hodnota `<řetězec>` se používá pouze v tabulkách [3.10.4.2]. Obsah buněk ve sloupci bude zarovnán tak, aby se uvedený řetězec nacházel v jedné linii. U ostatních prvků se interpretuje jako `left` (resp. `right` při opačném směru textu).

```
p.clanek { text-align: justify }
p.autor { text-align: right }
td.suma { text-align: "," }
```

Pozn.: Vlastnost text-align zarovnává pouze řádkové prvky uvnitř bloku. Pro vodorovné zarovnání bloků se používají hodnoty `margin-left` a `margin-right` [4.3.1].

4.4.2.5 Odsazení první řádky — text-indent

Vlastnost **text-indent** definuje *odsazení* první řádky textu od levé hrany bloku. Klient by měl toto odsazení zobrazit jako prázdný prostor (mezeru).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
text-indent	`<velikost> | <procenta>`	0	relativní k šířce omezujícího bloku	ano	blokové prvky	vizuální

Hodnota `<velikost>` určuje konkrétní hodnotu odsazení, `<procenta>` pak relativně k šířce *omezujícího bloku*. Hodnota může být záporná, ale její interpretace může být omezena

možnostmi prohlížeče. Pokud je hodnota záporná, první řádka bloku bude *předsazená* (obsah bude *přetékat* rám prvku a jeho viditelnost bude řídit vlastnost `overflow` [4.3.6.2]).

```
p { text-indent: 5% }
p.predsaz { text-indent: -3em }
```

4.4.2.6 Svislé zarovnání — vertical-align

Vlastnost **vertical-align** řídí svislé zarovnání rámů řádkových prvků na jedné řádce. V tabulkách má mírně odlišný význam [3.10.4.1].

–> Výpočet výšky řádek [3.8.7.11]

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`vertical-align`	`baseline` | `sub` | `super` | `top` | `middle` | `bottom` | `text-top` | `text-bottom` | `<procenta>` | `<velikost>`	`baseline`	relativní k výšce řádku (line-height) prvku samotného	ne	řádkové prvky a buňky tabulek (prvky table-cell)	vizuální

Možné hodnoty mají následující význam:

- `baseline` — zarovná účaří rámu (nebo jeho spodní hranu, pokud účaří nemá) na účaří rodičovského rámu
- `middle` — zarovná svislý střed rámu polovinu *x-výšky* rodičovského prvku nad účaří rodičovského rámu (střed rámu na střed výšky písma rodičovského prvku)
- `sub`, `super` — posune účaří rámu dolů, resp. nahoru na pozici odpovídající dolním, resp. horním indexům v rodičovském rámu; nemá to však žádný vliv na velikost písma
- `text-top`, `text-bottom` — zarovná horní, resp. dolní hranu rámu s horním, resp. dolním účařím textu rodičovského prvku
- `top`, `bottom` — zarovná horní, resp. dolní hranu rámu s horní, resp. dolní hranou bloku řádky
- `<procenta>`, `<velikost>` — posune rám nahoru (kladná hodnota) nebo dolů (záporná hodnota) o zadanou hodnotu. Procenta se vztahují k hodnotě `line-height`. Hodnota `0` znamená totéž jako `baseline`.

```
img { vertical-align: middle }
span.index {
      font-size: 67%;
      vertical-align: sub;
}
span.exponent {
      font-size: 0.5em;
      vertical-align: 80%;
}
```

4.4.2.7 Výška řádek — line-height

Vlastnost **line-height** upřesňuje výšku řádek. U řádkových prvků se používá pro výpočet výšky řádek (vyjma řádkových nahrazovaných prvků, kde je výška rámu dána hodnotou `height`). U blokového prvku, jehož obsah tvoří řádkové prvky, definuje minimální výšku bloků řádek uvnitř tohoto prvku.

–> Výpočet výšky řádek [3.8.7.11]

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`line-height`	`normal | <číslo> | <velikost> | <procenta>`	`normal`	relativní k velikosti písma (font-size) prvku samotného	ano (viz text)	všechny prvky	vizuální

Hodnota `<velikost>` určuje velikost, která se použije pro výpočet výšky řádky

Hodnoty `<číslo>` a `<procenta>` neudávají výšku řádky přímo, ale definují *koeficient* pro její výpočet. *Vypočítaná hodnota* výšky řádky vznikne vynásobením velikosti písma a tohoto koeficientu. Např. při velikosti písma 10pt a hodnotě `line-height` 1.2 bude výška řádky 12pt. Výchozí hodnota `normal` nařizuje prohlížeči, aby použil nějaký koeficient vhodný pro dobrou čitelnost textu. W3C doporučuje koeficient 1.0 až 1.2.

Pouze pokud je `line-height` udána jako `<číslo>`, dědí se toto číslo (jako *definovaná hodnota*), ve všech ostatních případech se dědí *hodnota vypočítaná.* Záporné hodnoty nejsou povoleny.

```
p { line-height: 1.3em }
#cast1 { line-height: 1.33 } /* potomky zdědí tento koeficient */
#cast2 { line-height: 133% } /* potomky zdědí vypočítanou hodnotu */
```

Vlastnosti textu	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
word-spacing											
normal	--	--	--	++	++	++	--	++	++	++	
<velikost>	--	--	--	++	++	++	--	++	++	++	
letter-spacing											
normal	++	++	++	++	++	++	--	++	++	++	
<velikost>	++	++	++	++	++	++	--	++	++	++	
text-decoration	!!!	!!!	!!!		!!!		!!!	!!!	!!!	!!!	
none	+	+	+	++	+	++	+	+	++	++	Většina prohlížečů zpracovává dekorace chybně. Má-li prvek nastavenu dekoraci a jeho potomek má none, má být rodičovský efekt zobrazen i na tomto prvku (i např. stejnou barvou podtržení). Kromě IE5/Mac a Opery však none u potomka zruší všechny efekty.
underline	++	++	++	++	+	++	+	++	++	++	
overline	++	++	++	++	++	++	--	++	++	++	
line-through	++	++	++	++	++	++	++	++	++	++	Opera a NN6/Moz nepoužijí dekorace na obrázky v textu. NN6/Moz navíc nepoužívá dekoraci rodičovského prvku, ale svým způsobem ji replikuje na všechny potomky, což neodpovídá specifikaci CSS.
blink	--	--	--	--	--	--	++	++	++	++	
text-transform											
capitalize	++	++	++	++	++	++	++	++	++	++	
uppercase	++	++	++	++	++	++	++	++	++	++	
lowercase	++	++	++	++	++	++	++	++	++	++	
none	++	++	++	++	++	++	++	++	++	++	
text-align											
left	++	++	++	++	++	++	++	++	++	++	
right	++	++	++	++	++	++	++	++	++	++	
center	++	++	++	++	++	++	++	++	++	++	
justify	++	++	++	++	--	++	!!!	++	++	++	> V NN4 selhává zarovnávání justify v tabulkách.
text-indent											
<velikost>	++	++	++	++	++	++	++	++	++	++	
<procenta>	++	++	++	++	++	++	++	++	++	++	
vertical-align											
baseline	++	++	++	++	++	++	--	++	++	++	
sub	++	++	++	++	++	++	--	++	++	++	
super	++	++	++	++	++	++	--	++	++	++	
top	--	--	++	++	++	++	--	++	++	++	
text-top	--	--	++	++	++	++	--	++	++	++	
middle	!!!	--	++	++	++	++	--	++	++	++	
bottom	--	--	++	++	!!!	++	--	++	++	++	
text-bottom	--	--	++	++	!!!	++	--	++	++	++	
<procenta>	--	--	--	+/-	!!!	+	--	++	++	++	> IE5/Mac počítá procenta nesmyslně, je-li line-height:normal. Při zadané výšce řádky už to provede korektně. IE6/Win sice procenta použije, ale výpočet je nepochopitelný — 0% vychází na dolní účaří, 100% na horní účaří; algoritmus nelze vysledovat.
line-height											
normal	++	++	++	++	++	++	++	++	++	++	
<číslo>	++	++	++	++	++	++	+/-	++	++	++	
<velikost>	++	++	++	++	++	++	!!!	++	++	++	> NN 4 zde chybně akceptuje záporné hodnoty.
<procenta>	++	++	++	++	++	++	+/-	++	++	++	

4.5 Barvy a pozadí prvků

4.5.1 Barva popředí — color

Vlastnost **color** definuje barvu popředí prvku (především barvu textu).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
color	<barva>	závisí na prohlížeči	nelze	ano	všechny	vizuální

Výchozí hodnotu určuje prohlížeč. Obvykle se barva zdědí z rodičovského prvku, výchozí barvy pro text a odkazy si většinou uživatel může nastavit sám. Hodnota **<barva>** definuje konkrétní barvu klíčovým slovem nebo hodnotami složek RGB (viz [3.3.6]).

```
body { color: black }
a:link { color: #00f }
a:visited { color: rgb(20%,20%,60%) }
```

4.5.2 Styl pozadí

Pozadí se zobrazuje pod celou *oblastí obsahu, výplně a rámečku*, okraje prvku jsou vždy průhledné. Obsah prvku toto pozadí vždy překrývá.

–> Styl pozadí [3.8.8.6]

4.5.2.1 Barva pozadí — background-color

Vlastnost background-color nastavuje barvu pozadí prvku.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
background-color	<barva> | transparent	transparent	nelze	ne	všechny	vizuální

Výchozí hodnota **transparent** zajišťuje, že skrze prvek „prosvítá“ pozadí rodičovského prvku. Hodnota **<barva>** má stejný význam jako ve vlastnosti **color**. Barvu pozadí rodičovského prvku (např. **html** nebo **body**) určuje prohlížeč, obvykle je výchozí pozadí šedé nebo bílé. Uživatel má většinou možnost tuto barvu nastavit.

```
body { background-color: white }
#menuitem { background-color: #CC99FF }
```

4.5.2.2 Obrázek na pozadí — background-image

Vlastnost **background-image** popisuje obrázek, který se má zobrazit na pozadí prvku. Pokud nepokrývá celou jeho plochu (viz následující vlastnosti), na zbylé části pozadí je zobrazena barva

určená vlastností **background-color**. Je-li použit obrázek na pozadí, měla by být také vždy definována barva pozadí — pro případ, že použitý obrázek nebude možno zobrazit.

Pokud obrázek (nebo jeho kopie při opakování) přesahuje plochu pozadí, přebývající části jsou oříznuty podle hran rámečku prvku.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
background-image	<url> | none	none	nelze	ne	všechny	vizuální

Výchozí hodnota **none** určuje, že nebude použit žádný obrázek. Hodnota **<uri>** (viz [3.3.5]) popisuje internetovou adresu obrázku. Není-li obrázek dostupný nebo jej nelze zobrazit, na pozadí se zobrazí pouze plocha definovaná vlastností **background-color**.

Autor by nikdy neměl použít pozadí pod textem, které jakkoli zhoršuje čitelnost textu nad ním. Pomocí obrázků na pozadí lze vytvářet i dynamická obrázková tlačítka [5.1.12].

```
body { background-image: url("../img/papir.gif)" }
```

4.5.2.3 Opakování obrázku — background-repeat

Vlastnost **background-repeat** určuje, zda a jakým způsobem se obrázek opakuje na celé ploše pozadí. Pokud se obrázek opakuje, je umístěn původní obrázek na pozici danou vlastností **background-position** (viz dále) a jeho kopie jsou rozmístěny po obou stranách tak, aby se jejich hrany přímo dotýkaly a vyplnily v definovaném směru celou plochu pozadí prvku.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
background-repeat	repeat | repeat-x | repeat-y | no-repeat	repeat	nelze	ne	všechny	vizuální

S hodnotou **repeat-x** se obrázek opakuje pouze vodorovně, s **repeat-y** pouze svisle. Výchozí hodnota **repeat** zajistí, že obrázek se opakuje v obou směrech a jako dlaždice pokryje celou plochu pozadí. Hodnota **no-repeat** opakování zakáže a obrázek se na pozadí umístí pouze v jediné kopii.

```
body { background-repeat: no-repeat }
#menuitem { background-repeat: repeat-y }
```

4.5.2.4 Umístění obrázku — background-position

Vlastnost **background-position** definuje pozici obrázku na pozadí. Pokud se obrázek opakuje, na tuto pozici je umístěn výchozí obrázek a jeho ostatní kopie jsou rozloženy kolem něj.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
background-position	[<procenta>|<velikost>]{1,2} | [[top|center|bottom] || [left|center|right]]	0% 0%	relativní k rozměrů rámu prvku samotného	ne	blokové a nahrazované prvky	vizuální

Hodnotu této vlastnosti tvoří dvojice (oddělená mezerou), udávající pozici obrázku ve vodorovném a svislém směru (první je vždy uvedena vodorovná pozice).

- `<procenta> <procenta>` — dvojice `0% 0%` určuje, že levý horní roh obrázku je umístěn v levém horním rohu *oblasti výplně* (oblast padding) prvku. Dvojice `100% 100%` pak umístí pravý dolní roh obrázku do pravého dolního rohu této oblasti. Procentní hodnoty mezi `0%` a `100%` udávají pozici plynule mezi těmito dvěma polohami.
- `<velikost> <velikost>` — dvojice `10px 20px` určuje, že levý horní roh obrázku bude posunut 10px doprava a 20px dolů od levého horního rohu oblasti výplně prvku.
- `top`, `top center` a `center top` — totéž jako `50% 0%`
- `right top` a `top right` — totéž jako `100% 0%`
- `left`, `left center` a `center left` — totéž jako `0% 50%`
- `center` a `center center` — totéž jako `50% 50%`
- `right`, `right center` a `center right` — totéž jako `100% 50%`
- `bottom left` a `left bottom` — totéž jako `0% 100%`
- `bottom`, `bottom center` a `center bottom` — totéž jako `50% 100%`
- `bottom right` a `right bottom` — totéž jako `100% 100%`

Pouze jediná procentní či číselná hodnota určuje pozici obrázku v pouze ve vodorovném směru, pro svislý směr se v tom případě použije hodnota `50%`. Číselné a procentní hodnoty je možné kombinovat (např. `2cm 50%`), s klíčovými slovy je však kombinovat nelze (všechny možné kombinace klíčových slov jsou uvedeny výše). Záporné hodnoty jsou povoleny.

```
body { background-position: 50% 50% }
#menuitem { background-position: left }
```

4.5.2.5 Ukotvení obrázku — background-attachment

Vlastnost **background-attachment** určuje ukotvení obrázku — zda se má posouvat při posouvání obsahu, či nikoli.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`background-attachment`	`scroll | fixed`	`scroll`	nelze	ne	všechny	vizuální

Výchozí hodnota `scroll` zajistí, že se obrázek na pozadí bude posouvat společně s obsahem, pokud je na něj aplikován nějaký posuvný mechanismus (posouvání dokumentu v *průzoru*, posouvání obsahu prvku při `overflow:scroll` atd.). S hodnotou `fixed` bude obrázek „ukotven“ na původním místě a posouvání na něj nebude mít vliv — obsah se bude pohybovat nad ním.

Klient může hodnotu `fixed` interpretovat jako `scroll`, měl by však podporovat alespoň `fixed` pro kořenový prvek dokumentu (`body` nebo `html`).

```
body { background-attachment: fixed }
```

4.5.2.6 Sdružená vlastnost background

Sdružená vlastnost **background** umožňuje definovat všechny vlastnosti pozadí současně.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
background	**<bg-col> || <bg-img> || <bg-rep> || <bg-att> || <bg-pos>**	viz dílčí vlastnosti	pro vlastnost background-position	ne	všechny	vizuální

```
<bg-col> = background-color; <bg-img> = background-image; <bg-rep> = background-repeat;
<bg-att> = background-attachment; <bg-pos> = background-position
```

Dílčí vlastnosti se mohou uvádět v libovolném pořadí, oddělují se mezerami. Pokud v CSS použijeme vlastnost `background`, jsou všem dílčím vlastnostem pozadí nejprve přiřazeny jejich výchozí hodnoty a pak jsou případně předefinovány ty, které zde uvedeme.

```
body { background: white url("logo.jpg") 50% 50% no-repeat fixed }
#menuitem { background: repeat-x top url("bull.gif") #CC99FF }
```

Vlastnosti barev a pozadí	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
color											
<barva>	++	++	++	++	++	++	++	++	++	++	
background-color											
<barva>	++	++	++	++	++	++	!!!	++	++	++	> NN4 nepoužije barvu na celé pozadí. Pokud má prvek rámeček, bude sice obdélníkové, ale mezi ním a rámečkem zůstane prázdné místo (nemá nic společného s oblastí padding). Pokud prvek rámeček nemá, pozadí bude dokonce pouze pod řádky textu.
transparent	++	++	++	++	++	++	!!!	++	!!!	++	> NN4 nepoužije průhledné pozadí na samotný prvek, ale na pozadí jeho rodiče (což dokáže způsobit podivné děrované plochy). Opera4 má problémy s opakovanými obrázky v kombinaci s průhledným pozadím.
background-image											
<uri>	++	++	++	++	++	++	++	++	++	++	
none	++	++	++	++	++	++	++	++	++	++	
background-repeat											
repeat	!!!	++	++	++	++	++	++	++	++	++	> IE4/Win opakuje obrázek jen směrem dolů a doprava. Pokud není umístěn do levého horního rohu, vlevo a nad ním nic nebude. NN4 se chová podobně, ale protože nepodporuje background-position, těžko říci, jakým způsobem obrázky na pozadí skládá.
repeat-x	!!!	++	++	++	++	++	+/-	++	++	++	
repeat-y	!!!	++	++	++	++	++	+/-	++	++	++	> IE4/Win opakuje obrázek pouze směrem doprava a dolů. Směry doleva a nahoru ignoruje.
no-repeat	++	++	++	++	++	++	++	++	++	++	
background-position											
<procenta>	++	++	++	++	++	++	--	++	++	++	
<velikost>	++	++	++	++	++	++	--	++	++	++	
top	++	++	++	++	++	++	--	++	++	++	
center	++	++	++	++	++	++	--	++	++	++	
bottom	++	++	++	++	++	++	--	++	++	++	
left	++	++	++	++	++	++	--	++	++	++	> NN4 tyto vlastnosti nepodporuje, umístění ani ukotvení obrázku nelze určit.
right	++	++	++	++	++	++	--	++	++	++	
background-attachment											
scroll	++	++	++	++	++	++	--	++	++	++	
fixed	++	++	++	++	++	++	--	++	++	++	
background											
<background-color>	++	++	++	++	++	++	!!!	++	++	++	
<background-image>	++	++	++	++	++	++	+/-	++	++	++	> Svérázný přístup NN4 k pozadí je popsán výše.
<background-repeat>	!!!	++	++	++	++	++	+/-	++	++	++	
<background-attachment>	++	++	++	++	++	++	--	++	++	++	
<background-position>	++	++	++	++	++	++	--	++	++	++	

4.6 Generovaný obsah a seznamy

Generovaný obsah zajišťují vlastnosti `quotes` a `content` pseudo-prvků `:before` a `:after`. Seznamy a jejich položky jsou formátovány stejnou technologií. Prohlížeče, které tyto vlastnosti dosud veřejně nepodporují, to dělají alespoň skrytě, jakoby tyto vlastnosti byly pro některé prvky jazyka podporovány (viz např. definici prvků `li` či `br` v ukázce výchozího stylu pro HTML 4 na konci knihy).

4.6.1 Generování obsahu — content, quotes

Vlastnost **content** definuje obsah, který bude přidán před obsah prvku (použití s `:before`), resp. na jeho konec (`:after`). Vlastnost **quotes** definuje styl uvozovek, které se mají použít jako hodnoty `open-quote` a `close-quote` vlastnosti `content`.

–> Generovaný obsah [3.9.1]

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
content	`[ <string> | attr(X) | open-quote | close-quote | no-open-quote | no-close-quote ]+`	"" (prázdný řetězec)	nelze	ne	pseudo-prvky :before a :after	všechna

Hodnotou vlastnosti `content` je seznam řetězců a klíčových slov, které dohromady tvoří text, který se vloží před/za obsah prvku. Tento obsah netvoří kód jazyka, ale čistý neformátovaný text, zobrazí se přesně tak, jak je zadán. Speciální znaky jazyka či entity HTML nebudou zpracovány — vložíme-li za obsah prvku text '`<br>`', je to totéž, jako bychom do kódu dokumentu na konec prvku přidali '`&lt;br&gt;`'. Překládají se pouze *escape-sekvence* (začínající znakem „`\`“), např. sekvence `\A` zajistí přechod na nový řádek (viz [3.3.7]).

Kromě řetězců zde můžeme použít i klíčová slova. Hodnota `attr(X)` vloží do generovaného textu hodnotu atributu `X` (např. `attr(title)`). Hodnoty `open-quote` a `close-quote` vloží uvozovku podle aktuální úrovně jejich vnoření, tak jak jsou definovány vlastností `quotes` tohoto prvku. Hodnoty `no-open-quote` a `no-close-quote` nevloží nic, pouze zvýší, resp. sníží úroveň vnoření uvozovek (jakoby do textu vložily neviditelné uvozovky).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
quotes	`[<řetězec> <řetězec>]+ | none`	závisí na prohlížeči	nelze	ano	všechny	vizuální

Hodnotou vlastnosti `quotes` je seznam párů řetězců, první z dvojice definuje počáteční (otevírací) uvozovku, druhý uvozovku koncovou (uzavírací). Další páry pak definují uvozovky pro další úrovně vnoření.

—> Definování uvozovek [3.9.1.2]

```
p.pozn:before { content: "Poznámka: " }
img.stextem:after { content: "\APopis: " attr(alt) }
cite { quotes: '„' '“' ',' '‘' '»' '«' }
cite:before { content: open-quote }
cite:after  { content: close-quote "[autor: attr(title)]" }
@media print {
      a:after { content: "[URL: " attr(href) "]" }
}
```

Generovaný obsah	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
content											
<řetězec>	--	--	--	--	--	--	--	++	--	++	Vzhledem k takto omezené podpoře generovaného obsahu s ním nelze ještě příliš počítat, především s možností definování uvozovek. S korektní prací s uvozovkami a citacemi vůbec bude třeba počkat na novější verze prohlížečů.
<klíčová slova>	--	--	--	--	--	--	--	++	--	++	
funkce attr()	--	--	--	--	--	--	--	++	--	--	
quotes	--	--	--	--	--	--	--	++	--	++	

4.6.2 Vlastnosti seznamů

Vlastnosti seznamů definují styl, jakým jsou formátovány jejich položky a značky u nich. Mají význam pouze u prvků, které jsou označeny jako položky seznamu (`display:list-item`). Formátování popisují pouze obecně, neumožňují autorům definovat přesné umístění, barvu ani velikost značek.

—> Formátování seznamů [3.9.2]

4.6.2.1 Typ seznamu — list-style-type

Vlastnost **list-style-type** definuje styl značky uvozující položky seznamu, pokud není definovaný nebo dostupný obrázek daný vlastností `list-style-image`.

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
list-style-type	disc | circle | square | decimal | decimal-leading-zero | lower-roman | upper-roman | lower-alpha | upper-alpha | lower-latin | upper-latin | lower-greek | hebrew | armenian | georgian | cjk-ideographic | hiragana | katakana | hiragana-iroha | katakana-ir	disc	nelze	ano	prvky s display:list-item	vizuální

Hodnota `none` nezobrazí před položkami žádnou značku. Ostatní hodnoty definují značky tří typů — symboly, číselné posloupnosti a abecední posloupnosti.

- **Symboly:** `disc`, `circle` a `square`. Jejich konkrétní podoba závisí na prohlížeči.
- **Číselné posloupnosti.** Označí položky čísly počínaje jedničkou. Následující položka seznamu má označení o 1 vyšší než položka předchozí.
  - `decimal` — číslování arabskými čísly (1, 2, 3, … 99)
  - `decimal-leading-zero` — totéž s počáteční nulou (01, 02, … 99)

  - `lower-roman` — číslování malými římskými čísly (i, ii, iii, iv, …)
  - `upper-roman` — číslování velkými římskými čísly (I, II, III, IV, …)
- **Abecední posloupnosti.** Označí položky písmeny příslušné abecedy počínaje prvním znakem. CSS nespecifikuje, jak má klient postupovat po vyčerpání všech znaků abecedy (pro delší seznamy by se proto mělo používat raději číselné označení).
  - `lower-alpha` a `lower-latin` — malé znaky ASCII (a, b, c, d, …)
  - `upper-alpha` a `upper-latin` — velké znaky ASCII (A, B, C, D, …)
  - `lower-greek` — malé znaky řecké alfabety (α, β, γ, δ, …)
  - `upper-greek` — velké znaky řecké alfabety (Α, Β, Γ, Δ, …)
  - další posloupnosti národních abeced, dosud vesměs nepodporované

Pokud klient nepodporuje definovaný číselný systém, měl by použít `decimal`.

```
ul { list-style-type: disc }
ol { list-style-type: upper-alpha }
ol ol { list-style-type: decimal }
ol ol ol { list-style-type: lower-roman }
```

4.6.2.2 Obrázková značka — list-style-image

Vlastnost **list-style-image** definuje obrázek, který se má použít jako značka u položek seznamu.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
list-style-image	`<uri>` | none	none	nelze	ano	prvky s display:list-item	vizuální

Pokud je definována hodnota `none`, nebo obrázek nelze zobrazit, použije se značka definovaná vlastností `list-style-type`.

```
ul { list-style-image: url("../img/hvezdicka.png") }
```

4.6.2.3 Umístění značky — list-style-position

Vlastnost **list-style-position** definuje umístění značky (*doplňkového rámu*) vůči hlavnímu rámu prvku (viz [3.9.2]).

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
list-style-position	inside | outside	outside	nelze	ano	prvky s display:list-item	vizuální

```
ul { list-style-position: outside }
ul.kompaktni { list-style-position: inside }
```

4.6.2.4 Sdružená vlastnost list-style

Sdružená vlastnost **list-style** umožňuje definovat všechny vlastnosti seznamů současně.

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`list-style`	**`<ls-type>`** || **`<ls-pos>`** || **`<ls-img>`**	viz dílčí vlastnosti	nelze	ano	prvky s display:list-item	vizuální

`<ls-type>=list-style-type; <ls-pos>=list-style-position; <ls-img>=list-style-image`

Hodnotou této vlastnosti jsou hodnoty **`list-style-type`**, **`list-style-image`** a/nebo **`list-style-position`** oddělené mezerami (v libovolném pořadí). Vynechané dílčí vlastnosti nabývají své výchozí hodnoty. Jediná hodnota **`none`** se přiřadí vlastnostem **`list-style-type`** i **`list-style-image`**.

```
ul > li { list-style: none }
ul > ul li { list-style: square url("ctverecek.gif") inside }
```

Vlastnosti seznamů	IE Windows 4	IE Windows 5	IE Windows 5.5	IE Windows 6	IE Mac 4	IE Mac 5	NN/Moz 4	NN/Moz 6	Opera 4	Opera 5	Poznámky
list-style-type											
disc	++	++	++	++	++	++	--	++	++	++	> NN4 vlastnosti seznamů příliš nepodporuje, řídí se pouze značkami OL / UL. Přesto některé hodnoty rozpoznává a páchá s nimi chyby (viz např. none dále).
circle	++	++	++	++	++	++	--	++	++	++	
square	++	++	++	++	++	++	--	++	++	++	
decimal	++	++	++	++	++	++	--	++	++	++	
lower-roman	++	++	++	++	++	++	--	++	++	++	
upper-roman	++	++	++	++	++	++	--	++	++	++	
lower-alpha	++	++	++	++	++	++	--	++	++	++	
upper-alpha	++	++	++	++	++	++	--	++	++	++	
decimal-leading-zero	--	--	--	--	--	--	--	++	--	--	
lower-latin	--	--	--	--	--	--	--	++	--	--	
upper-latin	--	--	--	--	--	--	--	++	--	--	> Většina prohlížečů v seznamech typu, který nepodporuje, zobrazí značky jako u seznamů UL (tj. většinou disc), jen IE5/Mac je zobrazí jako seznam OL (decimal), což je patrně korektnější postup.
lower-greek	--	--	--	--	--	--	--	++	--	--	
upper-greek	--	--	--	--	--	--	--	++	--	--	
none	++	++	++	++	++	++	--	++	++	++	> NN4/Mac zobrazí otazníky s hodnotou none místo značek u položek.
list-style-image											
<url>	++	++	++	++	++	++	--	++	++	++	> V IE5/Mac je menší chyba - nebere ohled na výšku obrázku a je-li vyšší než výška řádek, obrázky tvořící značky u položek se budou překrývat.
none	++	++	++	++	++	++	--	++	++	++	
list-style-position											
inside	++	++	++	++	+	!!!	--	++	++	++	> IE4/Mac vykresluje příliš široký okraj značky. IE5/Mac se „zasekne“ na kombinaci display:list-item a list-style-type:inside a stránku dál nevykreslí.
outside	!!!	++	++	++	++	++	--	++	++	++	> IE4 vykreslí značku přes rámeček prvku.
list-style											
<klíčové slovo>	+/-	+/-	+/-	+/-	+/-	+/-	--	++	+/-	+/-	
<position>	+	++	++	++	+	+	--	++	++	++	
<url>	++	++	++	++	++	++	--	++	++	++	

4.7 Efekty pro uživatelské rozhraní

—> Uživatelské rozhraní [3.8.8.7]

4.7.1 Vzhled ukazatele — cursor

Vlastnost **cursor** definuje vzhled ukazatele polohovacího zařízení, pokud ukazuje na prvek (např. podobu kurzoru myši).

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
cursor	`auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize | text | wait | help | progress`	`auto`	nelze	ano	všechny	vizuální, interaktivní

Jednotlivé hodnoty mají následující význam:

- **`auto`** — klient zobrazí ukazatel podle aktuálního kontextu
- **`crosshair`** — zaměřovací kříž, obvykle ve tvaru znaku „+“
- **`default`** — výchozí ukazatel na dané platformě, obvykle ve tvaru šipky
- **`pointer`** — ukazatel označující odkaz
- **`move`** — ukazatel symbolizující přesun objektů
- **`e-resize`**, **`ne-resize`**, **`nw-resize`**, **`n-resize`**, **`se-resize`**, **`sw-resize`**, **`s-resize`**, **`w-resize`** — ukazatel symbolizující přesun hrany objektu. Označení odpovídá světovým stranám: **`n`** (*north*, sever), **`s`** (*south*, jih), **`w`** (*west*, západ), **`e`** (*east*, východ), přičemž sever je nahoře, západ vlevo. Např. ukazatel **`nw-resize`** označuje přesun levého horního rohu.
- **`text`** — ukazatel pro označení textu, obvykle ve tvaru velkého „I“
- **`wait`** — ukazatel symbolizující, že je program zaneprázdněn a uživatel musí čekat. Obvykle ve tvaru přesýpacích či ručičkových hodin.
- **`progress`** — ukazatel průběhu, kdy program provádí nějakou činnost, ale uživatel s ním může dále pracovat (ve tvaru rotujícího míče či šipky s hodinkami)
- **`help`** — ukazatel symbolizující dostupnou nápovědu. Obvykle ve tvaru otazníku, šipky s otazníkem či komiksové bubliny.

```
#menu { cursor: pointer }
a.napoveda { cursor:help }
```

4.7.2 Vzhled obrysu prvku — outline

Vlastnost **outline** definuje obrys prvků, obdobně jako se obvykle zvýrazňují objekty, které získaly *zaměření* (*focus*). Obrys je obdobný rámečku prvku (`border`), s tím rozdílem, že obrys nezabírá žádné místo navíc (je zobrazen *kolem* zformátovaného prvku a může překrývat jeho okolí) a nemusí být nutně obdélníkový (podle zvyklostí použitého systému a klientu).

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`outline-width`	jako **`border-width`**	`medium`	nelze	ne	všechny	vizuální, interaktivní
`outline-style`	jako **`border-style`** (kromě `hidden`)	`none`				
`outline-color`	`<barva> | invert`	`invert`				
`outline`	**`<outline-width>`** || **`<outline-style>`** || **`<outline-color>`**	viz dílčí vlastnosti				

Obrys vytvořený vlastností `outline` vždy překrývá prvek, je vždy „nahoře“. Neovlivňuje pozici ani velikost prvku ani prvků okolních. Zobrazení či skrytí obrysu proto nezpůsobí změnu formátování dokumentu.

Vlastnost `outline-width` používá stejné hodnoty jako `border-width`, `outline-style` stejné jako `border-style` (vyjma hodnoty `hidden`). Vlastnost `outline-color` může používat všechny barvy, navíc má možnou hodnotu `invert`, která zajistí inverzní zobrazení všech obrazových bodů obrysového rámečku. Tím lze zajistit výrazný obrys na jakémkoli pozadí.

Vlastnost `outline` definuje všechny dílčí vlastnosti obrysů současně. Povšimněte si, že obrys je vždy na všech stranách stejný, neexistují vlastnosti jako *outline-top* či *outline-left*.

Protože obrys prvku nemá vliv na formátování a je vždy navrchu, může překrývat okolní prvky. CSS však nespecifikuje, jak se mají zobrazit překrývající se obrysy dvou prvků, nebo obrysy prvků skrytých za jinými objekty.

```
a:focus { outline: solid 4px invert }
```

Vizuální efekty	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
cursor											
crosshair	++	++	++	++	++	++	--	++	--	--	
pointer	!!!	!!!	!!!	++	++	++	--	++	--	--	> IE/Win pointer podporuje až od verze 6. Pro ukazatel všechny IE používají vlastní klíčové slovo hand. Je možné použít univerzální definici { cursor:pointer;cursor:hand } (kvůli chybnému zpracování definicv IE je nutné dodržet toto pořadí!)
move	++	++	++	++	++	++	--	++	--	--	
text	++	++	++	++	++	++	--	++	--	--	
wait	++	++	++	++	++	++	--	++	--	--	> Je třeba počítat s tím, že každý OS používá jiné kurzory. Např. wait vypadá ve Windows jako přesýpací hodiny, zatímco na MacOS jako náromkové hodinky; help ve Windows vypadá jako šipka s otazníkem, jinde je to jen (větší) otazník atd.
help	++	++	++	++	++	++	--	++	--	--	
progress	++	++	++	++	++	++	--	--	--	--	> Nové rozšíření z iniciativy MS.
„směr"-resize	+	++	++	++	+	++	--	++	--	--	> Pozor, zde se ukazatele **velmi** liší mezi různými platformami.
default	++	++	++	++	++	++	--	++	--	--	
auto	+	+	+	+	+	++	--	+	--	--	> Většina prohlížečů řeší hodnotu auto dost nejasně (ale nejasná je i specifikace). Raději nepoužívat.
<uri> (načtení kurzoru ze souboru)											> Z původní specifikace CSS byly externí kurzory staženy. V současnosti je svým způsobem podporují některé prohlížeče na některých OS (každý jinak a každý s jiným formátem souboru). Jednotné řešení přinese až CSS3.
outline	--	--	!!!	!!!	--	!!!	--	!!!	--	--	
outline-width	--	--	--	--	--	!!!	--	!!!	--	--	
outline-color	--	--	--	--	--	!!!	--	!!!	--	--	> Mozilla outline sice nepodporuje, ale má zase interní vlastnosti -moz-outline-width, -moz-outline-color a -moz-outline-style, které mají stejný význam. Implementace outline je však ještě v plenkách a velmi chybová. Doporučuji zatím nepoužívat.
outline-style	--	--	--	--	--	!!!	--	!!!	--	--	

4.8 Vlastnosti tabulek

4.8.1 Umístění nadpisu tabulky — caption-side

Vlastnost **caption-side** určuje umístění nadpisu tabulky vůči tabulce samotné.

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
caption-side	top | bottom | left | right	top	nelze	ano	prvky table-caption	vizuální

Výchozí hodnota **top** zajistí umístění nadpisu nad tabulkou, hodnota **bottom** pod tabulkou. Hodnoty **left** a **right** umístí nadpis po stranách tabulky, nejsou však zatím široce podporovány a prohlížeč místo nich použije hodnotu **top**.

4.8.2 Typ formátování tabulky — table-layout

Vlastnost **table-layout** definuje model, který se má použít pro výpočet šířky a formátování tabulky. S hodnotou **fixed** se použije fixní model, s hodnotou **auto** model automatický.

–> Výpočet šířky tabulky [3.10.3.2]

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
table-layout	auto | fixed	auto	nelze	ne	prvky table a inline-table	vizuální

```
table caption { caption-side: bottom }
```

4.8.3 Rámování v tabulkách

4.8.3.1 Slučování rámů — border-collapse

Vlastnost **border-collapse** vybírá model, který se má použít pro rámování prvků tabulky. S hodnotou **separate** se použije *oddělený model,* s hodnotou **collapse** *model slučovací.*

–> Rámečky v tabulkách [3.10.5]

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
border-collapse	collapse | separate	collapse	nelze	ano	prvky table a inline-table	vizuální

```
table.vysledky { border-collapse: separate }
```

Pozn.: I když je výchozí hodnotou **collapse**, některé prohlížeče používají chybně jako výchozí hodnotu **separate**. Není proto dobré spoléhat se jen na výchozí hodnotu a raději bychom rámovací model měli definovat v tabulce stylů.

4.8.3.2 Odstup mezi rámy — border-spacing

Vlastnost **border-spacing** definuje rozestup mezi rámečky buněk v *odděleném modelu* rámování tabulek. Je obdobou atributu `cellspacing` v HTML. Ve *slučovacím modelu* se ignoruje.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
border-spacing	`<velikost> <velikost>?`	0	nelze	ano	prvky table a inline-table	vizuální

Hodnotou je jedno nebo dvě čísla s jednotkami (oddělené mezerami). První údaj definuje vodorovný rozestup mezi rámečky, druhý údaj rozestup svislý. Pokud je uvedena pouze jedna hodnota, definuje shodně rozestup v obou směrech.

```
table.vysledky { border-spacing: 1em 0.5em }
table.cenik { border-spacing: 4px }
```

4.8.3.3 Zobrazení prázdných buněk — empty-cells

Vlastnost **empty-cells** definuje, zda se mají zobrazovat prázdné a neviditelné buňky v *odděleném modelu* rámování tabulek. Ve *slučovacím modelu* se ignoruje.

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
empty-cells	`show | hide`	`show`	nelze	ano	buňky tabulek (prvky table-cell)	vizuální

Při (výchozí) hodnotě `show` se zobrazí všechny buňky, s hodnotou `hide` se prázdné a neviditelné buňky, ani jejich rámeček nebudou vykreslovat, na jejich místě bude pouze pozadí tabulky.

Za prázdné se považují takové buňky, které nemají žádný obsah, nebo jejich obsah tvoří pouze *prázdný prostor,* tj. znaky ASCII CR (`\0D`), LF (`\0A`), tabelátor (`\09`) a mezera (`\20`). Jiné, byť neviditelné znaky (např. ` `) se za prázdný prostor nepovažují. Neviditelné jsou buňky s `visibility:hidden`.

```
table.vysledky { empty-cells: hide }
```

Vlastnosti tabulek	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
table-layout	--	--	+	+	--	--	--	++	--	+	> table-layout:fixed způsobí v IE podivné věci, pokud není udána šířka tabulky. Opera počítá rozměry nepřesně.
border-collapse	--	--	+	+	--	--	--	++	--	++	> V IE je problematické vykreslování tabulek, které mají definovaný cellspacing.
border-spacing	--	--	--	--	--	--	--	++	--	++	
empty-cells	--	--	--	--	--	!!!	--	++	--	++	> Částečná podpora v IE/Mac od verze 5.1

4.9 Řízení stránkování

Stránkovaná média [3.4] rozdělují dokument na jednotlivé stránky. Následující vlastnosti umožňují vynutit či zakázat přechod na novou stránku.

—> Formátování stránek [3.11]

4.9.1 Odstránkování před, za a uvnitř prvku — page-break-before, page-break-after a page-break-inside

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
`page-break-before` **`page-break-after`**	`auto | always | avoid | left | right`	`auto`	nelze	ne	blokové prvky	vizuální, stránko-vaná
`page-break-inside`	`auto | avoid`			ano		

Výchozí hodnota **`auto`** nevynucuje ani nezakazuje přechod na novou stránku před, za, resp. uvnitř rámu generovaného prvkem. Hodnota **`always`** vynutí odstránkování před/za prvkem, hodnota **`avoid`** odstránkování naopak zakáže. Hodnoty **`left`** a **`right`** vynutí jedno či dvě odstránkování tak, aby se následující obsah zobrazil na levé, resp. pravé stránce. CSS nespecifikuje, zda je první stránka levá či pravá, klient také může hodnoty **`left`** a **`right`** interpretovat jako **`always`**.

Stránkový zlom také ovlivňují hodnoty těchto vlastností u rodičovského, předcházejících a následujících prvků. Pokud mají hodnotu jinou než **`auto`**, platí, že **`always`**, **`left`** a **`right`** mají přednost před **`avoid`**.

Tyto vlastnosti ovlivňují stránkování jen u blokových prvků, které nejsou plovoucí. Stránkování navíc nelze vynutit uvnitř buněk tabulek a u absolutně pozicovaných prvků (**`absolute`** i **`fixed`**). Vynucená stránkování u těchto prvků musí klient ignorovat.

```
h1.kniha { page-break-before: right }
h2.kniha { page-break-before: always }
.kapitola p:first-child { page-break-after: avoid }
p.poznamka { page-break-inside: avoid }
```

Stránkování	IE Windows				IE Mac		NN/Moz		Opera		Poznámky
	4	5	5.5	6	4	5	4	6	4	5	
page-break-before	--	--	++	++	--	++	--	--	--	++	> Ostatní vlastnosti stránek definované v CSS2 (a vypuštěné v CSS2.1) podporuje pouze Opera 5. V CSS3 budou stránkovaná média řešena jinak a komplexněji.
page-break-after	--	--	++	++	--	--	--	--	--	++	
page-break-inside	--	--	--	--	--	--	--	--	--	++	

4.10 Vlastnosti zvukových stylů

Následující vlastnosti se používají na *zvukových médiích,* při vizuálním formátování dokumentu nemají žádný význam.

—> Zvukové styly [3.12]

4.10.1 Nastavení hlasitosti

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
volume	`<číslo>` | `<procenta>` `silent` | `x-soft` | `soft` | `medium` | `loud` | `x-loud`	`medium`	vztahují se ke zděděné hodnotě	ano	všechny prvky	zvuková

Hodnota **`volume`** představuje střední hlasitost přehrávaného zvuku. Hodnota **`<číslo>`** je číslo 0 až 100, kde 0 je *nejmenší slyšitelná* hlasitost, 100 *nejvyšší přijatelná* hlasitost. Procenta jsou vypočítána ze zděděné hodnoty a poté ořízntuty na rozmezí 0—100. Dalšími možnými hodnotami je jedno z klíčových slov: **`silent`** (vůbec žádný zvuk), **`x-soft`** (totéž jako hodnota 0), **`soft`** (jako 25), **`medium`** (50), **`loud`** (75) nebo **`x-loud`** (100).

Konkrétní úrovně odpovídající hodnotám 0—100 mohou být nastaveny uživatelem a odpovídat použitému zařízení — např. klient v automobilu počítá s vyšším hlukem a minimální hlasitost bude výrazně vyšší než minimum nastavené pro zařízení v domácím kině.

```
h1 { volume: loud }
p { volume: 40 }
```

4.10.2 Způsob čtení textu

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
speak	`normal` | `none` | `spell-out`	`normal`	nelze	ano	všechny prvky	zvuková

Tato vlastnost určuje, zda a jak bude čten textový obsah prvku. Hodnota **`none`** zakáže jakékoli čtení textu (obsah bude ignorován, potomky prvku ale mohou tuto hodnotu předefinovat). S hodnotou **`normal`** bude text přečten podle zvyklostí použitého jazyka, se **`spell-out`** bude text hláskován znak po znaku (což je užitečné především pro zkratky).

Pokud má prvek **`volume:silent`**, jeho prezentace zabere stejně času, jako by byl prvek přečten nahlas (nebude však nic slyšet) — naproti tomu hodnota **`speak:none`** způsobí, že prvek bude „vynechán“, jeho prezentace nezabere žádný čas.

```
p { speak: normal }
abbr { speak: spell-out }
#menu { speak: none }
```

4.10.3 Přestávky před a za textem

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
pause-before	<čas> | <procenta>	závisí na zařízení	viz text	ne	všechny prvky	zvuková
pause-after						
pause	[<čas> | <procenta>] {1,2}					

Tyto vlastnosti určují přestávku, která se má udělat před zahájením či po skončení čtení textu. Hodnota **<čas>** určuje absolutní délku přestávky (v sekundách či milisekundách). Autoři by měli dávat přednost relativním hodnotám: **<procenta>** se vztahují k převrácené hodnotě vlastnosti **speech-rate** (viz dále), která určuje rychlost čtení. Je-li např. rychlost čtení 120 slov za minutu, převrácenou hodnotou je doba trvání jednoho slova, což je v tomto případě 500ms na slovo. K této hodnotě se vztahuje relativní zadání přestávky, tedy **pause-before:100%** zde znamená 500ms, **pause-after:20%** bude 100ms.

Sdružená vlastnost **pause** definuje obě dílčí vlastnosti současně. Pokud jsou její hodnotou dva údaje, první je **pause-before**, druhý **pause-after**. Pokud je zadána hodnota jediná, bude použita shodně pro obě dílčí vlastnosti.

```
h1 { pause-before: 200% }
p { pause-after: 120% }
span.vsuvka { pause: 250ms }
```

4.10.4 Zvukové ikony

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
cue-before	<uri> | none	none	nelze	ne	všechny prvky	zvuková
cue-after						
cue	[<uri> | none] {1,2}					

Vlastnosti **cue-before** a **cue-after** specifikují zvukový soubor, který se má přehrát před, resp. za obsahem prvku. Autoři tak pomocí nich mohou vytvořit *zvukové ikony,* které napomáhají sémantickému rozlišení úseků dokumentu. *Přestávka* určená vlastnostmi **pause** (viz výše) se umísťuje mezi tyto zvukové ikony a vlastní obsah prvku.

Sdružená vlastnost **cue** definuje obě dílčí vlastnosti současně. Pokud jsou její hodnotou dva údaje, první je **cue-before**, druhý **cue-after**. Pokud je zadána hodnota jediná, bude použita shodně pro obě dílčí vlastnosti.

```
h1.kapitola { cue-before: url("znelka.wav") }
p { cue-after: url("pink.aiff") }
em.smich { cue: url("haha.au") }
```

4.10.5 Zvuk na pozadí

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
play-during	<uri> mix? repeat? | auto | none	auto	nelze	ne	všechny prvky	zvuková

Vlastnost **play-during** určuje zvuk, který se má přehrávat současně s obsahem prvku. Hodnotou může být **<uri>** souboru, volitelně doplněná klíčovými slovy **mix** a **repeat**, anebo hodnota **auto** či **none**. Pokud je určen zvukový soubor, klíčové slovo **mix** zajistí, že bude smíchán se zvukem přehrávaným pro rodičovský prvek. Pokud **mix** uvedeno není, *zvukové pozadí* rodičovského prvku bude přerušeno a na pozadí bude přehrán pouze daný zvuk. Klíčové slovo **repeat** zajistí opakování zvuku ve smyčce, pokud je kratší než obsah prvku. V opačném případě zvuk zazní pouze jednou.

S hodnotou **auto** bude plynule pokračovat pozadí rodičovského prvku — narozdíl od hodnoty **inherit**, kdy je pro prvek použit zvuk rodičovského prvku, ale je spuštěn znovu od začátku. Hodnota **none** způsobí, že na pozadí nebude znít žádný zvuk — pozadí rodičovského prvku je zcela ztišeno a pokračuje, až daný prvek skončí.

```
p.kapitola { play-during: uri("hudba.wav") }
p.kapitola strong { play-during: uri("bubny.wav") mix repeat }
p.kapitola em { play-during: none }
```

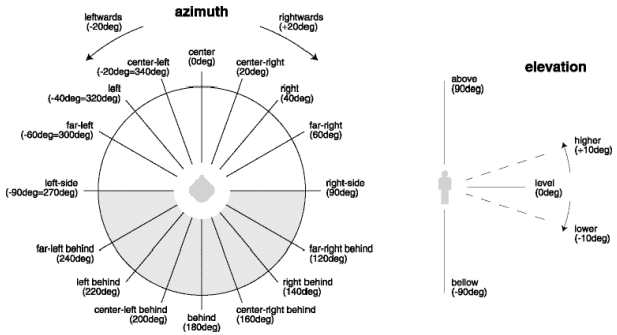
4.10.6 Prostorový zvuk

vlastnost	hodnota – uvedené možnosti, nebo inherit	výchozí	procenta	dědí se	použití na prvky	média
azimuth	<úhel> | [[left-side | far-left | left | center-left | center | center-right | right | far-right | right-side] || behind] | leftwards | rightwards	center	nelze	ano	všechny prvky	zvuková
elevation	<úhel> | below | level | above | higher | lower	level				

Vlastnosti **azimuth** a **elevation** zdánlivě umístí zdroj zvuku do prostoru kolem posluchače. Klient upraví zvuk do jednotlivých kanálů podle možností výstupního zařízení (stereofonní reproduktory, sluchátka, 3D-audio systém atd.) tak, aby měl posluchač dojem, že zvuk vychází z místa určeného těmito vlastnostmi.

Vlastnost **azimuth** určuje azimut (směr kolem svislé osy) zdroje zvuku. Hodnotou je buďto **<úhel>**, klíčové slovo zastupující úhel (volitelně doplněné klíčovým slovem **behind**), nebo relativní hodnota **leftwards** či **rightwards**. Úhel (viz jednotky CSS [3.3.8]) je v rozmezí **-360deg** až **360deg**, orientaci úhlů a význam klíčových slov vysvětluje obrázek níže. Hodnoty **leftwards** a **rightwards** posouvají zdroj zvuku proti, resp. po směru hodinových ručiček o 20° oproti aktuálnímu místění.

Vlastnost `elevation` určuje výškový úhel zdroje zvuku. Hodnotou je `<úhel>` v rozsahu `-90deg` až `90deg`, orientaci a klíčová slova popisuje obrázek níže. Hodnoty `higher` a `lower` posouvají zdroj zvuku výše, resp. níže o 10°oproti aktuálnímu umístění.



Obr. 43 — Prostorové umístění zdroje zvuku

CSS neurčuje, jakými prostředky se má požadovaného směru zdroje zvuku dosáhnout, popisuje pouze výsledný efekt. Pokud klient nedokáže umístit zdroj zvuku dozadu, definovaný úhel se převede na odpovídající pozici před posluchačem.

```
h1 { azimuth: right; elevation: above }
p.pozn { azimuth: behind }
th { azimuth: 270deg }
tr.prvni { elevation: above }
tr.druhy { elevation: level }
tr.treti { elevation: bellow }
```

4.10.7 Charakteristika hlasu

Následující skupina vlastností definuje charakteristiky hlasu, který má použít *hlasový syntezátor* při čtení obsahu prvku.

vlastnost	**hodnota** – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
speech-rate	`<číslo>` | `x-slow` | `slow` | `medium` | `fast` | `x-fast` | `faster` | `slower`	medium	nelze	ano	všechny prvky	zvuková
voice-family	`[<spec-hlas>|<obecný-hlas>],]* [<spec-hlas>|<obecný-hlas>]`	záleží na klientovi				
pitch	`x-low` | `low` | `medium` | `high` | `x-high`	medium				
pitch-range	`<číslo>`	50				
stress	`<číslo>`	50				
richness	`<číslo>`	50				

Vlastnost `speech-rate` určuje **rychlost čtení.** Hodnota `<číslo>` definuje počet slov za minutu. Výsledek je závislá i na použitém jazyku. Klíčová slova zastupují konkrétní rychlost čtení: `x-slow` (totéž jako 80 slov za minutu), `slow` (120 slov), `medium` (180 až 200 slov), `fast`

(300 slov), `x-fast` (500 slov). Relativní hodnota `slower` ubere 40 slov a `faster` přidá 40 slov za minutu k aktuální rychlosti čtení.

```
p { speech-rate: 200 }
p em { speech-rate: slower }
```

Vlastnost `voice-family` určuje **hlasovou rodinu (typ hlasu).** Syntaxe je obdobná vlastnosti `font-family` (viz [4.4.1.1]), její hodnotou je seznam typů hlasů seřazený podle priority, oddělených mezerami. Názvy obsahující znaky nepovolené pro identifikátory a názvy víceslovné musí být uzavřeny v uvozovkách. Hodnota `<spec-hlas>` je specifická rodina hlasů, používaná na konkrétním koncovém zařízení (např. Fred, Trinoids, Carlos, Victoria atd.), `<obecný-hlas>` je jedna z obecných rodin hlasů `male` (mužský), `female` (ženský) nebo `child` (dětský hlas).

```
h1 { voice-family: announcer, Fred, male }
p.dialog.matka { voice-family: Victoria, female }
p.dialog.dcera { voice-family: Princess, child }
```

Vlastnost `pitch` definuje **průměrnou výšku** (frekvenci) hlasu. Závisí na použitém typu hlasu, standardní mužský hlas je nižší (kolem 120Hz), ženský je vyšší (kolem 210Hz). Hodnota `<frekvence>` udává střední výšku hlasu v hertzích (Hz). Klíčová slova `x-low`, `low`, `medium`, `high` a `x-high` nedefinují konkrétní frekvenci, klient jim ale musí přiřadit odpovídající výšku hlasu podle použitého typu hlasu a uživatelského nastavení, přičemž musí dodržet pořadí (`low` představuje nižší frekvenci než `medium` atd.).

```
ul > li { pitch: high }
ul > ul > li { pitch: medium }
ul > ul > ul > li { pitch: low }
```

Vlastnost `pitch-range` definuje **výškové rozpětí** hlasu (rozsah intonace). Hodnotou je `<číslo>` v rozsahu 0 až 100. Hodnotě 0 odpovídá monotónní hlas konstatní výšky, 50 je běžná modulace hlasu, hodnoty nad 50 způsobí živější přednes.

```
p { pitch-range: 50 }
code { pitch-range: 0 }
```

Vlastnost `stress` určuje **sílu důrazů a přízvuků** v hlase. Hodnotou je `<číslo>` od 0 do 100. Výsledek závisí i na hodnotě `pitch-range` a na použitém jazyku. Např. `stress:50` pro mužský hlas s běžnou intonací a modulací v angličtině bude mít jiný význam než stejná hodnota v češtině nebo italštině.

```
body { stress: 50 }
```

Vlastnost `richness` definuje **plnost, jas** hlasu. Hodnotou je `<číslo>` 0 až 100. Vyšší hodnoty mají za následek ostřejší, *nosnější* hlas; s hodnotami nižšími je hlas měkčí, *medovější*.

```
p.vyhlaska { richness: 70 }
p.pohadka { richness: 20 }
```

4.10.8 Čtení zvláštních znaků

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
speak-punctuation	`code` | `none`	`none`	nelze	ano	všechny prvky	zvuková
speak-numeral	`digits` | `continuous`	`continuous`				

Tyto vlastnost řídí, jak budou přečteny některé speciální znaky v textu.

Vlastnost **speak-punctuation** určuje způsob čtení interpunkce. S hodnotou **code** budou interpunkční znaky vyslovovány, např. „(a,b)“ bude čteno jako „levá závorka a čárka b pravá závorka“. Hodnota **none** způsobí, že tyto znaky čteny nebudou a nahradí se odpovídajícími přestávkami nebo změnou intonace.

```
p { speak-punctuation: none }
code { speak-punctuation: code }
```

Vlastnost **speak-numerals** obdobně určuje způsob čtení čísel. Hodnota **digits** nastaví čtení po číslicích („123“ bude čteno jako „jedna dvě tři“), s hodnotou **continuous** se budou skupiny číslic číst dohromady jako čísla („123“ bude „sto dvacet tři“).

```
span.isbn { speak-numerals: digits }
table { speak-numerals: continuous }
```

4.10.9 Čtení záhlaví tabulek

vlastnost	hodnota – uvedené možnosti, nebo `inherit`	výchozí	procenta	dědí se	použití na prvky	média
speak-header	`once` | `always`	`once`	nelze	ano	prvky, které obsahují informaci o záhlaví tabulky	zvuková

Každý jazyk má jiné nástroje na definování záhlaví tabulek pro nevizuální zařízení (v HTML 4 se např. používají atributy **headers**, **scope** a **axis**) — CSS k nim doplňuje vlastnost **speak-header**, která určuje, zda mají být čtena pouze jednou, nebo před každou buňkou, která k tomuto záhlaví patří.

Hodnota **once** způsobí, že každé záhlaví bude čteno pouze jednou, jako uvození před skupinou buněk. S hodnotou **always** bude záhlaví zopakováno před každou související buňkou.

Např. část tabulky bude se **speak-headers:once** přečtena takto:

```
Pobočka 1 - srpen 2002 - náklady: 120,34 - výnosy: 150,30 - zisk: 30,04
```

S hodnotou **speak-headers:always** ale bude stejný úsek této tabulky přečten např. takto:

```
Pobočka 1 - srpen 2002 - náklady: 120,34
Pobočka 1 - srpen 2002 - výnosy: 150,30
Pobočka 1 - srpen 2002 - zisk: 30,04
```

5 Praxe

V této kapitole si představíme několik tipů pro efektivní používání CSS a řešení nejčastějších problémů, s nimž se webdesigner může setkat při používání kaskádových stylů.

5.1 Tipy a triky v CSS

5.1.1 Pozor na verze

Při návrhu vzhledu dokumentů si dobře vyberte podmnožinu CSS, kterou budete používat. Moderní prohlížeče v současnosti podporují prakticky úplné CSS1, některé z nich i podstatnou část CSS2. Situace se ale neustále a rychle mění. V budoucnu, po nástupu CSS3 a nových verzí prohlížečů, bude úplně jiná. Vždy ale patrně bude platit, že některé vlastnosti někde fungují bez problémů, někde chybně a jinde vůbec ne. Dobrý webdesigner se vždy dokáže přizpůsobit aktuální situaci a dává přednost třeba méně efektivním, ale obecněji použitelným řešením.

Měli bychom proto používat takové vlastnosti, které budou podporovány u co největší skupiny budoucích uživatelů. Ostatní by měly být užity jen tak, aby v prohlížečích, kde podporovány nejsou, nezhoršily přístupnost a použitelnost dokumentu. Jistě, některé vlastnosti bývají občas podporovány chybně — špatná interpretace je chybou prohlížeče, co bychom se o ni starali… Jenže pokud alternativním postupem můžeme takové všeobecně známé chyby *obejít* a přitom mít stále korektní a bezchybné CSS, je to určitě výhoda — rozšíříme tím počet spokojených návštěvníků, kteří se na naše stránky zase rádi vrátí.

Pro rozhodování o nejvhodnějších postupech mohou být dobrým pomocníkem tabulky podpory CSS v prohlížečích, které jste mohli zaregistrovat v předchozí kapitole u popisu jednotlivých vlastností. A některé z ověřených postupů najdete i v dalším textu.

5.1.2 Typ dokumentu a CSS

Dříve opomíjená *specifikace typu dokumentu* (DTD) v jeho záhlaví dnes nabývá zvláštního významu. I když pomineme, že v nových verzích (X)HTML je již povinná (příkaz DOCTYPE) a definuje normu, podle níž se má dokument zpracovávat, má také význam další, neméně podstatný. Výrobci nejpoužívanějších prohlížečů se totiž rozhodli vyjít vstříc autorům v období přechodu k přísným standardům — místo aby nové prohlížeče rázem přestaly tolerovat chyby a zastaralé konstrukce v dokumentech, pracují v několika různých režimech, které se aktivují právě podle použitého typu dokumentu.

V prohlížeči MSIE je **nestandardním** *(přechodovým)* **režimem** způsob zpracování dokumentů, který byl používán ještě v MSIE 5.5 pro Windows. Prohlížeč přehlíží lecjaké chyby ve stránkách, počítá chybně rozměry prvků v CSS (viz dále), tabulky přeruší řetězec dědičnosti, skripty i CSS neinteragují dobře se stromem dokumentu atd. Ve **standardním režimu** je od tohoto zastraralého zpracování upuštěno a prohlížeč se daleko více řídí příslušnými specifikacemi (X)HTML, CSS atd. Podřízení se standardům není ještě úplné, ale rozdíl je přesto výrazný. Tyto dva režimy jsou používány v MSIE od verze 6 pro Windows a od verze 5 pro MacOS.

Prohlížeče na bázi Mozilly mají režimy dokonce tři. Zpracování v **nestandardním** režimu se podobá chování Netscape 4.x, v režimu **plně standardním** jsou specifikace dodrženy prakticky bezvýhradně. Třetí režim, tzv. **převážně standardní** je mezistupněm mezi nimi — s co nejvyšší kompatibilitou se staršími konstrukcemi v dokumentech se současně snaží maximálně dodržovat standardy. Nutno je podotknout, že *standardní režim* v MSIE má k tomuto režimu podstatně blíže než k režimu *plně standardnímu*.

Následující tabulka uvádí režimy, které se aktivují podle použitého typu dokumentu v prohlížečích NN6/Mozilla (verze 1.1 a novější), MSIE 6 a novější ve Windows a MSIE 5.x a novější v MacOS.

Typ dokumentu podle DTD (Document Type Declaration uvedená v DOCTYPE)	NN6/Moz	IE6/Win	IE5/Mac
DOCTYPE chybí	ne-std	ne-std	ne-std
Před-HTML 4.0, např. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">	ne-std	ne-std	ne-std
HTML 4.x Strict, např. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	**STD**	STD	STD
HTML 4.x Transitional bez udání URL, např.: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">	ne-std	ne-std	ne-std
HTML 4.0 Transitional s URL, např.: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	ne-std	STD	STD
HTML 4.01 Transitional s URL, např.: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	p-STD	STD	STD
XHTML 1.0 Transitional bez deklarace XML, např.: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-	p-STD	STD	STD
XHTML 1.0 Transitional s deklarací XML, např.: <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-	p-STD	ne-std	STD
XHTML 1.0 Strict bez deklarace XML, např.: <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	STD	STD	STD
XHTML 1.0 Strict s deklarací XML, např.: <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">	**STD**	ne-std	STD
ISO HTML 2000, krátká forma, např.: <!DOCTYPE HTML PUBLIC "ISO/IEC 15445:2000//DTD HTML//EN">	**STD**	ne-std	ne-std
ISO HTML 2000, dlouhá forma, např.: <!DOCTYPE HTML PUBLIC "ISO/IEC 15445:2000//DTD HyperText Markup Language//EN">	**STD**	STD	STD
ISO HTML 1999, krátká forma, např.: <!DOCTYPE HTML PUBLIC "ISO/IEC 15445:1999//DTD HTML//EN">	**STD**	ne-std	ne-std
ISO HTML 1999, dlouhá forma, např.: <!DOCTYPE HTML PUBLIC "ISO/IEC 15445:1999//DTD HyperText Markup Language//EN">	**STD**	STD	STD

ne-std = nestandardní režim; p-STD = převážně standardní režim (Mozilla), STD = standardní režim (plně standardní režim v Mozi

Rozdíly mezi standardním a nestandardním režimem závisí na konkrétním prohlížeči. Obecně lze říci pouze to, že v režimu standardním se prohlížeče snaží formátovat stránky přibližně stejně, podle pravidel — a v nestandardním se vrací zpět k živelnému, nepředvídatelnému formátování „každý podle svého“.

5.1.3 Zapomeňte na „čtyřky“

Jak jste jistě zjistili z popisu podpory vlastností CSS v MSIE 4 (IE4) a především v Netscape 4.x (NN4), tyto prohlížeče mají s podporou kaskádových stylů opravdu velké problémy. Z praxe lze dokonce říci, že je prakticky nemožné vytvořit s CSS dobrý layout, který by v NN4 bez problémů fungoval. Jediné, nač se můžeme ještě trochu spolehnout, je definování stylu písma a barev. Na jakékoli pozicování či styl okrajů a rámečků je lépe raději zapomenout.

NN4 ani IE4 nemohou být považovány za prohlížeče podporující CSS. Nejlepším řešením je poskytnout komplexní CSS jen takovým prohlížečům, které s ním naloží rozumně — a NN4 i IE4 od těchto stylů „odříznout“ (viz dále).

Pokud náhodou zastaralé „čtyřky“ ve vaší cílové skupině stále převažují (např. firemní intranet, zahraniční komunita atd.), nerozpakujte se a vytvořte stránky starými postupy: formátované tabulkami a s minimálním CSS. Na to jsou tyto prohlížeče vynikající — roubovat moderní technologie na zastaralá zařízení je stejně špatné, jako používat zastaralé postupy na zařízeních moderních. Na obecně přístupné části Internetu ale mají novější prohlížeče drtivou převahu.

5.1.4 Připojování stylů

Nejčastěji se rozdělují prohlížeče na dvě skupiny:

- Staré prohlížeče se špatnou či žádnou podporou CSS. Především Netscape 4.x a MSIE 4 — vysoce chybové CSS, některé vlastnosti způsobí naprosté zhroucení vzhledu stránky. Takovému klientu můžeme poskytnout nanejvýš styl písma a barevnost, složitější styly by kvůli chybám v implementaci mohly způsobit vážné problémy.
- Moderní prohlížeče — MSIE 5+, NN6/Moz, Opera atd. Lze se již spoléhat na vyšší stupeň podpory standardů.

Podle své cílové skupiny se rozhodněte, zda má vůbec smysl pro staré prohlížeče vůbec nějaké styly vytvářet. Pokud je na vašem webu např. průměrně kolem 1 % uživatelů MSIE 4, je práce se zvláštním stylem pro tento prohlížeč jenom mrhání časem. Jejich podíl bude neustále klesat a dostatečným řešením je poskytnout jim pouze *čisté HTML*. O obsah nebudou uživatelé ochuzeni a jistě nepočítají s tím, že se starým prohlížečem budou mít k dispozici všechny nejmodernější technologie. Jedinou podmínkou je neumožnit takovému prohlížeči použít komplexnější styly, s nimiž by napáchal nedozírný zmatek.

Pokud je podíl starších prohlížečů přece jen vyšší a nechcete je „odbývat“ jen holým HTML, vytvořte jednoduchý styl, který ještě dokáží pochopit, a poskytněte jim ho. Ve tabulkách stylů pro ostatní prohlížeče pak tyto styly předefinujete a rozšíříte.

Pro distribuci různých verzí stylů mezi prohlížeče s různou podporou CSS můžeme využít jejich vlastních slabostí:

- Všechny prohlížeče zpracují styly definované ve značce `<style>` přímo ve stránce i načtené ve značce `<link>`
- NN4 podporuje načtení stylů pouze ve značce `<link>`, příkaz `@import` ignoruje.
- IE4 zná pouze syntaxi `@import url("URL")`, alternativní zápis `@import "URL";` ignoruje

Několikrát už zde bylo uvedeno, že styly v samostatných souborech jsou efektivnější. Nyní můžete objevit jejich další výhodu — chcete-li poskytnout styl všem prohlížečům, načtěte soubor značkou `<link>`. Má-li být od stylu „odstřižen“ NN4, použijte `@import` ve značce `<style>`. A nemá-li se ke stylu dostat ani IE4, použijte syntaxi `@import "URL"`.

Příklad:

```
<link rel="stylesheet" type="text/css" href="styl_pro_vsechny.css">
<style type="text/css">
<!--
   @import url("dodatky_pro_msie4.css");
   @import "styl_jen_pro_moderni_prohlizece.css";
-->
</style>
```

5.1.5 Alternativní styly

Styly, které jsou zapsány přímo v dokumentu a ty, které jsou načteny běžným způsobem značkami `<link>` a příkazy `@import` jsou **styly permanentní.** Všechny dohromady tvoří **základní styl** a použijí se vždy.

Tabulky načítané značkou `<link>` mohou být navíc pojmenovány (atributem `title`). Všechny tabulky, které mají stejný *název,* tvoří dohromady jeden **pojmenovaný styl.**

```
<link rel="stylesheet" title="Základní" type="text/css"
href="styl_vzhled.css">
<link rel="stylesheet" title="Základní" type="text/css"
href="styl_pismo.css">
```

Pojmenovaný styl s klíčovým slovem `alternate` tvoří **alternativní styl:**

```
<link rel="alternate stylesheet" title="Jiný styl" type="text/css"
href="jiny_styl.css">
```

Prohlížeč, který alternativní styly nepodporuje, bude takové značky ignorovat. V opačném případě zobrazí název stylu v nabídce. Uživatel si potom může vybrat styl, který mu vyhovuje.

Autor může použít jeden *pojmenovaný styl* jako **preferovaný** (bez klíčového slova `alternate`) a jeden nebo více *pojmenovaných stylů* alternativních. Stránka se na začátku zformátuje výchozím stylem (*základní* styly + styl *preferovaný,* je-li určen). Pokud si uživatel z nabídky vybere jeden z alternativních stylů, prohlížeč vypne všechny aktivní styly kromě *základních* (ty jsou aktivní vždy), k nim připojí tabulky tvořící zvolený styl a dokument znovu zformátuje. Např.:

```
<link rel="stylesheet" type="text/css" href="spolecne.css">
<link rel="stylesheet" title="Základní" type="text/css"
href="styl.css">
<link rel="alternate stylesheet" title="Styl A" type="text/css"
href="styl_A1.css">
<link rel="alternate stylesheet" title="Styl A" type="text/css"
href="styl_A2.css">
<link rel="alternate stylesheet" title="Styl B" type="text/css"
href="styl_B.css">
```

V tomto případě tvoří **základní styl** tabulka *spolecne.css* a všechny další styly definované přímo v dokumentu. Dále zde autor určil **preferovaný styl** (tabulka *styl.css*) a **dva alternativní styly:** „Styl A“ (tvoří jej tabulky *styl_A1.css* a *styl_A2.css*) a „Styl 2“ (tabulka *styl_B.css*). Ve vhodném prohlížeči bude mít uživatel k dispozici výběr ze stylů „Základní“, „Styl A“ a „Styl B“.

Pozn.: Mezi styly se lze také přepínat skriptem (dynamické HTML, viz dále).

5.1.6 Jak na rozměry prvků

Mnoho problémů při definování přesného vzhledu stránek působí rozdílný výpočet rozměrů. Ve starších verzích MSIE (a v *nestandardním režimu* prohlížečů novějších) vlastnosti `width` a `height` zahrnují i rozměr *výplně* a *rámečku* — zatímco v prohlížečích pracujících podle specifikace CSS `width`/`height` udávají pouze rozměr *obsahu*. Existuje naštěstí několik postupů, jak se s tím vypořádat a poskytnout uživatelům shodný vzhled stránek v jakémkoli prohlížeči.

První, nepříliš dobrou možností by mohlo být použít model Microsoftu. Stránka se navrhne s předpokladem, že prohlížeč počítá rozměry prvků chybně — u korektně pracujících prohlížečů se pak takové chování vynutí vlastností `box-sizing:border-box` (viz [4.3.4.3]). Podporuje ji však zatím velmi málo prohlížečů, ostatní (včetně MSIE 6) by stránku zformátovaly jinak, než chceme. Toto řešení je proto nevhodné; bude ale mít své místo, až bude vlastnost `box-sizing` všeobecně podporována.

Mohli byste také použít nějaký starší *typ dokumentu* (např. *HTML 3.2* nebo *HTML 4 Transitional*) a vynutit tak v prohlížeči *nestandardní režim* formátování. *Nestandardní* je však takový režim právě proto, že na platné standardy nebere ohled a používá algoritmy ze starých verzí prohlížeče. Přinutíte sice MSIE 6, aby stránku zformátoval stejně jako MSIE 5, rozhodně to ale nebude fungovat v prohlížečích ostatních. Některé více režimů nemají a pracují vždy korektně. Jiné se sice do *nestandardního režimu* přepnou, ale stránka se pak formátuje podle

úplně jiných pravidel než v MSIE. Nestandardní režim má vždy nestandardní a nepředvídatelné chování.

Nejvhodnějším a naprosto spolehlivým řešením je následující postup:

- Prvek, který nemá rámeček ani výplň (`padding:0`) má stejné rozměry v obou modelech a zobrazí se všude stejně.
- Prvek, který má rámeček a/nebo nenulovou výplň, uzavřete do neutrálního prvku (`div`). Vnějšímu prvku nastavíme požadované rozměry a nulový rámeček i výplň, prvku vnitřnímu pak požadovaný rámeček a výplň a rozměry `auto`. Vnější prvek se všude zobrazí stejně velký, vnitřní prvek se díky automatickým rozměrům přizpůsobí jeho velikosti.

Chcete-li např. zobrazit prvek `<p id="p3">`, který má být široký 100 px včetně 5px rámečku a 10px výplně (obsah tedy bude široký 70 px), podle CSS by měl být definován takto:

```
#p3 { border-width:5px; padding:10px; width:70px }
```

Ve starších verzích MSIE se však prvek zobrazí celkem 70 px široký (šířka obsahu bude jen 40 px). Naopak s definicí:

```
#p3 { border-width:5px; padding:10px; width:100px }
```

se prvek v ostatních prohlížečích (včetně MSIE 6 ve *standardním režimu*) zobrazí celkem 130 px široký. Výše uvedené řešení však zajistí shodné zobrazení ve všech prohlížečích:

```
#p3obal { width: 100px }
#p3 { border-width:5px; padding:10px }
...
<div id="p3obal">
   <p id="p3">
   ...
   </p>
</div>
```

Obdobným způsobem je vyřešena i kompatibilita výsledného formátování ukázkové stránky z úvodu knihy (ukázku najdete na Internetu jako *Příklad 2g*).

5.1.7 Násobné třídy

V (X)HTML může být hodnotou atributu `class` nejen jeden název třídy, ale také jejich seznam (položky se oddělují mezerami). Jeden prvek pak může být zařazen v několika třídách současně.

V jednom dokumentu pro to obvykle příliš využití není, pokud ale připravujete společnou tabulku stylů pro celý web, mohou být tyto *násobné třídy* poměrně užitečné. Např. budete chtít několik typů prvků lišících se stylem písma, které mají být na různých místech různě barevné. Můžete nadefinovat třídy pro každou z možných kombinací nebo použít kontextové selektory

tam, kde je to možné. Jednodušším a elegantnějším řešením však bude nadefinovat několika třídám styl určující pouze vzhled písma a jiným třídám styl definující jenom barevnost. Prvku pak přiřadíte obě třídy současně. Při formátování se pro prvek postupně použijí vlastnosti každé z použitých tříd, např.:

```
.zpravazdomova { font: bold 12pt/1.5 sans-serif }
.zpravazesveta { font: normal 12pt/1.5 serif }
...
.zpravadne { color:white; background:black }
.normalni  { color:blue; background:yellow }
.perlicka  { color:black; background:white }
...
<p class="zpravazdomova normalni">...
<p class="zpravazdomova zpravadne">...
<p class="zpravazesveta normalni">...
<p class="zpravazesveta perlicka">...
```

Možností použití násobných tříd je nepřeberně. Často může být výhodné připravit si sadu tříd upravujících jedinou vlastnost a s jejich pomocí vytvářet variace prvků:

```
.doprava { text-align: right }
.mensi { font-size: 90% }
.zvyrazneno { background: yellow }
.kurziva { font-style: italic }
...
<h1 class="kapitola zvyrazneno">...
<p class="poznamka doprava mensi">...
<p class="poznamka mensi kurziva zvyrazneno">...
/* atd. */
```

5.1.8 Využití dědičnosti a kaskády

Pravidla kaskády a dědičnost CSS jsou mohutným nástrojem. Důkladné pochopení a využití všech jejich možností vám usnadní přípravu stylů. Především u rozsáhlejších webů, kde rychle narůstá počet pravidel v CSS a velmi záhy se můžete v definicích stylů přestat orientovat, přinese vhodné využívání těchto nástrojů výrazné zjednodušení a zpřehlednění.

5.1.8.1 Dědičnost pracuje za vás

Mnoho prvků v celém dokumentu má shodné vlastnosti. Pokud je vlastnost dědičná, je zbytečné ji určovat znovu a znovu pro každý prvek. Postačí, když ji nadefinujeme jejich rodičovskému prvku (nejlépe prvku tvořícímu *kořen dokumentu,* např. `body`), ostatní prvky ji potom zdědí.

Až příliš často lze na webu potkat CSS typu:

```
p {
      font-family: 'Arial CE', 'Helvetica CE', Arial, sans-serif;
      color: black;
      margin: 1em 0;
}
h1 {
      font-family: 'Arial CE', 'Helvetica CE', Arial, sans-serif;
      color: black;
      margin: 2em 0 1em 0;
}
h2 {
      font-family: 'Arial CE', 'Helvetica CE', Arial, sans-serif;
      color: black;
      margin: 1.5em 0 1em 0;
}
/* atd. */
```

Přitom není nic snazšího, než nadefinovat dědičné vlastnosti `font-family` a `color` jen jednou — ostatní prvky hodnotu zdědí a výsledek bude stejný. CSS ale bude výrazně kratší a přehlednější:

```
body {
      font-family: 'Arial CE', 'Helvetica CE', Arial, sans-serif;
      color: black;
}
p  { margin: 1em 0 }
h1 { margin: 2em 0 1em 0 }
h2 { margin: 1.5em 0 1em 0 }
```

Pozn.: Některé starší prohlížeče chybně přerušují řetězec dědičnosti, především nedědí vlastnosti do tabulek. Jakkoli je to chyba prohlížeče, můžeme jim snadno vyjít vstříc a zajistit tak shodné zobrazení tím, že výchozí vlastnosti zopakujeme i pro tabulkové prvky:

```
body, th, td {
      font: 12pt/1.3 'Arial CE', 'Helvetica CE', Arial, sans-serif;
      color: blue;
}
```

5.1.8.2 Nedefinujte výchozí hodnoty

Každá vlastnost má svou výchozí hodnotu. Pokud je jednoznačně dána (např. nezávisí na volbě prohlížeče), je zbytečné ji v CSS znovu definovat. V tabulce stylů definujte pouze vlastnosti, které mají mít jinou hodnotu než *výchozí* nebo *zděděnou* od rodičovského prvku.

```
p {
      font-size: 100%;
      text-align: left;
      margin: 1.5em 0;
      width: auto;
      background: transparent;
}
```

Kromě vlastnosti `margin` všechny ostatní definice kopírují výchozí hodnoty pro prvek `p` a jsou zde úplně zbytečné. Definice stylu proto může být výrazně zkrácena:

```
p { margin: 1.5em 0 }
```

5.1.8.3 Využívejte kaskádu

Kaskáda CSS [3.7] mj. zajišťuje, že konkrétnější definice přepíší definice obecnější a že pravidla uvedená později nahradí pravidla předchozí. V tabulkách stylů to můžete často využívat, uspořit spoustu místa a své styly zpřehlednit.

Má-li více prvků několik shodných vlastností, nadefinujte je všechny najednou a pak pouze upravte odlišnosti:

```
h1, h2, h3, h4 {
      font: bold 200%/1.1 sans-serif;
      margin: 1em 0;
      color: green;
}
h2 { font-size: 150% }
h3 { font-size: 120% }
h4 { font-size: 100% }
```

První definice sice nastaví všem nadpisům velikost `200%`, ale další definice to změní.

Používejte sdružené vlastnosti, i když se některá z dílčích hodnot odlišuje od ostatních — následně ji můžete upravit:

```
div.box {
      border: 1px solid black;
      border-left:none;
      }
```

Co nejvíce vlastností nadefinujte s co nejobecnějšími selektory, s konkrétnějšími selektory už jen upravujte detaily:

```
* { color: black; font-family: serif }
h1 { color: blue; font-weight: bold }
h1.clanek { color: red }
#kapitola h1.clanek { color: blue; background: yellow }
```

5.1.9 Přetékání plovoucích prvků z rámečků

Je-li v orámovaném prvku umístěn nějaký plovoucí prvek (např. obrázek zalomený v textu) a je vyšší než vnější prvek, bude přesahovat ven z rámečku. To je sice správné chování podle specifikace CSS, ovšem nikoli podle typografických pravidel — obtékané obrázky protínat rámeček nesmí.

Existuje celkem jednoduché řešení, které však není příliš korektní z hlediska sémantiky dokumentu. Přidává do něj totiž obsah, který má pouze formátovací účel. V tomto případě to lze ale tolerovat — jsou zde v ostrém protikladu malý prohřešek sémantický a velký prohřešek typografický, buď uděláte jeden, nebo druhý.

Na konec vnějšího prvku přidejte blok s `clear:both`. Tím se ukončí obtékání plovoucího prvku a rámeček se vykreslí až pod ním. Aby to fungovalo, nesmí však tento blok být prázdný (prázdné bloky se vůbec nemusí zobrazit) — to je ten přidaný obsah navíc. Obsahem ale může být nějaký neviditelný prvek (průhledný 1px obrázek, neviditelný znak atd.). Navíc je vhodné, aby tento prvek byl co nejnižší (výška 1px, nulové okraje atd.). Např.:

```
div.ukonciobtekani { clear: both; font-size: 1px; height: 1px }
...
<p class="srameckem">
      <img class="plovoucivlevo" src="foto.jpg" ... />
      text text text
      <div class="ukonciobtekani"> </div>
</p>
```

5.1.10 Ladění CSS

Pokud testujete své styly a nejde vám do hlavy, co že se to vlastně zobrazuje a proč, použijte ladicí styly. Jednoduše do CSS dočasně přidejte pravidla zvýrazňující některé informace, které vám z pohledu na stránku nejsou jasné. Až skončíte ladění, nezapomeňte je z tabulky stylů zase smazat.

Velmi užitečné je například nechat si orámovat všechny prvky, aby byly zřetelně vidět jejich rozměry, pozice a vzájemné vnoření. Stačí do CSS přidat pravidlo:

```
* { border: 1px solid black !important }
```

Další možností je například zvýraznit si všechny prvky určitého typu kontrastním pozadím:

```
p { background: yellow !important }
```

Příkaz `!important` zajistí, že pravidlo bude mít přednost před všemi ostatními definicemi použité vlastnosti.

5.1.11 CSS a dynamické HTML

Všechny vlastnosti CSS je možné dynamicky měnit. V klientech podporujících *Objektový model dokumentu (DOM)* jsou dostupné jako vlastnosti objektu `style` každého prvku, např. `MujPrvek.style.fontSize`. Některé starší prohlížeče používají ještě vlastní objektový model, moderní prohlížeče se už ale jednotně podporují standard DOM.

Vlastnosti objektu `style` však neobsahují vypočítané hodnoty jednotlivých vlastností CSS, tak jak vzniknou v procesu kaskády CSS. To by bylo příliš náročné (vypočítané hodnoty lze zjistit jinými cestami, např. metodou `document.defaultView.getComputedStyle()` z DOM2 nebo z nestandardních objektů MSIE `currentStyle` a `runtimeStyle`). V objektech `style` jsou proto pouze hodnoty vlastností odpovídající přímým stylům (ty, které jsou zapsány přímo ve značce), neboť ty mají nejvyšší prioritu a nehrozí konflikt s předefinováním v rámci kaskády. Např.:

```
p { color:blue }
p.pozn { color:red }
...
<p id="p1">...
<p id="p2" class="pozn">...
<p id="p3" class="pozn" style="color:green">...
```

Prvek `#p1` bude mít modrý text (vyhovuje pravidlu pro `p`), prvek `#p2` bude červený (vyhovuje prvnímu i druhému pravidlu, druhé má v kaskádě vyšší prioritu). Prvek `#p3` vyhovuje také oběma pravidlům, použitý *přímý styl* však má nejvyšší prioritu a bude tedy zobrazen zeleně.

Z hlediska skriptu však pouze třetí prvek má definovánu vlastnost `prvek.style.color`, a ta má hodnotu `'green'`. Ostatní dva prvky přímý styl nepoužívají, takže jejich vlastnost `prvek.style.color` bude nedefinovaná. Pokud jim však skriptem hodnotu přiřadíme, bude to mít stejný efekt, jako bychom zapsali pravidlo do jejich značky. Příkaz Javascriptu:

```
document.getElementById('p1').style.color = 'orange';
```

má tedy z hlediska kaskády CSS stejný efekt jako zápis:

```
<p id="p1" style="color:orange">...
```

Přiřazováním hodnot vlastnostem objektu `style` tedy můžeme ve skriptech definovat vlastnosti prvků (s nejvyšší prioritou) a dynamicky měnit formátování dokumentu, vytvářet animace atd. Budeme-li nějakou vlastnost měnit skriptem, měli bychom jí vždy na začátku přiřadit nějakou výchozí hodnotu, nebo zjistit (výše uvedenými metodami) hodnotu aktuální.

```
mujPrvek = document.getElementById('prvek123');
mujPrvek.style.top = '120px';
Animuj(mujPrvek);
```

Problematika dynamického HTML (DHTML) je velmi rozsáhlá a vydá přinejmenším na jednu samostatnou knihu. Zájemci by si měli prostudovat některou z řady dostupných publikací, které se jí důkladněji věnují. Zde pouze stručně zmíním několik drobností, které se přímo týkají tématu této knihy.

5.1.11.1 Práce s tabulkami stylů

DOM poskytuje celou řadu objektů a metod pro práci s kaskádovými styly. Je možné upravovat jednotlivá pravidla i dynamicky vytvářet celé tabulky stylů. Jedním ze základních stavebních kamenů je pole `document.styleSheets[]`, které obsahuje všechny tabulky stylů připojené k dokumentu (v pořadí, v němž jsou v dokumentu uvedeny). Každá položka v tomto poli má vlastnost `disabled`, jež může mít hodnotu `true` nebo `false` a řídí, zda je tato tabulka stylů aktivní, či nikoli. V prohlížečích, které pole `styleSheets` podporují, tak můžeme např. snadno přepínat styly dokumentu — na základě volby uživatele nebo použitého prohlížeče. Např.:

```
<link rel="stylesheet" type="text/css" href="zakladni.css" />
<script type="text/javascript">
<!--
...
if (document.styleSheets) {
   var css = null;
   if (volba1) css = "jiny_styl_01.css";
   if (volba2) css = "jiny_styl_02.css";
   if (volba3) css = "jiny_styl_03.css";
   if (css) {
      document.styleSheets[0].disabled = true;
      document.write('<style type="text/css">');
      document.write('@import "'+css+'";');
      document.write('</style>');
   }
}
...
-->
</script>
```

5.1.11.2 Animace a dynamické efekty

Pro dynamické efekty, které způsobují rychlé změny v dokumentu (např. animace, efekty při pohybu kurzoru atd.), bychom měli zásadně **používat jen ty vlastnosti, jejichž změna nezpůsobí přeformátování celého dokumentu.**

Např. pro skrytí a zobrazení prvku můžeme použít vlastnosti `visibility` i `display`. Obě mají své výhody i nevýhody a každá dělá něco jiného. Pokud ale dynamicky změníme hodnotu vlastnosti `display`, celý dokument se přeformátuje, protože tato vlastnost výrazně ovlivňuje nejen vzhled prvku samotného, ale i vzhled všech prvků okolních. Kdybychom ji změnili častěji

po sobě, může to způsobit nepředvídatelné nepříjemné efekty, případně i *pád* prohlížeče — uživatelův počítač prostě nemusí tolik akcí tak rychle po sobě stihnout zpracovat.

Vlastnost `visibility` naproti tomu okolní prvky nijak neovlivní. Můžeme ji dynamicky měnit často, dokument se nebude přeformátovávat a změna bude zobrazena velmi rychle. Měli bychom jí tedy vždy dávat přednost a vlastnost `display` používat jen výjimečně, tam kde jiné řešení není možné.

Prakticky totéž platí pro animaci pohybu prvků. Vykreslení absolutně pozicovaných prvků je podstatně náročnější, než zobrazení prvků v normálním toku. Chceme-li nějaký prvek rozpohybovat, měli bychom proto **dát přednost relativnímu pozicování** — dynamická změna jeho *souřadnic* bude výrazně rychlejší než stejná animace u prvků pozicovaných absolutně.

5.1.12 Dynamická obrázková tlačítka v CSS

Snad vůbec nejčastěji se skripty na stránkách používají pro dynamická tlačítka, měnící vzhled při pohybu kurzoru (změna obrázku při událostech `onmouseover` a `onmouseout`). Prostředky CSS můžete dosáhnout obdobného efektu, aniž byste museli použít skript.

Pokud si vystačíme s pouhou změnou barevnosti prvku, je řešení velmi snadné, např.:

```
a.menu {
      display: block;
      width: 130px;
      background: red;
      color: white
}
a.menu:hover {
      background: black;
      color: yellow
}
```

Stejným postupem lze ale vyřešit i obrázková tlačítka — s výhodou zde využijeme *obrázky na pozadí.* Toto řešení má navíc jeden příjemný vedlejší efekt: všechny obrázky se načítají pouze tehdy, použije-li uživatel náš styl. V opačném případě zůstanou odkazy textové a v dokumentu se nenačítají data, která uživatel nechce či nemůže využít.

Pokud chcete např. vytvořit grafické menu, v kódu stránky jej pouze označíte strukturálními značkami — tak aby se menu přehledně zobrazilo i bez použití stylů, např.:

```
<ul id="menu">
      <li><a href="...">Odkaz 1</a></li>
      <li><a href="...">Odkaz 2</a></li>
      ...
      <li><a href="...">Odkaz 3</a></li>
</ul>
```

Vhodným stylem pak zajistíte zobrazení v požadované podobě. Pokud chcete např. z odkazů vytvořit červená tlačítka s výraznou žlutou šipkou vpravo, která se mají po *najetí kuzoru* změnit na zelená, zobrazit jako stisknutá, text posunout mírně doprava a šipka změnit na kolečko, může styl vypadat např. takto:

```
#menu { margin:0; padding:0 }
#menu li {
      list-style-type: none;
      margin:0; padding:0;
      width: 150px;
}
#menu li a {
      display: block;
      margin:0; padding: 1ex 60px 1ex 1ex;
      color: black;
      font-size: 14pt;
      background: red url('sipka.gif') right center no-repeat;
      border: 2px outset silver;
}
#menu li a:hover {
      padding-left: 2ex;
      background: green url('kolecko.gif') right center no-repeat;
      border-style: inset;
}
```

Odkazy `a` budou zobrazeny jako blok (vyplní celou plochu prvku `li`, který je zde jejich *omezujícím blokem*). Vlastnost `padding` zajistí, že obsah bude odsazen `1ex` zleva (po aktivaci `:hover` se zvětší na 2ex a text se posune) a `60px` zprava (prostor na šipku). Pozadí bude červené s obrázkem šipky umístěným vpravo na střed, ve stavu `:hover` se změní na zelené s obrázkem kolečka. Rámeček `outset` vytvoří dojem vystupujícího 3D tlačítka, ve stavu `:hover` bude styl rámečku `inset`, který dělá dojem tlačítka stisknutého.

Obdobným způsobem je vyřešeno i menu ve výsledné podobě ukázkové stránky z úvodu knihy (ukázku najdete na Internetu jako *Příklad 2g*).

5.2 CSS v českém prostředí

Nejčastější chyby v CSS vznikají při definování písem pro české dokumenty. Především autoři připravující stránky v prostředí MS Windows si často neuvědomí, jak jsou spravována písma na jiných platformách. Nevhodným předpisem písem mohou znehodnotit obsah uživatelům, kteří používají např. MacOS nebo Linux, či kteří se na jejich web připojují z ciziny.

5.2.1 Operační systémy a písma

5.2.1.1 Názvy písem

Od počátku nasazení počítačů do oblasti stolního publikování (DTP) byly doménou typografů především grafické stanice na bázi Unix (např. SGI) a počítače Apple Macintosh. Písma (*fonty*) zde používaná vycházela z písem klasické typografie a nejběžnější z nich se stala *základními písmy* těchto platforem (Helvetica, Times, Palatino, Garamond, Futura, Courier atd.).

Microsoft se při nástupu systému Windows chtěl z jistých důvodů vyhnout použití standardních profesionálních písem (většinou vytvořených společností Adobe) a jal se vytvářet písma vlastní. Vzniklo tak několik typograficky nepříliš podařených parafrází (Arial) a postupně i celá řada dalších „kopií“, odvozených od písem klasických, kvůli autorským licencím se od nich ale v detailech lišících a s mnohdy až komicky podobnými názvy — Times New (Times), Switzerland (Helvetica), Fujiyama (Futura), Aardvark (Aachen) atd.

Ve Windows tak *klasická písma* najdeme pouze na počítačích profesionálních grafiků, většina uživatelů používá pouze tato uměle vytvořená nová písma. Ta se naopak běžně nevyskytují na ostatních platformách (viz tabulku níže).

5.2.1.2 Písma ve Windows

V původních Windows 3.x měla každá znaková sada (západoevropská latinka, východoevropská latinka, cyrilice/azbuka, řečtina atd.) své vlastní písmo. Základem je zde obvykle západoevropská znaková sada, od níž jsou ostatní varianty odvozeny — např. písmo Verdana (západoevropské), Verdana CE (středoevropské), Verdana CY (cyrilika) atd.

Počínaje Windows 95 jsou již písma řešena odlišně. Všechny jazykové varianty jednoho písma byly sloučeny ve formátu podobnému standardu Unicode (není s ním však zcela kompatibilní) a jediné písmo tak v sobě může obsahovat hned několik znakových sad. Např. Arial ve Windows 98 již obsahuje všechny znaky pro západo- i středoevropské jazyky, azbuku, řeckou, arabskou a hebrejskou abecedu. Není to však pravidlem pro všechna písma — některá jich obsahují méně či dokonce obsahují znakovou sadu jedinou.

5.2.1.3 Písma na jiných platformách

Na ostatních platformách se stále často používá pro každou znakovou sadu samostatné písmo, rozlišené příponou za názvem — např. Helvetica, Helvetica CE, Helvetica CY atd. (MacOS). Písmo bez přípony obsahuje znaky pro západoevropské jazyky, na místě českých znaků s diakritikou jsou znaky úplně jiné. Český text je s takovým písmem prakticky nečitelný — např. slovo „výřečně“ se zobrazí třeba jako „v˘fieãnû“.

V Linuxu obvykle písma obsahují znaky západo- i středoevropských jazyků, není to však pravidlem; některá jsou pouze západoevropská. Písma ze základní instalace systému však zobrazení českých znaků podporují.

Modernější formáty písem, např. *Adobe OpenType* (dostupné na všech platformách), či písma v novém systému MacOS X a na platformách na bázi Unixu jsou často již vícejazykové a podporují několik jazykových sad současně. Standardizace a celosvětové sjednocení na formátu Unicode je však ještě v plenkách, proto zatím není možné se na všeobecně dostupná vícejazyková písma spoléhat.

5.2.1.4 Přehled typických písem

Následující tabulka zobrazuje přehled typických písem používaných na jednotlivých platformách. *Středoevropská písma* budou většinou nainstalována pouze v systému používajícím nějaký středoevropský jazyk. Součástí základní instalace západoevropské verze systému obvykle nebývají a znaky s českou diakritikou zde nebude možné zobrazit.

Windows 3.x				**MacOS**		
Times New CE	**středoevropský**		**serif**	Courier	západoevropský	monospace
Times New Roman	západoevropský		serif	**Courier CE**	**středoevropský**	**monospace**
Courier New CE	**středoevropský**		**monospace**	Geneva	západoevropský	sans-serif
Couier New Roman	západoevropský		monospace	**Geneva CE**	**středoevropský**	**sans-serif**
Arial CE	**středoevropský**		**sans-serif**	Helvetica	západoevropský	sans-serif
Arial	západoevropský		sans-serif	**Helvetica CE**	**středoevropský**	**sans-serif**
Windows 32bit				Monaco	západoevropský	monospace
Andale Mono	vícejazykový (Unicode)	latin, cyrilic, greek	sans-serif	**Monaco CE**	**středoevropský**	**monospace**
Arial	**vícejazykový (Unicode)**	**latin, cyrilic, greek, hebrew, arabic**	**sans-serif**	New York	západoevropský	serif
Arial Black	vícejazykový (Unicode)	latin, cyrilic, greek	sans-serif (tučné)	New York CE	středoevropský	serif
Arial Narrow *	vícejazykový (Unicode)	latin, cyrilic, greek	sans-serif (zúžené)	Palatino	západoevropský	serif
Bookmann OldStyle *	vícejazykový (Unicode)	latin, cyrilic, greek	serif	Palatino CE	středoevropský	serif
Comic Sans MS	vícejazykový (Unicode)	latin, cyrilic, greek	fantasy	Symbol	matem. symboly	—
Courier New	**vícejazykový (Unicode)**	**latin, cyrilic, greek, hebrew, arabic**	**monospace**	Techno	západoevropský	sans-serif (zúžené)
Garamond *	vícejazykový (Unicode)	latin, cyrilic, greek	serif	Techno CE	středoevropský	sans-serif (zúžené)
Georgia	vícejazykový (Unicode)	latin, cyrilic, greek	**serif**	Times	západoevropský	serif
Haettenschweiler *	vícejazykový (Unicode)	latin, cyrilic, greek	sans-serif (zúžené)	**Times CE**	**středoevropský**	**serif**
Impact	vícejazykový (Unicode)	latin, cyrilic, greek	sans-serif (zúžené)	Zapf Dingbats	ornamenty	—
Lucida Console *	vícejazykový (Unicode)	latin, cyrilic, greek	monospace	**Linux** – základní písma pro XWin (ISO 8859-2 a 8859-1)		
Monotype Sorts	ornamenty (Unicode)	—	—	**charter**		**serif**
MS Sans Serif **	vícejazykový (Unicode)	latin, cyrilic, greek, hebrew	sans-serif	**courier**		**monospace**
Symbol *	matem. symboly (Unicode)	—	—	**fixed**		**monospace**
Tahoma *	vícejazykový (Unicode)	latin, cyrilic, greek, hebrew, arabic	sans-serif	**helvetica**		**sans-serif**
Times New Roman	**vícejazykový (Unicode)**	**latin, cyrilic, greek, hebrew, arabic**	**serif**	**lucida**		**sans-serif**
Trebuchet MS	vícejazykový (Unicode)	latin	sans-serif	**lucidabright**		**serif**
Verdana	vícejazykový (Unicode)	latin, cyrilic, greek	sans-serif	**lucidatypewriter**		**monospace**
Webdings	ornamenty (Unicode)	—	—	**new century schoolbook**		**serif**
Windings	ornamenty (Unicode)	—	—	**times**		**serif**
** pouze Win 98+; ** pouze Win95 — latin = jazyky založené na latince (západo-, středoevropské i jiné)*				**utopia**		**serif**

Tučně jsou vyznačena *základní* písma, která jsou obvykle dostupná vždy (po základní instalaci operačního systému)

Musíme navíc počítat s faktem, že mnoho uživatelů MacOS používá některou z aplikací Microsoftu (MSIE, Word, Excel, Outlook), které při své instalaci přidají do systému sadu písem převzatých z Windows. Webdesignerům by to mohlo zjednodušit život, protože se tím jen rozšiřuje okruh uživatelů, kteří mají stejná písma.

To však platí jen na západ od Aše. V českém prostředí to situaci naopak velmi komplikuje, protože tyto aplikace se u nás nelokalizují, a tak nejsou lokalizována ani písma. Uživatelé MacOS proto mají často v systému písma Arial, Times New Roman, Impact, Verdana atd., jsou to ale písma v západoevropském kódování a neobsahují české znaky. Autoři WWW stránek tak mají o starost víc, protože musí zajistit, aby se k takovému písmu prohlížeč na MacOS nikdy nedostal — stránka by nebyla čitelná.

Existují i středoevropské varianty písem z Windows — většinou neoficiální, které si uživatelé pro svou platformu vytvářejí sami. Můžeme se tak setkat s písmy Arial CE, Verdana CE, Impact CE atd., není to ale pravidlem (písma pro MacOS viz Reference na konci knihy).

5.2.2 Definování písem v CSS pro české stránky

5.2.2.1 Obecná písma

Ve většině případů si vystačíme s obecnými písmy CSS `serif`, `sans-serif` atd. Konkrétní písmo, které se místo této *obecné rodiny* použije, si **obvykle** může uživatel nastavit sám. Na všech platformách tak bude použito správné písmo — volba je pouze na uživateli.

Jediná potíž je ve slově *obvykle*. Některé prohlížeče (především ty starší, jako Netscape 4.x, MSIE 4, některé verze Opery atd.) neumožňují uživateli písmo odpovídající jednotlivým *obecným rodinám* CSS nastavit. Pokud takový prohlížeč nemá k dispozici *vícejazyková* písma, není lokalizován ani nerozlišuje písma podle použitého jazyka a kódování dokumentu, použije se písmo nastavené výrobcem. A to často není středoevropské. Např. ve starších verzích Netscape na starších verzích MacOS se pro písma `serif` zásadně používá písmo Times (západoevropské), pro `sans-serif` Helvetica atd. a není možné to nijak ovlivnit. Úmyslně zde zdůrazňuji slova *starší* — novější prohlížeče i novější systémy již tento problém řeší (umožní písmo nastavit, vyberou správnou verzi písma podle použitého kódování nebo mají k dispozici vícejazyková písma).

V naprosté většině případů s *obecnými rodinami* písma problém není a můžeme je použít.

```
body { font-family: serif }
h1,h2,h3,h4 { font-family: sans-serif }
```

Podíl prohlížečů, které by zde použily nevhodné písmo, je velmi malý (obvykle pod 1 %) a stále klesá. Na druhé straně je k nim třeba připočíst i uživatele, kteří nepoužívají středoevropský systém a nemají česká nebo vícejazyková písma vůbec k dispozici (např. někteří návštěvníci ze zahraničí). Můžeme ale uživatelům nabídnout **alternativní verzi stránek bez diakritiky,** třeba pomocí automatického překódování na serveru. Návštěvník pak dostane stránky, které obsahují

pouze znaky ASCII a jazyková verze písma nebude mít na jejich zobrazení vliv. Tento postup je patrně nejlepším (ne-li jediným možným) řešením pro všechny, kteří nemohou české znaky zobrazit, a pokud jej použijeme, můžeme bez výčitek používat i *obecné rodiny* písem.

5.2.2.2 Podrobné definice

Pokud grafický návrh našeho webu upřednostňuje použití konkrétních písem, můžeme je definovat v CSS podrobněji. Design stránek však nikdy nesmí být na použitém typu písma závislý — uživatel nemusí mít žádné z námi definovaných písem k dispozici a musíme počítat i s tím, že každý má možnost změnu písma v prohlížeči zakázat. V CSS proto vždy **písma pouze doporučujeme.** Neexistuje možnost je vynutit a design stránky na to musí být připraven.

Seznam písem pro vlastnost `font-family` musíme připravovat citlivě, s ohledem na algoritmus jeho zpracování [3.8.8.4] a v českém prostředí především vzhledem k odlišnostem jednotlivých platforem (viz výše). I když připravujeme stránky ve Windows a víme, že naprostá většina jejich návštěvníků používá tento systém, nesmíme ignorovat ani ty ostatní, byť by jejich podíl byl sebemenší. Jen když jsme si naprosto jisti, že všichni uživatelé budou používat jednotnou platformu (např. na firemním intranetu), můžeme definice písem zjednodušit.

Pokud definujeme písma pro veřejně přístupné české dokumenty, je důležité postupovat podle následujících pravidel. Vycházíme z předpokladu, že většina návštěvníků bude používat systém Windows 95 a vyšší (v době psaní této knihy tvořili 80 až 90 % uživatelů Internetu v ČR):

- Vybereme si **primární vícejazykové písmo** (pro Windows), kterým se má text přednostně zobrazovat (např. Georgia).
- Pokud *primární písmo* není *základní* (tj. dostupné po standardní instalaci Windows, v tabulce výše jsou vyznačena tučně), vybereme jedno či více **náhradních vícejazykových písem** (použijí se, pokud primární písmo nebude k dispozici). Alespoň jedno z nich by mělo být *základní* (např. Times New Roman).
- Vybereme alternativní **písma pro Windows 3.x (Win3x).** Musíme vybírat jen ze *středoevropských písem* (mají přívlastek CE). Může to být opět písmo hlavní a jedno (či několik) písem náhradních; alespoň jedno z nich musí být *základní* (např. Times New CE).
- Vybereme alternativní **písma pro MacOS.** Stejně jako v předchozím případě to jsou písma s přívlastkem CE a alespoň jedno z nich musí být *základním písmem* v MacOS (např. Times CE).
- Vybereme alternativní **písma pro Linux/Unix.** Zde jsou bez přípony (písma jsou obvykle k dispozici v západo- i středoevropské variantě). Opět alespoň jedno z nich musí být základním písmem.

Z této sady písem sestavíme seznam pro CSS (připomeňme, že názvy písem, obsahujících mezeru, musí být uzavřeny v uvozovkách):

1. Jako první uvedeme **primární vícejazykové písmo doplněné o přívlastek CE.** Je zde pro případ, že cílová platforma bude mít náhodou k dispozici CE-variantu primárního písma. Např. `'Georgia CE'`.
2. Pokud písmo podle bodu (1) není *základním písmem* ve Windows 3.x (jako je např. Arial CE), přidáme **písma pro Win3x.** *Základní písmo* musí být uvedeno jako poslední. Např.: `'Times New CE'`.
3. Pokud žádné z doposud uvedených písem není *základním písmem* v MacOS, doplníme **písma pro MacOS.** Jako poslední musí být uvedeno jedno ze základních písem MacOS. Např.: `'New York CE'`, `'Times CE'`.
4. Přidáme **primární vícejazykové písmo** a za něj případná **náhradní vícejazyková písma** (písma pro Windows). Jedno ze *základních písem* Windows musí být uvedeno jako poslední. Např. `Georgia`, `'Times New Roman'`. Pokud postačí, aby prohlížeč jako náhradu použil *obecnou rodinu* CSS, náhradní písma uvádět nemusíme.
5. Pokud žádné dosud z uvedených písem není základním písmem v Linuxu, přidáme **písma pro Linux.** Např. `utopia`, `times`. Pokud jako náhrada postačí *obecná rodina* CSS, tato písma přidávat nemusíme.
6. Jako poslední **vždy** doplníme obecnou rodinu písem CSS. Každá definice písma ji musí obsahovat jako poslední položku. Např. `serif`.

Vytvořený seznam pak vypadá např. takto:

```
'Georgia CE','Times New CE','New York CE','Times CE',Georgia,serif
```

Takto vytvořený seznam si ještě ověříme. Projdeme jej stejně, jako to udělá prohlížeč (zleva doprava) a zkusíme, které písmo se v případě jednotlivých platforem použije. Pokud narazíme na písmo, které může být interpretováno jako západoevropské na systému, který nepoužívá vícejazyková písma, je někde chyba. Seznam je v pořádku pouze v těchto případech:

- [1] Obsahuje pouze obecné rodiny CSS
- [2] Obsahuje pouze písma s přívlastkem CE následovaná obecnými rodinami CSS
- [3] Pokud seznam obsahuje písma bez přívlastku CE, před prvním z nich se musí nacházet jedno ze základních CE písem pro Win3x a jedno ze základních CE písem pro MacOS. Jako poslední musí být uvedena obecná rodina CSS.

5.2.3 Příklady definování písem v CSS

5.2.3.1 Časté chyby při používání písem

Následující ukázky představují příklady **chybně sestavených** seznamů písem.

```
CHYBNĚ: Arial, Helvetica, sans-serif
```

Porušuje pravidlo [3]. Písma Arial a Helvetica jsou na systémech Win3x a MacOS západoevropská (neobsahují české znaky), ale prohlížeč je použije (Arial je základní písmo ve Win3x, Helvetica v MacOS). Text pak nebude čitelný.

```
CHYBNĚ: Helvetica CE, Arial, sans-serif
```

Název Helvetica CE obsahuje mezeru, musí být proto v uvozovkách. Navíc je stále porušeno pravidlo [3] — chybí základní CE písmo pro Win3x. Prohlížeč by zde použil západoevropské písmo Arial.

```
CHYBNĚ: 'Arial CE', 'Helvetica CE', Arial
```

Chybí obecná rodina CSS. Pokud žádné z uvedených písem nebude k dispozici, text se nemusí vůbec zobrazit!

```
CHYBNĚ: 'Arial CE', Arial, 'Helvetica CE', sans-serif
```

Prohlížeč na MacOS může mít k dispozici západoevropské písmo Arial. Když jej použije, text nebude čitelný. Základní CE písmo proto musí být uvedeno dříve.

```
NEVHODNĚ: 'Arial CE', 'Helvetica CE', sans-serif, Arial
```

Obecnou rodinu `sans-serif` použije prohlížeč vždy, k písmu Arial se tedy nikdy nedostane a je zde zcela zbytečně. Obecné rodiny CSS musí být vždy až na konci seznamu.

```
SPRÁVNĚ: 'Arial CE', 'Helvetica CE', Arial, sans-serif
```

5.2.3.2 Příklady nejčastějších sad písem

V následující tabulce jsou uvedena nejčastější písma Windows používaná na webu, jim odpovídající písma z ostatních systémů a nakonec i výsledné seznamy písem pro CSS, optimalizované pro všechny platformy. Není sice možné tvrdit, že seznam bude bezvýhradně funkční, pokaždé se najdou výjimky (např. pokud si uživatel Win3x nebo MacOS smaže jedno ze základních písem) — to již ale jako autoři CSS ovlivnit nemůžeme.

Nezvýrazněné položky lze vynechat bez újmy na korektnosti. **Tučně** označené jsou povinné, položky označené *kurzívou* by měly (ale nemusí) být uvedeny. Použitá základní písma pro jednotlivé systémy je však možné nahradit jinými, např. místo `'Helvetica CE'` lze použít třeba `'Geneva CE'` (MacOS), místo `times` můžeme použít třeba `charter` (Linux) atd.

Vícejazyčné písmo (Windows)		CE varianta	Windows 3.x	MacOS		Linux	CSS
primární	odpovídající základní	(primární + CE)	odpovídající základní	náhradní	odpovídající základní	základní	
Arial	**Arial**	'Arial CE'	**'Arial CE'**	**'Helvetica CE'**	**'Helvetica CE'**	helvetica	**sans-serif**
Verdana	Arial	'Verdana CE'	**'Arial CE'**	'Lucida Grande CE'	**'Helvetica CE'**	lucida	**sans-serif**
Tahoma	Arial	'Tahoma CE'	**'Arial CE'**	**'Helvetica CE'**	**'Helvetica CE'**	helvetica	**sans-serif**
'Times New Roman'	**'Times New Roman'**	'Times New Roman CE'	**'Times New CE'**	**'Times CE'**	**'Times CE'**	times	**serif**
'Courier New'	**'Courier New'**	'Courier New CE'	**'Courier New CE'**	**'Courier CE'**	**'Courier CE'**	courier	**monospace**
Georgia	'Times New Roman'	'Georgia CE'	**'Times New CE'**	'New York CE'	**'Times CE'**	times	**serif**
Impact	Arial	'Impact CE'	**'Arial CE'**	'Techno CE'	**'Helvetica CE'**	lucida	**sans-serif**
'Comic Sans MS'	Arial	'Comic Sans MS CE'	**'Arial CE'**	'Sand CE'	**'Helvetica CE'**	lucida	**fantasy**

Odpovídající definice písem — pořadí je nutno dodržet; **tučné** položky jsou povinné, ostatní lze vynechat. Položky zvýrazněné *kurzívou* je ale vhodnější uvést.	
bezpatkové	**font-family: sans-serif;**
patkové	**font-family: serif;**
neproporciální	**font-family: monospace;**
Arial	**font-family: 'Arial CE', 'Helvetica CE', Arial,** helvetica, **sans-serif;**
Verdana	**font-family:** 'Verdana CE', **'Arial CE'**, *'Lucida Grande CE'*, **'Helvetica CE', Verdana,** Arial, *lucida*, **sans-serif;**
Tahoma	**font-family:** 'Tahoma CE', **'Arial CE', 'Helvetica CE', Tahoma,** Arial, *lucida*, **sans-serif;**
Times New Roman	**font-family:** 'Times New Roman CE', **'Times New CE', 'Times CE', 'Times New Roman',** times, **serif;**
Courier New	**font-family: 'Courier New CE', 'Courier CE', 'Courier New',** courier, **monospace;**
Georgia	**font-family:** 'Georgia CE', **'Times New CE',** 'New York CE', **'Times CE', Georgia,** times, **serif;**
Impact	**font-family:** 'Impact CE', **'Arial CE',** *'Techno CE'*, **'Helvetica CE', Impact,** *lucida*, **sans-serif;**
Comic Sans MS	**font-family:** 'Comic Sans MS CE', **'Arial CE',** *'Sand CE'*, **'Helvetica CE', 'Comic Sans MS',** *lucida*, **fantasy;**

Ukázková tabulka stylů pro HTML 4.0

Následující tabulka (uvedená ve specifikaci CSS 2.1) popisuje typické formátování dokumentů jazyka HTML 4.0. Vývojáři ji mohou použít jako *výchozí tabulku stylů* v implementacích prohlížečů.

```
          ADDRESS, BLOCKQUOTE, BODY,
          DD, DIV, DL, DT, FIELDSET,
              FORM, FRAME, FRAMESET,
            H1, H2, H3, H4, H5, H6,
           IFRAME, NOFRAMES, OBJECT,
          OL, P, UL, APPLET, CENTER,
                 DIR, HR, MENU, PRE    { display: block }
                                LI     { display: list-item }
                              HEAD     { display: none }
                             TABLE     { display: table }
                                TR     { display: table-row }
                             THEAD     { display: table-header-group }
                             TBODY     { display: table-row-group }
                             TFOOT     { display: table-footer-group }
                               COL     { display: table-column }
                          COLGROUP     { display: table-column-group }
                            TD, TH     { display: table-cell }
                           CAPTION     { display: table-caption }
                                TH     { font-weight: bolder; text-align: center }
                           CAPTION     { text-align: center }
                              BODY     { padding: 8px; line-height: 1.33 }
                                H1     { font-size: 2em; margin: .67em 0 }
                                H2     { font-size: 1.5em; margin: .83em 0 }
                                H3     { font-size: 1.17em; margin: 1em 0 }
     H4, P, BLOCKQUOTE, UL, FIELDSET,
               FORM, OL, DL, DIR, MENU     { margin: 1.33em 0 }
                                H5     { font-size: .83em; line-height: 1.17em; margin: 1.67em 0 }
                                H6     { font-size: .67em; margin: 2.33em 0 }
    H1, H2, H3, H4, H5, H6, B, STRONG     { font-weight: bolder }
                         BLOCKQUOTE     { margin-left: 40px; margin-right: 40px }
            I, CITE, EM, VAR, ADDRESS     { font-style: italic }
             PRE, TT, CODE, KBD, SAMP     { font-family: monospace }
                               PRE     { white-space: pre }
                               BIG     { font-size: 1.17em }
                    SMALL, SUB, SUP     { font-size: .83em }
                               SUB     { vertical-align: sub }
                               SUP     { vertical-align: super }
                     S, STRIKE, DEL     { text-decoration: line-through }
                                HR     { border: 1px inset }
                OL, UL, DIR, MENU, DD     { margin-left: 40px }
                                OL     { list-style-type: decimal }
          OL UL, UL OL, UL UL, OL OL     { margin-top: 0; margin-bottom: 0 }
                           U, INS     { text-decoration: underline }
                           CENTER     { text-align: center }
                        BR:before     { content: "\A" }
                      ABBR, ACRONYM     { font-variant: small-caps; letter-spacing: 0.1em }
                            A[href]     { text-decoration: underline }
                             :focus     { outline: thin dotted invert }

/* začátek nastavení bidi (neměnit) */
                       BDO[DIR="ltr"]     { direction: ltr; unicode-bidi: bidi-override }
                       BDO[DIR="rtl"]     { direction: rtl; unicode-bidi: bidi-override }
```

```
                        *[DIR="ltr"]    { direction: ltr; unicode-bidi: embed }
                        *[DIR="rtl"]    { direction: rtl; unicode-bidi: embed }
          /* Blokové prvky v HTML4 */
   ADDRESS, BLOCKQUOTE, BODY, DD, DIV,
       DL, DT, FIELDSET, FORM, FRAME,
     FRAMESET, H1, H2, H3, H4, H5, H6,
   IFRAME, NOSCRIPT, NOFRAMES, OBJECT,
   OL, P, UL, APPLET, CENTER, DIR, HR,
MENU, PRE, LI, TABLE, TR, THEAD, TBODY,
 TFOOT, COL, COLGROUP, TD, TH, CAPTION    { unicode-bidi: embed }
/* Konec nastavení bidi */

                      @media print {
                             @page      { margin: 10% }
               H1, H2, H3, H4, H5, H6    { page-break-after: avoid; page-break-inside: avoid }
                      BLOCKQUOTE, PRE    { page-break-inside: avoid }
                          UL, OL, DL     { page-break-before: avoid }
                                 }

                      @media aural {
               H1, H2, H3, H4, H5, H6    { voice-family: paul, male; stress: 20; richness: 90 }
                                  H1     { pitch: x-low; pitch-range: 90 }
                                  H2     { pitch: x-low; pitch-range: 80 }
                                  H3     { pitch: low; pitch-range: 70 }
                                  H4     { pitch: medium; pitch-range: 60 }
                                  H5     { pitch: medium; pitch-range: 50 }
                                  H6     { pitch: medium; pitch-range: 40 }
                             LI, DT, DD  { pitch: medium; richness: 60 }
                                  DT     { stress: 80 }
                           PRE, CODE, TT { pitch: medium; pitch-range: 0; stress: 0; richness: 80 }
                                  EM     { pitch: medium; pitch-range: 60; stress: 60; richness: 50 }
                              STRONG     { pitch: medium; pitch-range: 60; stress: 90; richness: 90 }
                                 DFN     { pitch: high; pitch-range: 60; stress: 60 }
                            S, STRIKE    { richness: 0 }
                                    I    { pitch: medium; pitch-range: 60; stress: 60; richness: 50 }
                                    B    { pitch: medium; pitch-range: 60; stress: 90; richness: 90 }
                                    U    { richness: 0 }
                               A:link    { voice-family: harry, male }
                             A:visited   { voice-family: betty, female }
                              A:active   { voice-family: betty, female; pitch-range: 80; pitch: x-high
}
```

Reference

5.2.4 Použité příklady a ukázky

Všechny příklady a ukázky zmíněné v této knize naleznete na adrese:

http://www.knihy.cpress.cz/K0798/priklady

5.2.5 Důležité adresy a dokumenty

Seznam technických dokumentů W3:	http://www.w3.org/TR
Specifikace CSS1:	http://www.w3.org/TR/REC-CSS1
Specifikace CSS2:	http://www.w3.org/TR/REC-CSS2
(Předběžná) specifikace CSS2.1:	http://www.w3.org/TR/CSS21
Připravované nové verze CSS:	http://www.w3.org/Style/CSS/current-work
Škola CSS na W3C:	http://www.w3.org/Style/CSS
HTML 4.0:	http://www.w3.org/TR/REC-html40/
XHTML 1.0:	http://www.w3.org/TR/xhtml1/
XML 1.0:	http://www.w3.org/TR/REC-xml
URI:	http://www.ietf.org/rfc/rfc2396.txt
sRGB:	http://www.w3.org/Graphics/Color/sRGB.html
ISO 10646:	http://directory.google.com/Top/Science/Reference/ /Standards/Individual_Standards/ISO_10646/
CE písma z Windows pro MacOS:	http://www.pixy.cz/downloads/MS_fonty_pro_Mac
Tipy a triky pro CSS:	http://cssblog.pixy.cz

Výše uvedené odkazy najdete i na adrese **http://www.knihy.cpress.cz/K0798/odkazy**