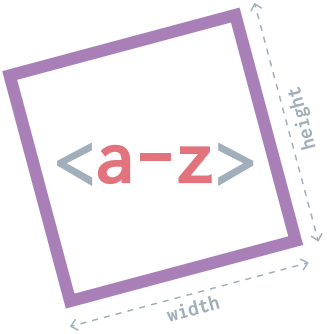
czechitas



Web od A do Z

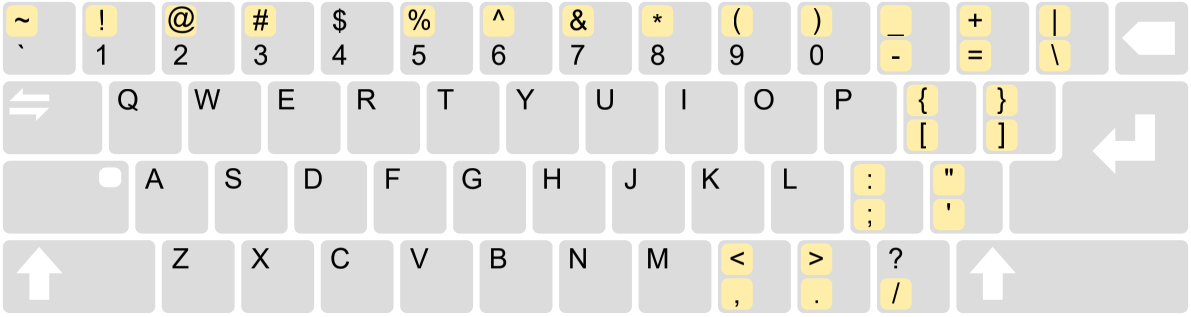
Podklady pro kurz

Zpracoval a lektoruje:
Luděk Roleček

Evropská unie
Evropský sociální fond
Operační program Zaměstnanost

Česká klávesnice

Anglická (americká) klávesnice



+	plus
*	hvězdička
-	minus
/	lomítko
%	procenta
@	zavináč
=	rovná se
!	vykřičník
' "	uvozovky
\	zpětné lomítko
,	čárka
.	tečka
:	dvojtečka
;	středník
#	mřížka
&	ampersand
|	svislítko
^	stříška
~	tilda
( )	závorky
{ }	složené závorky
[ ]	hranaté závorky
< >	ostré závorky
_	podtržítko

Atom - užitečné klávesové zkratky

Příkaz	Windows	Mac
Příkazová paletka	`ctrl-shift-p`	`shift-cmd-p`
Seznam otevřených souborů	`ctrl-b`	`cmd-b`
Hledání	`ctrl-f`	`cmd-f`
Najít další	`F3`	`cmd-g`

Editace

Zduplikovat řádek / vybrané řádky	`ctrl-shift-d`	`shift-cmd-d`
Vymazat řádek	`ctrl-shift-k`	`shift-cmd-k`
Přesunout řádek nahoru/dolů	`ctrl-↑` `ctrl-↓`	`ctrl-cmd-↑` `ctrl-cmd-↓`
Zakomentovat řádek	`ctrl-/`	`cmd-/`
Změnit odsazení řádku	`ctrl-[` `ctrl-]`	`cmd-[` `cmd-]`
Uzavřít aktuální tag	`ctrl-alt-.`	`cmd-alt-.`

Výběr / vícenásobné kurzory

Vybrat další shodný text	`ctrl-d`	`cmd-d`
Zrušit poslední výběr shodného textu	`ctrl-u`	`cmd-u`
Vícenásobný kurzor	`ctlr-click`	`cmd-click`
Sloupcový výběr	`ctrl-alt-↑` `ctrl-alt-↓`	`ctrl-shift-↑` `ctrl-shift-↓`

Úvod do HTML

- HTML - HyperText Markup Language = značkovací jazyk pro tvorbu HTML stránek.
- Aktuálně ve verzi HTML 5.
- Pomocí HTML vytváříme strukturu a obsah stránky.
- HTML říká webovému prohlížeči, co která část stránky znamená. Dává význam textu. Něco je nadpis, odstavec, seznam, odkaz, ...
- HTML prvky mají sice prohlížečem nastavený výchozí vzhled, ale jen s pomocí HTML "hezkou" stránku neuděláme.
- Vizuální vzhled stránky nastavujeme v CSS.

Elementy, tagy

- Strukturu stránky vytváříme pomocí speciálních značek, neboli tagů.
- Tag říká, jaký je sémantický význam textu uvnitř tagu, vytváří na stránce tzv. element.
- Tag je vždy uzavřen uvnitř ostrých závorek **<** a **>**.
- Většina tagů je párových - tj. je tvořena dvěma značkami; na začátku a na konci textu, který obalují.
  - Otevírací tag **<názevtagu>**
  - Uzavírací tag **</názevtagu>**
- Nezapomínejte tagy uzavírat!
- Některé tagy jsou nepárové, tj. mají pouze otevírací značku.

Obrázek
(nepárový tag)

```
<img src="obrazek.jpg">

<h1>Nadpis</h1>

<p>Odstavec textu...</p>
```

Nadpis

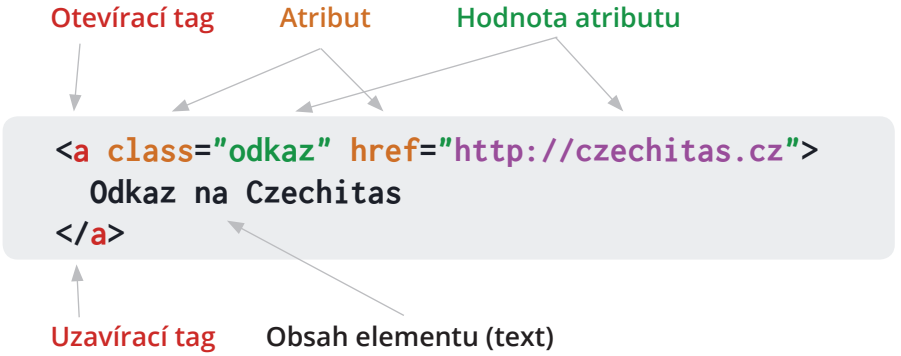
Lorem Ipsum

Odstavec

Eque praese nimpore laccata tiore-caest lantior rovidun turescim acea quid magnistrum fugia commodi tet dolupti cuptate ctorpores apictatist.

Atributy

- Uvnitř otevíracího tagu elementu mohou být uvedeny atributy.
- Atribut poskytuje o elementu doplňující informace, lze nastavovat různé vlastnosti elementu (např. u obrázku jeho velikost).
- Atribut má název a hodnotu. Hodnota je uvedena v uvozovkách.



- Některé atributy jsou specifické pro konkrétní tag. Např. `alt`, `width`, `height` pro obrázek, `href` pro odkaz, apod.
- Jiné atributy jsou obecné a jdou aplikovat na jakýkoliv tag: `class`, `id`, `lang`, `title`, `data`

- Názvy tagů i atributů uvádějte malými písmeny.

Zanořování tagů

- Tagy se mohou zanořovat jeden do druhého, tj. máme element uvnitř jiného elementu.
- Některé typy obsahu zanořování elementů přímo vyžadují (seznam, tabulka, a další).
- Dbejte správnou posloupnost otevíracích a zavíracích tagů.
- Co se první otevře, musí se zavřít jako poslední. Co se otevře jako poslední, musí se jako první zavřít.
- Doporučuje se zanořené tagy odsazovat, aby byla struktura dokumentu jasná na první pohled. Nezapomínejte tagy uzavírat!


```
<ul>
  <li>Položka seznamu</li>
  <li>Druhá položka seznamu</li>
  <li>Poslední <strong>tučná</strong> položka seznamu</li>
</ul>

<article>
  <h1>Nadpis článku</h1>
  <p>
    <a href="http://www.google.com"><img src="obrazek.jpg"></a>
  </p>
</article>
```

Příklad zanořování tagů v několika úrovních

Nejpoužívanější tagy

Nadpisy	`<h1>Text nadpisu</h1>`
	Pro nadpisy dalších úrovní používejte:
	`<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`

Odstavec	`<p>Odstavec textu</p>`
Odkaz	`<a href="www.adresa.cz">Text odkazu</a>` `<a href="mailto:muj@email.cz">Napiš mi na email</a>`

Seznam	`<ul>`
	`  <li>První položka seznamu</li>`
	`  <li>Druhá položka seznamu</li>`
	`</ul>`
	`<ul>` je seznam s odrážkami, `<ol>` je číslovaný seznam

Obrázek	`<img src="obrazek.jpg" alt="popis obrázku" width="200" height="100">`
Kontejnery div a span	`<div class="csstrida">` ... `</div>` Blokový prvek bez daného sémantického významu, slouží k obalování částí stránky pro potřeby layoutu apod. Inline obdoba je prvek `<span>`...`</span>`

Základní struktura HTML dokumentu

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <meta charset="utf-8">
    <title>Název stránky</title>
    <link rel="stylesheet" type="text/css" href="styly.css">
  </head>
  <body>

    <!-- zde je obsah stránky -->

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

- Typ dokumentu HTML5
- Začátek dokumentu
- Hlavička dokumentu
- Kódování stránky
- Název stránky (titulek okna/záložky)
- Externí CSS styly
- Konec hlavičky dokumentu
- Tělo dokumentu
- Zde bude kompletní obsah stránky
- Externí soubor s JavaScriptem, např. Google Analytics nebo vlastní skript. (nepovinné)
- Konec těla dokumentu
- Konec HTML souboru

- V sekci `<head>...</head>` popisujeme vlastnosti stránky.
- V sekci `<body>...</body>` uvádme obsah stránky.

Úvod do CSS

- CSS = Cascading Style Sheets.
- Aktuálně ve verzi CSS 3.
- Pomocí CSS vytváříme vzhled stránky. Nastavujeme styl prvkům stránky.
- Jde psát jako součást HTML stránky mezi tagy `<style>` `</style>`, ale je to nepraktické a používá se jen v odůvodněných případech.
- Většinou píšeme CSS do samostatného souboru, který do HTML připojíme uvnitř sekce `<head>`:

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

Syntaxe CSS

```
selektor {
  vlastnost: hodnota;
}
```

- Definice vlastností je uvedena ve složených závorkách.
- Vlastnost a hodnota jsou od sebe odděleny dvojtečkou.
- V rámci jednoho selektoru lze nastavit i více vlastností najednou. Jednotlivé vlastnosti od sebe oddělujeme středníkem.

```
p {
  color: red;
  background-color: yellow;  /* css komentář */
  font-size: 24px;
}
```

Příklad nastaví všem odstavcům v dokumentu barvu textu na červenou, barvu pozadí na žlutou a velikost písma na 24 px.

Základní CSS selektory

- Selektor vybere v dokumentu prvky, na který se budou aplikovat css vlastnosti uvnitř selektoru.
- Existují selektory mnoha typů - liší se podle toho, jakým způsobem prvky v dokumentu vybírají.

Selektor podle elementu

Selektor typu element vybere v dokumentu všechny tagy stejného jména. Selektor se zapisuje pouze jako název tagu (např. `p`)

```
<p>První odstavec</p>

<p>Druhý odstavec</p>
```

```
p {
  color: red;
}
```

Vyberou se všechny elementy `<p>`, oba odstavce v příkladu budou mít červenou barvu.

Selektor podle třídy

Vybere všechny prvky označené příslušnou třídou. Jde o nejpoužívanější typ selektoru. Zapisuje se jako název třídy s tečkou na začátku.

```
<p class="zvyrazneni">
  První odstavec
</p>

<p>Druhý odstavec</p>

<p class="zvyrazneni">
  Třetí odstavec
</p>
```

```
.zvyrazneni {
  color: red;
}
```

Vyberou se všechny elementy s třídou `zvyrazneny`, první a třetí odstavec budou mít červenou barvu.

Selektor podle ID

Vybere prvek s daným ID (může být pouze jeden v dokumentu). Selektor se zapisuje jako id s mřížkou na začátku. Snažíme se nepoužívat!

```
<div id="ramecek">
  text s rámečkem
</div>
```

```
#ramecek {
  border: 1px solid black;
}
```


Nejpoužívanější CSS vlastnosti

Textové vlastnosti

`font-size: 24px;`	velikost písma
`font-weight: bold;`	tloušťka písma (normal, bold)
`font-style: italic;`	sklon písma (normal, italic, oblique)
`font-family: "Arial", sans-serif;`	font; lze uvést wvvíce fontů oddělených čárkou (použijí se v daném pořadí není-li některý z fontů dostupný)
`line-height: 1.5;`	výška řádku (v násobcích velikosti písma), určuje rozestup mezi řádky textu
`text align: center;`	zarovnání textu (left, right, center, justify)

Barvy

Barvy lze uvádět jako pojmenované hodnoty (red, green, blue, atd.) nebo jako číselné hodnoty v hexadecimálním tvaru #123456, nebo jiné (rgba, hsla).

`color: red;`	barva textu
`background-color: #f5a376;`	barva pozadí

Box model

`width: 50%;`	šířka boxu (px, em, %)
`height: 30px;`	výška boxu (px, em, %)
`border: 2px solid red;`	rámeček kolem prvku ve formátu: `tloušťka typ-čáry barva` typ čáry: solid, double, dashed, dotted
`padding: 2em;`	odstup obsahu prvku od jeho rámečku
`margin: 20px;`	odstup prvku od ostatních prvků

Specifičnost a pořadí v CSS

Jeden prvek může mít v CSS nadefinovanou stejnou vlastnost na několika místech. V takovém případě se musí prohlížeč rozhodnout, kterou hodnotu použije.

- Na prvním místě rozhoduje tzv. specifičnost - pravidla pro výpočet specifičnosti selektoru jsou složitější, ale zjednodušeně platí, že ID selektor je důležitější (má vyšší specifičnost) než Class selektor a ten je důležitější než Element selektor.
- Pokud je specifičnost dvou selektorů stejná, rozhoduje pořadí v CSS - použije se poslední platná hodnota.

```
<p>odstavec</p>
```

```
p {
  color: red;
}

p {
  color: green;
}
```

Odstavec bude zelený.
Rozhoduje pořadí v CSS.

```
<p class="barva">odstavec</p>
```

```
.barva {
  color: red;
}

p {
  color: green;
}
```

Odstavec bude červený, i když má v CSS jako poslední nastavenou zelenou barvu.
Vyhraje totiž selector .barva, protože má větší specifičnost než selektor p.

Vlastnosti z více selektorů

Prvek získá vlastnosti ze všech CSS definic, jejichž selektorům vyhovuje. Např. v attributu class u elementu lze uvést více CSS tříd oddělených mezerou. Vlastnosti ze všech uvedených tříd se na zkombinují.

```
<p class="velky cerveny">
   tučný velký červený text
</p>
```

```
p { font-weight: bold; }
.velky { font-size: 36px; }
.cerveny { color: red; }
```


Box model

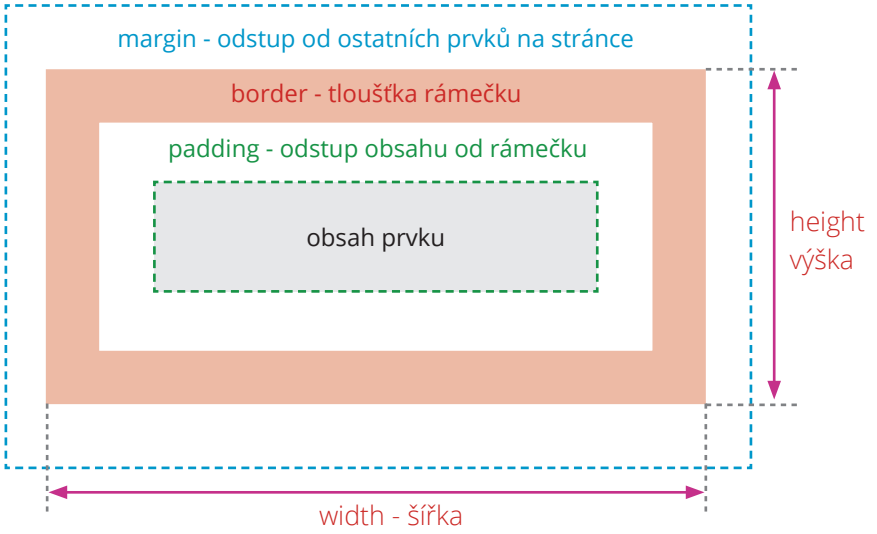
- Zjednodušeně: všechny prvky v HTML/CSS jsou obdélník (box).
- Když mluvíme v CSS o box modelu, mluvíme o vlastnostech, které ovlivňují umístění, tvar a chování tohoto obdélníku.

V CSS existují několik typů box modelu, které se chovají mírně odlišně. My budeme používat nejvíce praktický a intuitivní typ border-box, který ale bohužel není nastaven jako výchozí. Musíme ho v CSS vždy zapnout. Na začátek našeho CSS souboru vždy uvedeme:

```
*, :after, :before {
  box-sizing: border-box;
}
```

Tím zapneme požadovaný typ box modelu pro všechny prvky na stránce.

Box model - typ border-box

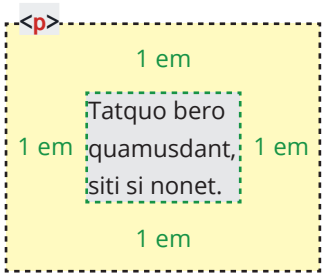


Margin a padding

- **Padding** = mezera mezi obsahem a okrajem/rámečkem prvku. Např. nechci aby byl text nalepený na rámeček kolem něj, chci aby byla z každé strany malá mezera.
- **Margin** = mezera mezi prvkem a ostatními prvky na stránce. Např. chci aby mezi nadpisem a textem odstavce byla mezera.

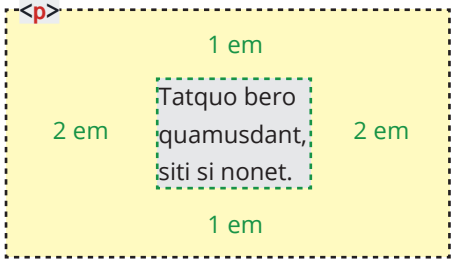
```
p {
  padding: 1em;
}
```

Uvedu-li jednu hodnotu, použije se na všechny strany.



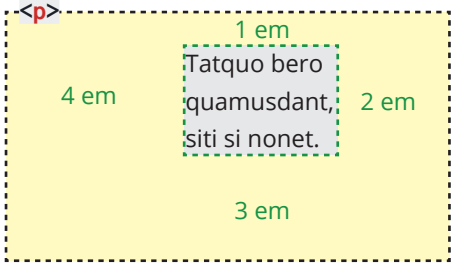

```
p {
  padding: 1em 2em;
}
```

Uvedu-li dvě hodnoty, je první hodnota pro horní a dolní okraj, druhá hodnota pro pravý a levý okraj.




```
p {
  padding: 1em 2em 3em 4em;
}
```

Čtyři hodnoty = samostatná hodnota pro každou stranu, začínáme od horního okraje po směru hodinových ručiček.



- Chceme-li nastavit pouze určité hodnoty a ostatní nechat nedotčené, můžeme použít konkrétní vlastnosti `padding-left`, `padding-right`, `padding-top`, `padding-bottom`.
- Podobně i pro margin, border.

Blokové a řádkové elementy

- **Blokové elementy** zabírají celou dostupnou šířku rodičovského prvku (nebo dokumentu). Typickým představitelem je například odstavec `<p>` nebo neutrální `<div>`.

  Ve výchozím stavu jsou tyto prvky přes celou šířku stránky, i když je obsah v nich kratší. Každý blokový element začíná na novém řádku (opět platí, pokud není výslovně nastaveno jiné chování).

- **Řádkové elementy** existují v rámci řádku a zabírají vždy jen tolik místa, kolik potřebují. Do řádkových elementů nelze vnořovat blokové elementy.

  Typickým představitelem jsou například elementy `<strong>`, `<a>` nebo neutrální `<span>`.

- **Řádkově blokové** elementy jsou hybridní - mají některé vlastnosti blokových a některé vlastnosti řádkových elementů. Řadí se za sebou v rámci "řádku", ale uvnitř se chovají jako blok. Text se zalamuje uvnitř elementu a lze jim např. nastavit šířku a výšku.

`<p>`

Blokový prvek zabírá celou dostupnou šířku.

`<div>`

Další blokový prvek začíná na novém řádku.

Uvnitř textu mohou být `<strong>` **řádkové elementy** zabírající pouze `<span>` nezbytně nutné místo. Všimni si, jak se řádkové elementy zalamují na konci řádku.

`<div>`

Řádkově blokové prvky jsou bloky, které se řadí za sebe v rámci řádku.

`<div>`

Řádky textu se zalamují uvnitř bloku.

Display

HTML elementy mají každý svoje výchozí chování - některé jsou blokové, některé řádkové, některé jiného typu. Způsob zobrazení prvku můžeme kdykoliv změnit pomocí CSS vlastnosti `display`.

Hodnoty vlastnosti display

`block`	blokový prvkek
`inline`	řádkový prvek
`inline-block`	řádkově-blokový prvek
`none`	skrytý prvek

Pro vlastnost display existuje celá řada dalších hodnot (table, table-call, list-item, apod.). Poněkud speciální je pak hodnota `display: none`, která prvek skryje, takže se v prohlížeči vůbec nezobrazí.

```
span {
  display: block;  /* původně řádkový element je nyní blokový */
}

div {
  display: inline;        /* řádkový prvek */
}

div {
  display: inline-block;  /* řádkově blokový prvek */
}

div {
  display: none;          /* skrytý prvek */
}
```


Obtékání obsahu: float

Obtékání obsahu typicky požadujeme např. u obrázků, kdy chceme text byl okolo obrázku. Obtékání zařídíme pomocí CSS vlastnosti `float`. Tuto vlastnost dáváme prvku, který má být obtékán (tj. obrázek).

Hodnoty vlastnosti float

`left`	prvek "plave" vlevo, text ho obtéká zprava
`right`	prvek "plave" vpravo, text ho obtéká zleva

Ita con eatem aut eosam, untistecest, volupta voluptature ea sus que conseque cum ad ma volum rendaer iaecusam veni quia doloritatus.

```
<p>
  <img src="dog.jpg">
  Ita con eatem ...
</p>
```

Bez obtékání.

Ita con eatem aut eosam, untiste-cest, volupta voluptature ea sus que conseque cum ad ma volum rendaer iaecusam veni quia doloritatus esto quid quos inis volupta tiant, ut quaspienes perrum sundios re cum solor sit as ipsam.

```
img {
  float: left;
}
```

Obtékání zprava.

Ita con eatem aut eosam, untiste-cest, volupta voluptature ea sus que conseque cum ad ma volum rendaer iaecusam veni quia doloritatus esto quid quos inis volupta tiant, ut quaspienes perrum sundios re cum solor sit as ipsam.

```
img {
  float: right;
}
```

Obtékání zleva.

Float pro vytváření layoutu

Prvky s nastavenou vlastností `float` jsou obtékány nejen textem, ale i dalšími prvky na stránce. Toho můžeme využít a vytvořit komplexnější rozložení stránky, tzv. layout.

Např. nastavíme-li dvěma prvkům šířku na 50% a necháme je plavat vedle sebe, získáme tzv. dvousloupcový layout.

První sloupec

Druhý sloupec

```
<div>
   První sloupec
</div>

<div>
   Druhý sloupec
</div>
```

```
div {
  width: 50%;
  float: left;
}
```

U blokových prvků s nastavenou vlastností `float` je nutné vždy nastavit šířku, jinak budou automaticky zabírat 100 % dostupného prostoru a bude to vypadat, jako že nejsou obtékány.

Obdobným způsobem můžeme jednoduše vytvořit i rozložení s jiným počtem sloupců:

Menu

`float: left;`
`width: 30%;`

Obsah

`width: 50%;`
`margin-left: 30%;`
`margin-right: 20%;`

Reklama

`float: right;`
`width: 20%;`

Všimněte si mírně odlišné techniky - menu plave vlevo, reklama plave vpravo, sloupec uprostřed neplave vůbec, ale má nastavené okraje, aby se mezi boční dva sloupce vešel.

Ukončení obtékání: clear

Aby mohly být plovoucí prvky (s nastaveným `float`) obtékány, jsou vyjmuty ze standardního toku dokumentu. Vypadá to, jako kdyby přestaly být součástí rodičovského elementu. Rodičovský element se tak ukončí dřív, než by se možná zdálo logické.

`float: left;`

Entibus doluptas dignatis siti accatia ad explis voluptatur aut officat emporum ex eum abor senem eosto magnis nonse niet ulparum quisqua turitat iistiae.

Obrázek trčí mimo svého rodiče. Někdy to nevadí a je to žádoucí chování. Chceme-li ale mít kolem textu a obrázku rámeček nebo barevné pozadí, máme problém.

CSS vlastnost `clear` slouží k zastavení obtékání. Přidává se k prvku, u kterého už nechceme, aby obtékal element s nastaveným float.

Hodnoty vlastnosti clear

`left`	zastaví obtékání prvku plovoucího vlevo
`right`	zastaví obtékání prvku plovoucího vpravo
`both`	zastaví obtékání zleva i zprava

`float: left;`

Bez clear

`float: left;`

S clear začně prvek až pod obrázkem

`clear: left;`

Pozicování prvků: position

CSS vlastnost `position` ovlivňuje, jakým způsobem se nastavuje pozice prvku na stránce.

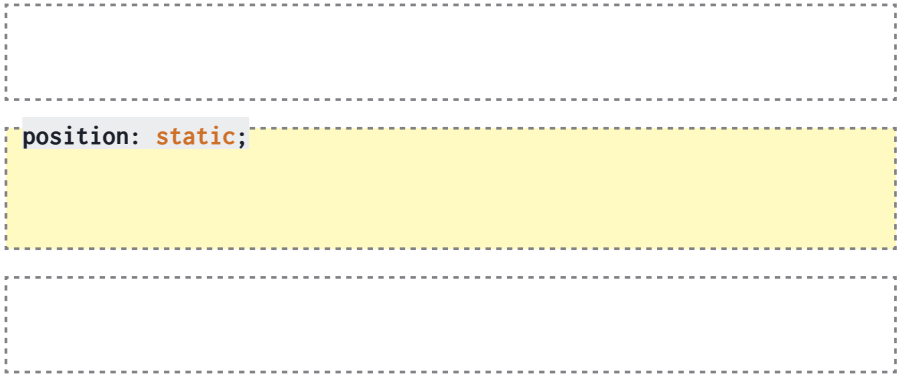
Hodnoty vlastnosti position

`static`	bez pozicování, zobrazí se normálně v toku dokumentu
`relative`	relativní pozicování k místu, kde by byl prvek umístěn v běžném toku dokumentu
`fixed`	fixní pozicování k oknu prohlížeče
`absolute`	absolutní pozicování vzhledem k rodičovskému prvku

position: static

Není potřeba uvádět, jde o výchozí hodnotu pro všechny elementy.

`position: static` říká, že element není pozicovaný - tj. vykreslí se v toku dokumentu v místě, kde při procházení HTML kódu přišel na řadu.

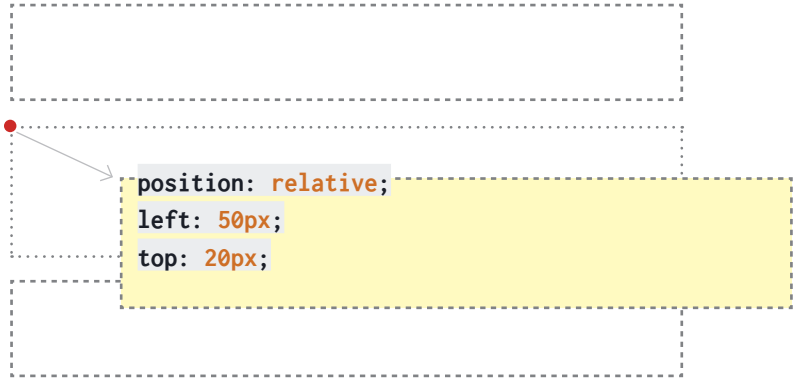


Prvky s jakoukoliv jinou hodnotou než `static` jsou pozicované.

position: relative

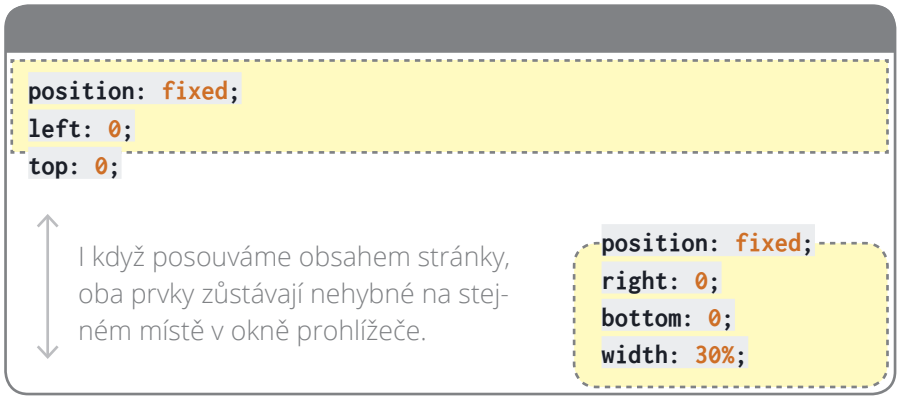
Prvek je pozicovaný relativně k místu, kde by byl normálně umístěný v toku dokumentu. `position: relative` bez dalších nastavených vlastností se chová stejně jako `position: static`.

Přidáme-li prvku souřadnice, můžeme ho vysunout mimo jeho běžné umístění.



position: fixed

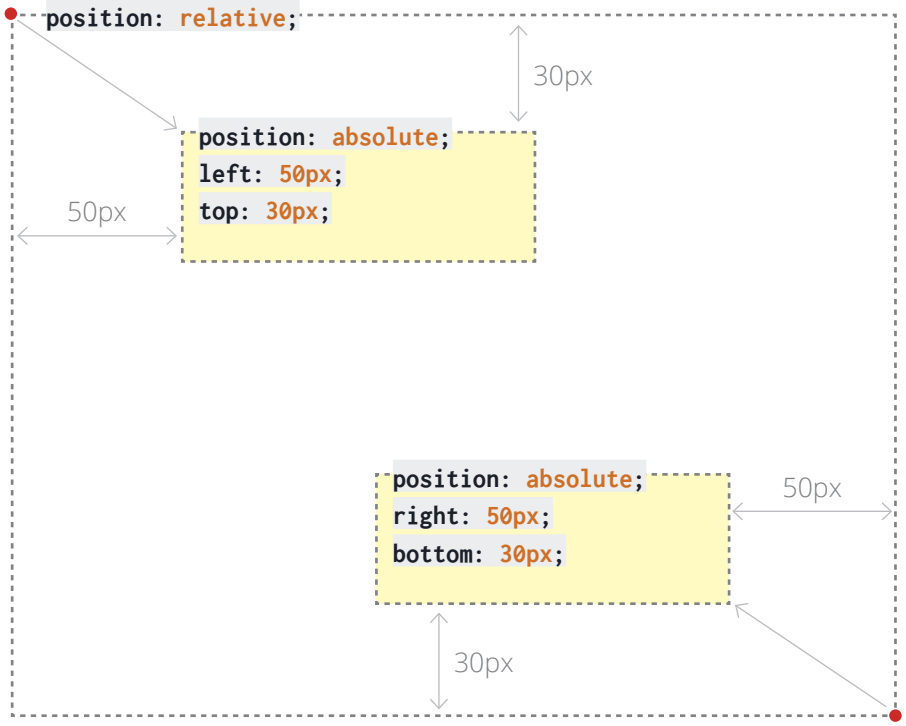
Prvek s `position: fixed` je pozicovaný relativně k oknu prohlížeče a zůstává nehybný na stejném místě, i když posouváme obsahem stránky. Používá se například na hlavičky nebo patičky webu, které mají být stále zobrazeny u horního nebo dolního okraje, a další podobné případy.



position: absolute

Prvek s `position: absolute` je pozicovaný relativně ke svému nejbližšímu pozicovanému rodičovskému elementu (tj. k prvku, který má `position` nastaveno na cokoliv, mimo hodnotu `static`). Pokud takový rodičovský prvek neexistuje, pozicuje se náš element vzhledem k hornímu levému rohu těla dokumentu.

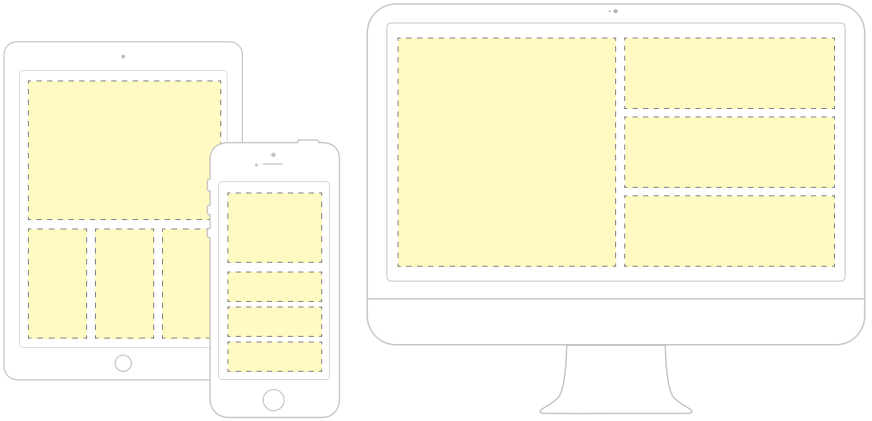
`position: absolute` se obvykle používá pro přesné rozmisťování prvků na stránce nebo uvnitř prvků uživatelského rozhraní.



Pozicovat lze nejen levý a horní okraj (vlastnosti `left` a `top`) elementu, ale i jeho pravý a dolní okraj (vlastnosti `right` a `bottom`). V tom případě se souřadnice počítají od pravého a/nebo dolního okraji rodičovského prvku.

Responzivní webdesign

- Termínem responsive webdesign (RWD) se označují techniky, pomocí kterých se obsah a vzhled stránky přizpůsobuje kontextu.
- Nejsnáze pochopitelný a nejčastěji používaný kontext je velikost zařízení, na kterém je web prohlížen - např. mobil, tablet, PC.
- Jeden zdrojový kód (HTML a CSS) se může zobrazovat na každém zařízení jinak.



Stejné HTML zobrazené na každém zařízení jinak.

3 základní techniky responzivního webdesignu:

- Fluidní layout - velikost boxů na stránce určujeme v procentech, jejich velikost se pak automaticky přizpůsobuje velikosti okna prohlížeče/zařízení.

- Flexibilní obrázky (a jiná média, např. videa) - nemají pevnou velikost, ale přizpůsobují se velikostí rodičovskému kontejneru.

- Media Queries - CSS deklarace uzavřené uvnitř media query se aplikují, pouze je-li splněna podmínka (např. minimální šířka okna prohlížeče/zařízení). Tímto způsobem můžeme mít v rámci jednoho CSS styly pro malá, střední, velká zařízení.

Media Queries

Media queries nám umožňují definovat podmínky, při jejichž splnění se použijí CSS deklarace uvnitř media query. Podmínky můžeme vytvářet pro různé vlastnosti daného média - např. velikost dostupné zobrazovací plochy (tj. velikost okna prohlížeče), orientace zařízení, rozlišení displeje, apod.

```
@media typ-média and (podmínka) {
  /* css deklarace při splnění podmínky */
}
```

Nejpoužívanější vlastnosti pro podmínky v media queries

`min-width`	podmínka je splněna, je-li dostupná plocha (okno prohlížeče, velikost mobilního zařízení) stejně široká nebo širší než nastavená hodnota, např. `min-width: 480px;`
`max-width`	podmínka je splněna, je-li dostupná plocha (okno prohlížeče, velikost mobilního zařízení) stejně široká nebo užší než nastavená hodnota, např. `max-width: 960px;`
`min-height` `max-height`	stejné jako `min-width` a `max-width`, ale pro výšku
`orientation`	testuje orientaci zařízení, zda ho uživatel drží na výšku (`portrait`) nebo na šířku (`landscape`) např. `orientation: landscape;`
`min-resolution` `max-resolution`	testuje rozlišení zařízení např. `min-resolution: 160dpi;`
`aspect-ratio`	jaký je poměr stran obrazovky např. `aspect-ratio: 16/9;`

Typické CSS s media query vypadá např.takto:

```
@media screen and (min-width: 480px) {
  .box { width: 50%; }    /* na displejích širších než 480px
}                            má box šířku 50% */
```


Breakpoint

V naprosté většině případů budeme používat media queries pro změnu layoutu stránky při změně šířky zobrazovací plochy (okna prohlížeče). Tj. zatím snadno vystačíme pouze s vlastnostmi `min-width` a `max-width`. Šířka okna, kde se mění layout stránky, označujeme jako breakpoint.

Mobile first přístup k návrhu webu

Tzv. mobile first (někdy také mobile up) přístup znamená, že při návrhu webu budujeme CSS od nejmenších mobilních zařízení směrem nahoru. Tj. za výchozí rozložení našeho webu považujeme layout pro mobilní telefony a vzhled postupně vylepšujeme a obohacujeme pro větší displeje.

Např. máme článek, který obsahuje nadpis, obrázek a text. Na malé obrazovce mobilního telefonu chceme mít vše pod sebou. Na větších zařízeních můžeme mít obrázek vedle textu.

```
<h1>Nadpis článku</div>
<img class="foto" src="foto.jpg" alt="Obrázek">
<p>Text článku...</p>
```

```
.foto {
  /* ve výchozím stavu je obrázek sám na řádku a zabírá celou šířku */
  display: block;
  width: 100%;
  height: auto;
}

@media screen and (min-width: 480px) {
  /* je-li displej širší než 480px, aplikuj následující styl:
  obrázek plave vpravo, je široký 50% stránky a má kolem sebe okraj */
  .foto {
    float: right;
    width: 50%;
    margin: 0 0 1em 1em;
  }
}
```

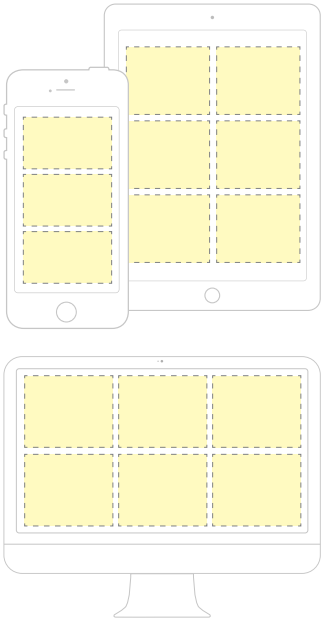
Stejným způsobem můžeme měnit například šířku sloupců ve vícesloupcovém layoutu:

```
<div class="box"></div>
<div class="box"></div>
...
```

```
/* na mobilu plná šířka a pod sebou */
.box { width: 100%; }

/* na tabletu 2 vedle sebe */
@media screen and (min-width: 480px) {
  .box {
    float: left;
    width: 50%;
  }
}

/* na PC 3 vedle sebe */
@media screen and (min-width: 768px) {
  .box {
    width: 33.3333%;
  }
}
```



Jaké jsou hodnoty breakpointů pro mobil, tablet a PC?

Žádná standardní hodnota breakpointu pro mobil, tablet nebo PC neexistuje. Na trhu jsou stovky až tisíce různých mobilních zařízení, každé s jiným rozlišením displeje. Není možné vzít jednu hodnotu a říci, že to je šířka displeje mobilního telefonu nebo tabletu. Je zbytečné se o to snažit.

Breakpointy nenastavujte podle typu zařízení, ale vždy podle obsahu. Rozložení stránky měňte ve chvíli, kdy to pro obsah dává smysl. Např. při které šířce už není praktické mít 2 sloupce vedle sebe, protože jsou moc úzké, ale je lepší přepnout na jednosloupcový layout, kde jsou informace z obou sloupců pod sebou? To místo je ideální breakpoint.

Veškerý obsah na webu se také nemusí lámat ve stejných místech. Je možné (pravděpodobné!), že například menu je vhodné přepnout na jiný typ zobrazení v odlišném místě, než seznam zboží pod ním.

Z praktických důvodů se obvykle používají 2–3 stejné breakpointy v rámci celého webu. V místech, kde se obsah láme nevhodně a potřebujeme breakpoint jinde, se ale nebojte vytvořit nový breakpoint specifický pro tento kousek obsahu.

Fluidní layout

Rozměry boxů na stránce nenastavujeme v pixelech ani jiných absolutních jednotkách, ale v procentech. Velikost se pak automaticky přizpůsobuje šířce okna prohlížeče a zůstává poměrově stejná na různých zařízeních.

Potřebujeme-li na procenta převést konkrétní šířku v pixelech, použijeme vzorec:

```
šířka v % = (požadovaná šířka v px / kontext v px) * 100
```

Kontext je šířka obalujícího boxu (nebo celé stránky / okna prhlížeče).

Příklad:
Máme "tablet" se rozlišením 768px na šířku. To je náš kontext. Chceme, aby při zobrazení na tomto zařízení, měl levý sloupec šířku 250px.
`Požadovaná šířka v % = (250 / 768) * 100 = 32.55208333%`
Procenta nezaokrouhlujeme, aby byl výpočet co nejpřesnější.

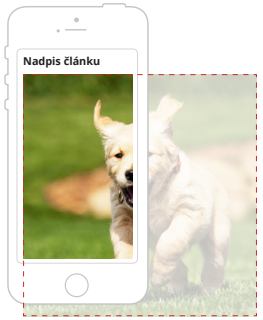
Flexibilní obrázky

Při návrhu responzivního webu chceme, aby se obrázky pružně přizpůsobovaly velikosti okna prohlížeče stejně jako náš layout. Použijeme CSS vlastnost `max-width`, která zajistí, že obrázek nikdy nepřesáhne nastavenou velikost. Velký obrázek pak např. na mobilním telefonu nevyčuhuje z obrazovky ven, ale přizpůsobí se velikosti rodičovského boxu.

```
img {
  max-width: 100%;
  height: auto;
}
```

`Height: auto` zaručuje stejný poměr stran obrázku, jako má originál. Obrázek se nedeformuje.

max-width:100%



Bez max-width

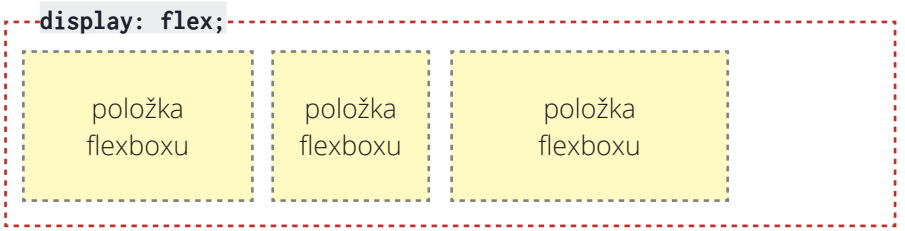
Nastavení viewportu

Pro správné zobrazení responzivních stránek v mobilních prohlížečích přidáme do hlavičky stránky následující řádek:

```
<head>
  …
  <meta name="viewport" content="width=device-width, initial-scale=1">
  …
</head>
```


Flexbox

- Flexbox (flexible box) poskytuje efektivní způsob, jak rozmístit a zarovnat prvky uvnitř kontejneru a rozdělit mezi ně zbývající volné místo.
- Flexbox funguje, i když je velikost prvků uvnitř dopředu neznámá nebo se dynamicky mění.
- Standardně flexbox rozmisťuje prvky ve vodorovné ose, ale lze ho přepnout i na osu svislou.
- Položkou flexboxu (flex item) se stává každý přímý potomek kontejneru, který má na sobě nastaveno `display: flex;`

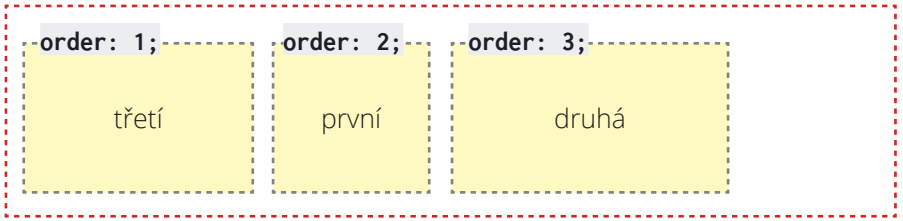



```
<div class="kontejner">
  <div>položka flexboxu</div>
  <div>položka flexboxu</div>
  <div>položka flexboxu</div>
</div>
```

```
.kontejner {
  display: flex;
}
```

order: změna pořadí položek flexboxu

- Položky uvnitř flexboxu můžeme seřadit za sebe v libovolném pořadí pomocí vlastnosti `order`.
- Pozor: Výchozí hodnota `order` je `0`. Nastavíme-li tedy nějaké položce `order: 1` v očekávání, že bude první v řadě, budeme zklamaní, protože bude až za všemi položkami, které nemají `order` nastavený vůbec (tj. mají ho `0`).
- Pro přehlednost a zaručeně správný výsledek je tedy nejlepší nastavit `order` explicitně všem položkám uvnitř flexboxu.
- Pořadí lze nastavit i na zápornou hodnotu (ty jsou pak řazeny před položkami s nulou).

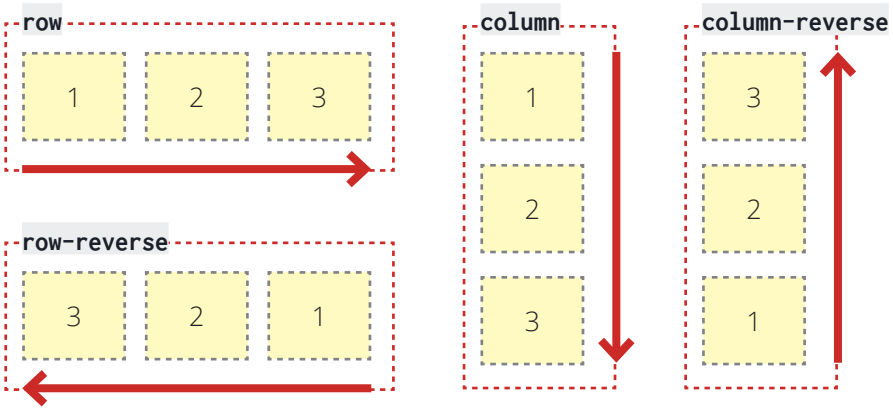



```html
<div class="kontejner">
  <div class="prvni">první</div>
  <div class="druha">druhá</div>
  <div class="treti">třetí</div>
</div>
```

```css
.kontejner {
  display: flex;
}
.prvni { order: 2; }
.druha { order: 3; }
.treti { order: 1; }
```

flex-direction: směr flexboxu

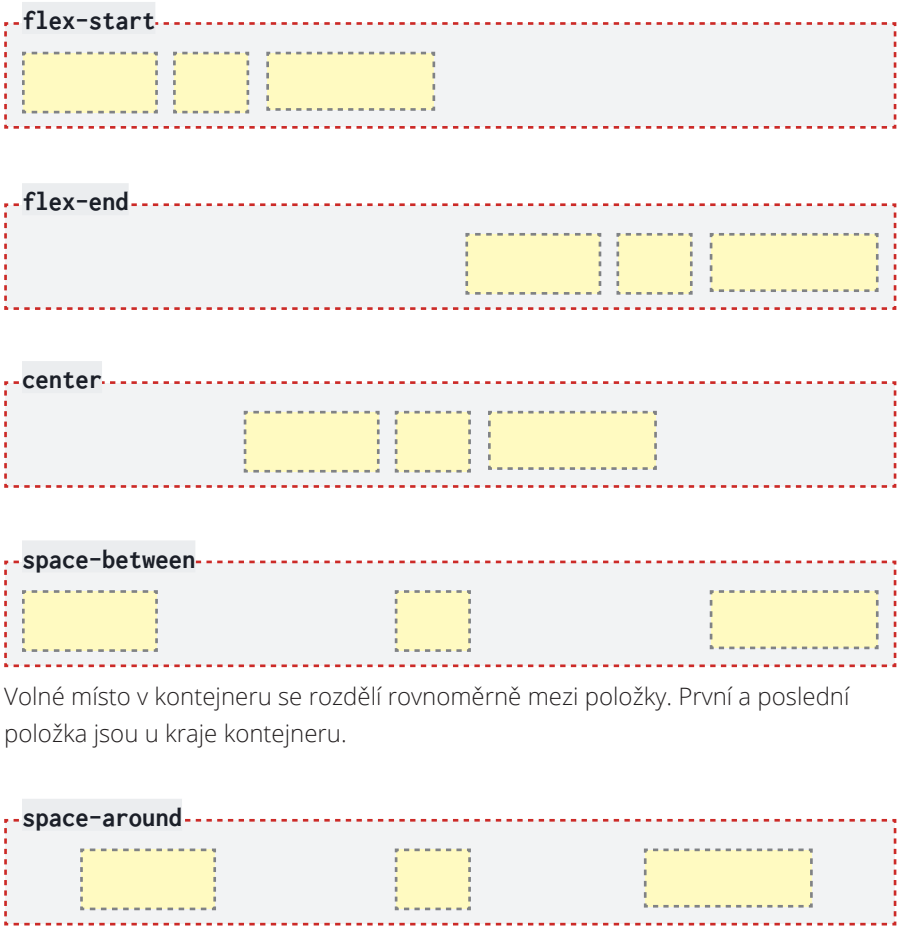
- Položky flexboxu se ve výchozím stavu řadí za sebe ve vodorovném směru. Tomuto směru se říká hlavní osa.
- Směr lze změnit pomocí vlastnosti `flex-direction`.
- Možné hodnoty: `row`, `row-reverse`, `column`, `column-reverse`
- Nastavuje se na rodičovském kontejneru.



Při nastavení `flex-direction: column;` se prvky neřadí vedle sebe, ale pod sebe. Hlavní osa se změní z vodorovné na svislou.

justify-content: zarovnání položek ve směru hlavní osy

- Pomocí vlastnosti `justify-content` můžeme položky flexboxu zarovnat ve směru hlavní osy.
- Možné hodnoty:
  `flex-start`, `flex-end`, `center`, `space-between`, `space-around`
- Nastavuje se na rodičovském kontejneru.



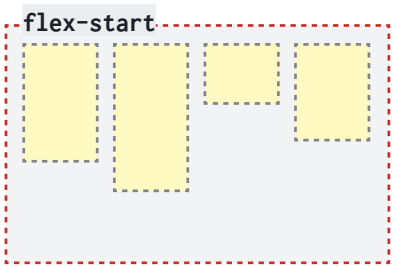
Volné místo v kontejneru se rozdělí rovnoměrně mezi položky. První a poslední položka jsou u kraje kontejneru.

Volné místo v kontejneru se rozdělí rovnoměrně vpravo i vlevo každé položky.

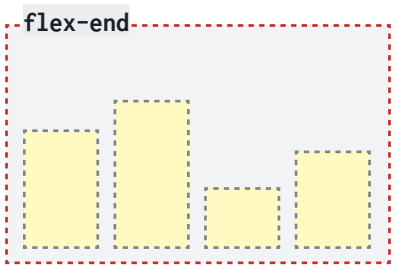
Je-li `flex-direction: column`, zarovnávají se prvky stejným způsobem, ale ve vertikální ose.

align-items: zarovnání položek ve směru vedlejší osy

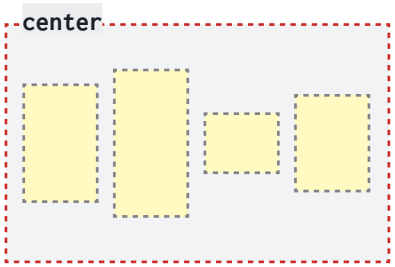
- Pomocí vlastnosti `align-items` můžeme položky flexboxu zarovnat ve směru vedlejší osy.
- Možné hodnoty: `flex-start`, `flex-end`, `center`, `stretch`, `baseline`.
- Výchozí hodnota je `stretch` - všechny položky v řadě jsou stejně vysoké.
- Nastavuje se na rodičovském kontejneru.



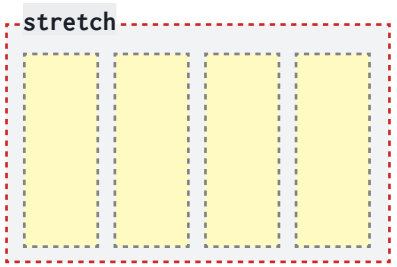
Na začátek (horní hranu) kontejneru



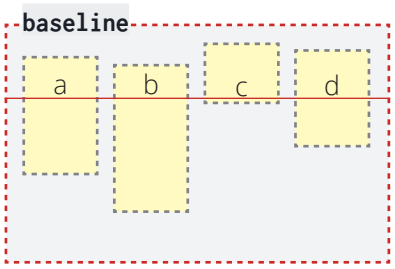
Na konec (dolní hranu) kontejneru



Na střed kontejneru



Všechny položky se roztáhnou na výšku kontejneru. Nemá-li kontejner nastavenou výšku, roztáhnou se všechny položky podle nejvyšší z nich.

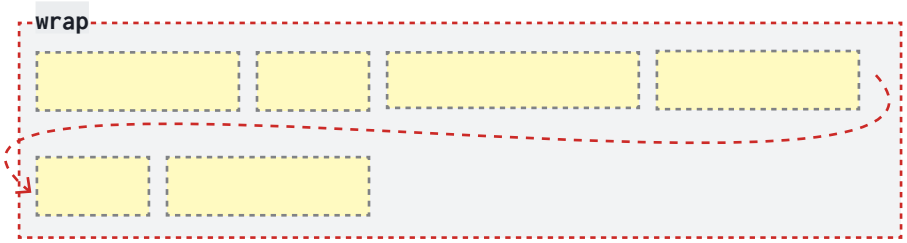


Zarovnání na účaří textu uvnitř položek.

Je-li `flex-direction: column`, natahují a zarovnávají se položky stejným způsobem, ale ve vodorovné ose.

flex-wrap: zalomení řady položek flexboxu

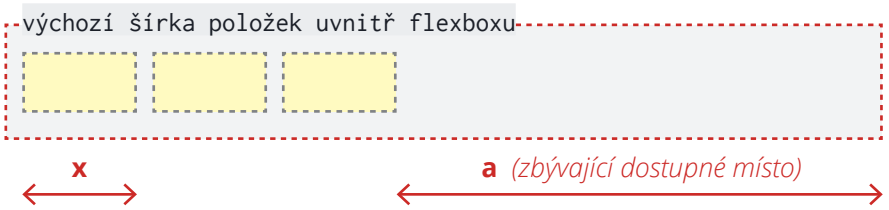
- Ve výchozím stavu se flexbox snaží dostat všechny položky vedle sebe do jednoho řádku.
- Pomocí vlastnosti `flex-wrap` můžeme flexboxu říci, že obsah, který se do řádku nevejde, se má na konci řádku zalomit a pokračovat na dalším řádku.
- Možné hodnoty: `nowrap`, `wrap`, `wrap-reverse`
- Výchozí hodnota `nowrap` - nezalamuje se.
- Nastavuje se na rodičovském kontejneru.

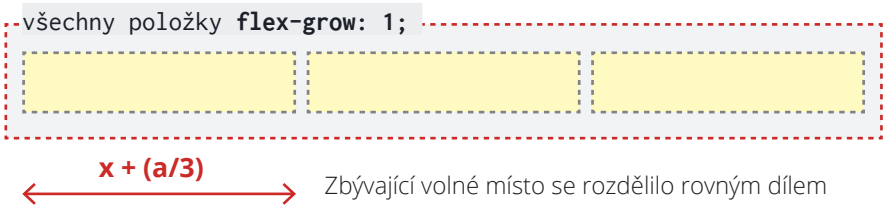


Hodnota `wrap-reverse` funguje obdobně, ale řadí položky zprava doleva.

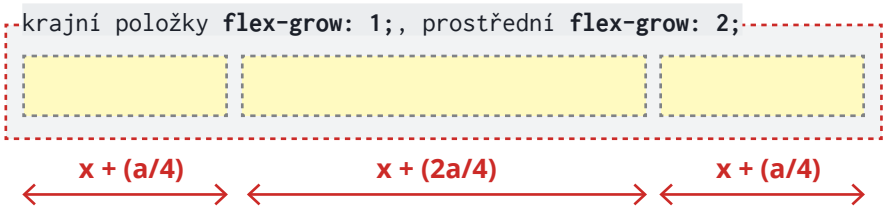
flex-grow: zvětšování položek flexboxu

- Nastavuje položce flexboxu schopnost .zvětšovat se v případě potřeby.
- Jako hodnota se uvádí kladné číslo bez jednotky, které slouží jako poměr, v jakém se má položka zvětšovat vzhledem k ostatním položkám flexboxu.
- V tomto poměru se mezi položky flexboxu rozděluje zbývající volné místo.
- Nastavuje se na položce flexboxu (potomek rodičovského kontejneru).
- Mají-li všechny položky flexboxu nastaveno `flex-grow: 1;`, rozděluje se zbývající volné místo rovnoměrně mezi ně.
- Má-li jedna z položek f`lex-grow: 2;`, bude se k ní dostupné volné místo přidávat dvakrát rychleji než k ostatním položkám s hodnotou `1`.
- Výchozí hodnota je `0` (položka se nezvětšuje).





Zbývající volné místo se rozdělilo rovným dílem mezi všechny položky flexboxu.



Zbývající volné místo se rozdělilo na 4 díly (součet všech flex-grow = 1+2+1 = 4) a jeden díl se přidal krajním položkám, dva díly prostřední položce (natahuje se v poměru 2:1 k ostatním díky flex-grow: 2).

flex-shrink: zmenšování položek flexboxu

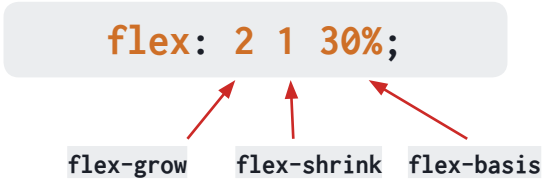
- Funguje obdobně jako `flex-grow`, ale pro zmenšování položek.
- Je-li ve flexboxu málo místa na to, aby se položky vešly vedle sebe, flexbox začne položky smršťovat. Flex-shrink určuje, v jakém poměru k ostatním položkám se bude položka zmenšovat.
- Výchozí hodnota je `1` (položky se zmenšují ve stejném poměru).
- Nastavuje se na položce flexboxu (potomek rodičovského kontejneru).

flex-basis: výchozí velikost položky flexboxu

- Nastavuje výchozí velikost položky flexboxu předtím, než je mezi položky rozděleno zbývající volné místo.
- Jako hodnota se udává velikost ve standardních jednotkách (%, em, px) nebo `auto`. Hodnota `auto` znamená, že se velikost položky flexboxu nastaví podle její hodnoty `width` nebo `height`.
- Výchozí hodnota je auto.
- Nastavuje se na položce flexboxu (potomek rodičovského kontejneru).

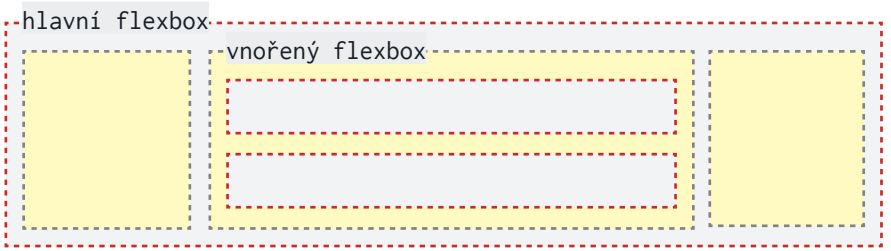
flex: zkrácený zápis

- Vlastnost `flex` se používá jako zkrácený zápis pro vlastnosti `flex-grow`, `flex-shrink` a `flex-basis`.



Vnoření více flexboxů do sebe

- Při tvorbě složitějších layoutů lze flexboxy vnořovat do sebe.
- Položka jednoho flexboxu může bý zároveň sama flexboxem s vlastními položkami.



CSS selektory II.

element1 > element2

Vybere element2, který je přímým potomkem elementu1.

```
<div>
  <p>První odstavec</p>
  <span>
    <p>Druhý odstavec</p>
  </span>
</div>
```

```
div > p {
  color: red;
}
```

První odstavec
Druhý odstavec

První odstavec bude červený, protože jeho přímý rodič je `div`. Druhý odstavec nebude červený. Je sice také vnořený uvnitř `div`, ale až ve druhé úrovni; jeho přímým rodičem je `span`..

element1 + element2

Vybere element2, který následuje hned po elementu1, kde oba jsou potomkem stejného rodiče.

```
<div>
  <img src="dog.jpg">
  <p>První odstavec</p>
  <p>Druhý odstavec</p>
</div>
```

První odstavec
Druhý odstavec

```
img + p {
  color: red;
}
```

První odstavec `p` bude červený, protože následuje ihned za obrázkem `img` a oba jsou sourozenci na stejné úrovni (jsou potomkem stejného rodiče `div`). Druhý odstavec není hned za obrázkem, zůstane černý.

Pseudo-třídy

- Selektor pseudo-třídy začíná vždy dvojtečkou.
- Pseudo-třídy umožňují stylovat stav určitého prvku na stránce (např. odkaz, nad kterým je kurzor myši, aktivní formulářový prvek, apod.).

Interaktivní pseudo-třídy

`a:link` `a:visited`	nenavštívený odkaz dříve navštívený odkaz
`:hover`	prvek, nad kterým je kurzor myši (vhodné pro odkazy, ale funguje na většinu prvků)
`:focus`	prvek, který má na stránce focus (buď jsme na něj klikli myší nebo jsme se na něj doťukali tabelátorem)
`:active`	prvek, který je momentálně aktivován uživatelem (např. odkaz nebo tlačítko, na které uživatel kliknul levým tlačítkem myši, ale ještě tlačítko nepustil)

:first-child :last-child :nth-child(a)

Vybere element, který je prvním/posledním/n-tým potomkem svého přímého rodiče.

```
<div>
  <p>První odstavec</p>
  <p>Druhý odstavec</p>
  <p>Třetí odstavec</p>
  <p>Čtvrtý odstavec</p>
  <p>Pátý odstavec</p>
</div>
```

```
p:first-child {
  color: red;
}

p:last-child {
  color: blue;
}

p:nth-child(3) {
  color: green;
}
```

První odstavec
Druhý odstavec
Třetí odstavec
Čtvrtý odstavec
Pátý odstavec

Pozor: častá chyba. Např. `p:first-child` nevybírá první odstavec. Vybírá prvek, který je odstavec a zároveň je prvním potomkem svého přímého rodiče.

```
<div>
  <img src="dog.jpg">
  <p>První odstavec</p>
  <p>Druhý odstavec</p>
</div>
```

První odstavec
Druhý odstavec

```
p:first-child {
  color: red;
}
```

První odstavec nebude červený, protože není prvním potomkem svého přímého rodiče (`div`). Prvním potomkem je v tomto příkladuobrázek `img`.

:nth-child(an) :nth-child(an+b)

Zatímco `:nth-child(3)` vybere třetího potomka, `:nth-child(3n)` vybere každého třetího (tj. 3, 6 ,9 , ...).
Nejpokročilejší verze je pak ve formátu `:nth-child(3n+2)` a vybere každého třetího potomka, počínaje druhým (tj. 2, 5, 8, ...).
Lze použít i `:nth-child(odd)` a `:nth-child(even)` pro výběr všech lichých nebo sudých potomků.

`:nth-child(3)`

`:nth-child(3n)`

`:nth-child(3n+2)`

Pseudo-elementy

- Selektor pseudo-elementu začíná vždy dvěma dvojtečkami. (Dříve se uváděla pouze jedna, nově se uvádějí dvě, aby se odlišily pseudo-elementy od pseudo-tříd).
- Pseudo-element je část elementu, která fakticky není v HTML nijak vyznačená, ale přesto ji můžeme stylovat.

::first-letter

První písmeno elementu. Lze použít např. pro nastylování prvního písmene v kapitole knihy (iniciála).

```
<p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean vitae faucibus.</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean vitae faucibus.

```
p::first-letter {
  display: inline-block;
  float: left;
  color: red;
  font-size: 3em;
}
```

::first-line

První řádek textu uvnitř elementu..

```
<p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Aenean vitae faucibus.</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean vitae faucibus.

```
p::first-line {
  color: red;
  font-size: 16px;
}
```

JavaScript

- Přidává do stránky chování, stránka může "něco dělat".
- Umožňuje stránce reagovat na vstup uživatele, ať už se jedná o stisk klávesy, pohyb nebo kliknutí myši, vyplnění formuláře, kliknutí na tlačítko, apod.
- JavaScript píšeme do externího souboru s koncovkou .js a ten pak vložíme do stránky pomocí značky `<script></script>` a to obvykle buď do hlavičky stránky nebo na její konec před značku `</body>`.
- Skript v hlavičce blokuje další načtení stránky, dokud se celý nestáhne, a nemá ještě přístup k obsahu stránky (ještě není načtená). Je-li to možné, doporučuje se dávat skript na konec dokumentu.

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <!-- hlavička stránky -->
  </head>
  <body>
    <!-- zde je obsah stránky -->

    <script src="skript.js" type="text/javascript"></script>
  </body>
</html>
```

Proměnné

- Proměnná je pojmenované místo v paměti, kam si můžeme uložit vlastní hodnotu, se kterou budeme v programu dále pracovat.
- Abychom mohli proměnnou použít, musíme ji na začátku programu (nebo funkce) nejprve nadefinovat (říct, počítači, že s takovou proměnnou budeme pracovat). To se dělá pomocí klíčového slova `var`, za kterým následuje jméno proměnné.


```
var jmeno;
jmeno = 'Jana';

var vek;
vek = 27;

// Jana má narozeniny
// přičteme k věku jedničku
vek = vek + 1;
```

Do proměnných můžeme vkládat text, čísla nebo další datové typy. Datový typ v JavaScriptu nemusíme dopředu určovat, nastaví se sám podle vkládané hodnoty.

Text (řetězec) se uzavírá do uvozovek nebo apostrofů.

Podmínka

- V programu se často potřebujeme rozhodovat a provádět různé příkazy podle toho, jaká je zrovna v proměnné hodnota. K tomu slouží podmínka `if`.
- Příkazy uvnitř větve if se vykonají, je-li podmínka splněna.
- Podmínka může mít i větev `else`, která se provede v případě, že podmínka splněna není.

```
if (pocet === 3) {
  // tyto příkazy se vykonají, je-li počet roven 3
} else {
  // je-li jiný počet, vykonají se tyto příkazy
}
```

Rovnost hodnot porovnávej pomocí trojitého rovnítka `===`. Použít můžeš samozřejmě i další srovnávací operátory `<`, `>`, `<=`, `>=` nebo `!==` (nerovná se).

Výrazy uvnitř podmínek lze skládat pomocí logických operátorů `||` (nebo), `&&` (a zároveň), `!` (ne, neplatí).

```
if (jmeno === 'Jana' && vek >= 18) {
  // zde kód pouze pro Jany, které jsou zároveň plnoleté
}
```

```
if (vek <= 3 || vek >= 65) {
  // pro mimina nebo důchodce
}
```

Cyklus

- Slouží pro opakované provedení sady příkazů.
- Cyklus `for` se obvykle používá když dopředu známe počet opakování. Např. chceme něco udělat 5x.
- Cyklus `while` se obvykle používá, když dopředu počet opakování neznáme a chceme příkazy provádět tak dlouho, dokud nedojdeme do nějakého stavu (dokud není splněna podmínka). Např. sčítej čísla, dokud není suma větší než 100.

```
for (var i = 0; i < 5; i++) {
  // příkazy uvnitř cyklu se provedou celkem 5x
  // i++ na konci každé smyčky cyklu zvětší i o 1
  // proměnná i postupně nabývá hodnot 0, 1, 2, 3 a 4
}
```

```
var soucet;
soucet = 0;
v// dokud platí podmínka, prováděj příkazy uvnitř cyklu
while (soucet < 100) {
  // budeme k součtu přičítat náhodné číslo
  // dokud nebude součet větší než 100
  soucet = soucet + nahodneCislo;
}
```

Funkce

- Skupina příkazů, které si pomocí klíčového slova `function` zabalíš do funkce. Funkce má jméno a tímto jménem ji pak můžeme volat - vykoná se sada příkazů uvnitř funkce.
- Do funkce můžeš (ale nemusíš) předat hodnoty, tzv. parametry funkce..

```
// nadefinujeme funkci
function nazevFunkce() {
  // příkazy uvnitř funkce
}

// funkci zavoláme jménem
nazevFunkce();
```

```
// funkce s parametry
function soucet(a, b) {
  return a + b;
}

// zavoláme funkci s parametry
x = soucet(37, 5);
// v x je hodnota 42
```


Jak JavaScriptem ovlivňovat prvky HTML stránky

- Pro práci s HTML prvkem na stránce musíš tento prvek nejprve najít a získat na něj odkaz, který si většinou uložíš do proměnné, aby byla další práce s ním snazší.

Vyhledání HTML prvku na stránce

Lze použít několik metod. Pokud má HTML prvek nastavený atribut **id**, je nejjednodušší a nejrychlejší použít metodu: `document.getElementById('id')`.

```
// na stránce vyhledáme prvek s id="nadpis"
// ten vypadá třeba takto <h1 id="nadpis">Toto je nadpis</h1>
// odkaz na nalezený prvek si uložíme do proměnné prvek
var prvek;
prvek = document.getElementById('nadpis');
```

Potřebuješ-li hledat prvek podle jiných kriterií než je id, použij metodu `document.querySelector` (najde první prvek na stránce, který odpovídá daným kritériím) nebo `document.querySelectorAll` (nalezne všechny prvky na stránce, které odpovídají daným kritériím, a vrátí jejich seznam v poli).

Jako vyhledávací kritéria lze v tomto případě použít zápis shodný s tím, jak se zapisují selektory v CSS. Lze tedy hledat podle id, názvu třídy, typu elementu, včetně kombinovaných selektorů i pseudoselektorů.

```
// najdeme div s přiřazenou css třídou "menu"
var prvek = document.querySelector('div.menu');

// najdeme div s id="nadpis"
var prvek = document.querySelector('#nadpis');

// najdeme první odstavec v patičce stránky
var prvek = document.querySelector('footer p:first-child');
```

Přímé nastavení CSS vlastností

Nejčastějším úkonem prováděným nad HTML prvky je změna jejich CSS vlastností (zobrazení/skrytí, barva, poloha, ...). K CSS vlastnostem přistupuj pomocí konstrukce ve tvaru `prvek.style.cssVlastnost`.

Všechny víceslovné názvy CSS vlastností (mají v názvu pomlčku) v JavaScriptu zapisujeme pomocí tzv. camel case - pomlčky se odstraní, první písmeno je malé, každé další slovo začíná velkým písmenem.

v CSS	v JavaScriptu	
font-size	fontSize	velikost písma
background-color	backgroundColor	barva pozadí

```
// najdeme na stránce prvek s id="nadpis"
// a nastavíme jeho barvu na červenou a zvětšíme mu písmo
var nadpis;
nadpis = document.getElementById('nadpis');
nadpis.style.color = 'red';
nadpis.style.fontSize = '80px';
```

Práce s CSS třídami

Obvykle není vhodné nastavovat CSS vlastnosti prvku přímo, ale raději bychom k prvku přidali nebo z něj odebrali CSS třídu (máme ji nastavenou v CSS souboru, přidáním třídy můžeme nastavit spoustu vlastností najednou).

S třídami pracujeme pomocí konstrukce `prvek.classList` a používáme metody **`add`**, **`remove`** nebo **`toggle`** (toggle přidá třídu, pokud na prvku ještě není; odebere třídu, pokud už na prvku je).

```
var nadpis = document.getElementById('nadpis');
nadpis.classList.add('velkycerveny');    // přidá třídu
nadpis.classList.remove('velkycerveny'); // odebere třídu
nadpis.classList.toggle('velkycerveny'); // přepne třídu
```


Události

- V HTML stránce dochází k různým událostem - stisk klávesy, pohyb nebo kliknutí myši, dokončení načtení stránky, apod. V JavaScriptu můžeme na tyto události reagovat. Kompletní seznam událostí je mnohem delší, toto jsou ty nejpoužívanější, se kterými si zatím vystačíme.

Událost	Popis
`onLoad`	k události dojde, když se do prohlížeč dokončil načítání celá stránky; hodí se, potřebujeme-li provést něco hned při "startu" stránky
`onClick`	při kliknutí na prvek myší
`onMouseOver`	při najeti kurzorem myši nad prvek
`onMouseOut`	při odjetí kurzoru myši pryč z prvku
`onKeyDown`	při stisknutí klávesy
`onKeyUp`	při uvolnění klávesy

Událost, na kterou chceme reagovat, přidáme jako atribut k HTML prvku a jako hodnotu atributu uvedeme název funkce, která se má provést, když k události dojde.

```
<h1 onClick="obarvit();">Kliknutím na nadpis ho obarvi</h1>
```

```
function obarvit() {
  this.style.color = 'red';
}
```

Uvnitř funkcí reagujících na události můžeme použít klíčové slovo **`this`**, které odkazuje na prvek, který událost vyvolal.

Už umíš dělat weby!

Juchuchů :)