Webové fonty

Pokud chceme aktuálně používat nějaký font na webové stránce, musí být nainstalován na uživatelově zařízení a to ještě pod nějakým generalizovaným jménem. To je pro většinu stránek šílený problém a proto si fonty můžeme na webové stránce načítat z jiných zdrojů.

Fonty načítáme uvnitř CSS a v prohlížeči se na daný font načte a uloží do cache. V cache je po nějakou dobu a nemusí se po danou dobu stahovat.

Fonty jsou relativně velké soubory, musí uchovávat spoustu informací a proto bych jich na webových stránkách nemělo být spousta. Obecně limit samozřejmě daný není, ale doporučuje se mít max. 3 různá písma s různými variantami. Mezi varianty se počítá tloušťka písma a například i styl kurzívy.

My se nebudeme zaobírat tím, jak se přímo fonty registrují v CSS, ale ukážeme si službu Google Fonts, která nám velmi jednoduše přidá jakýkoliv font z jejich velké kolekce, kde jsou všechny k dispozici zdarma. Na stránkách Google Fonts naleznete jazyky označené Latin Extended, které mohou obsahovat znaky, které Český jazyk používá.

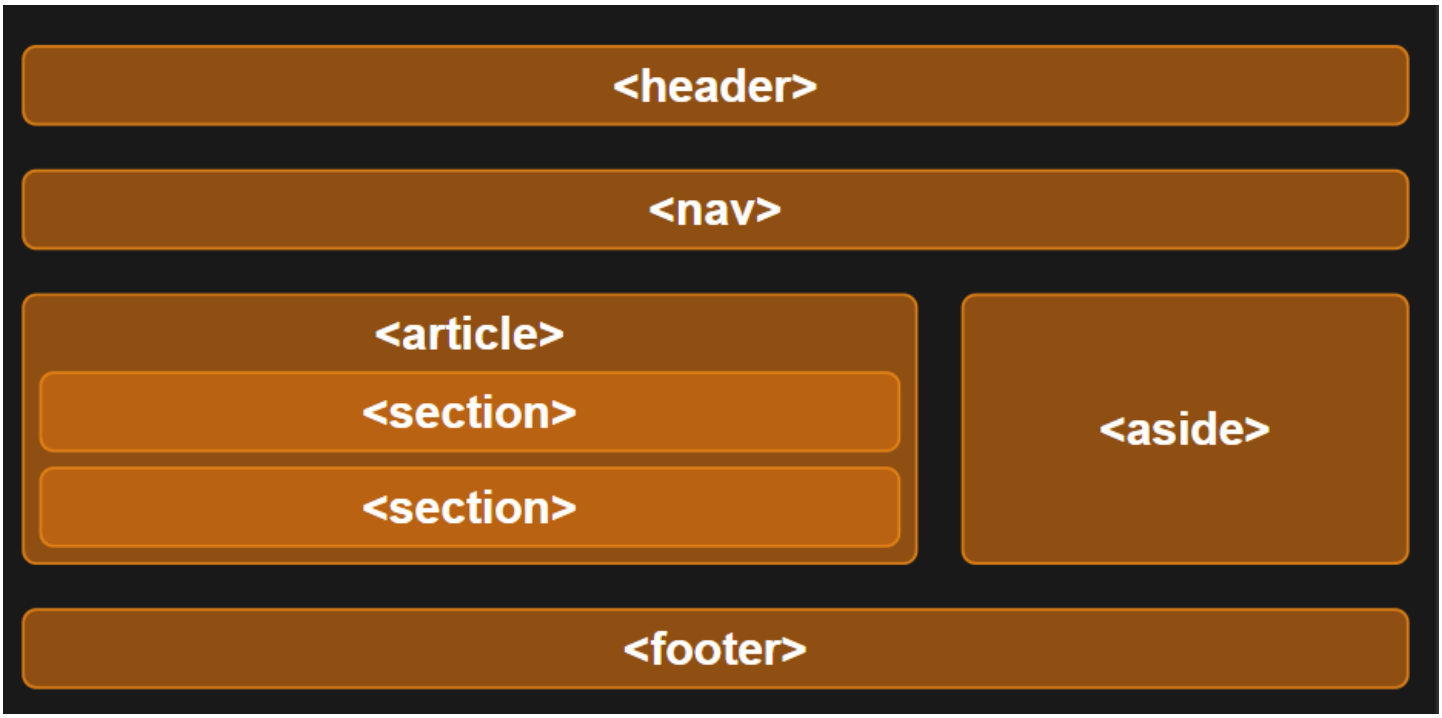
Tagy HTML rozložení

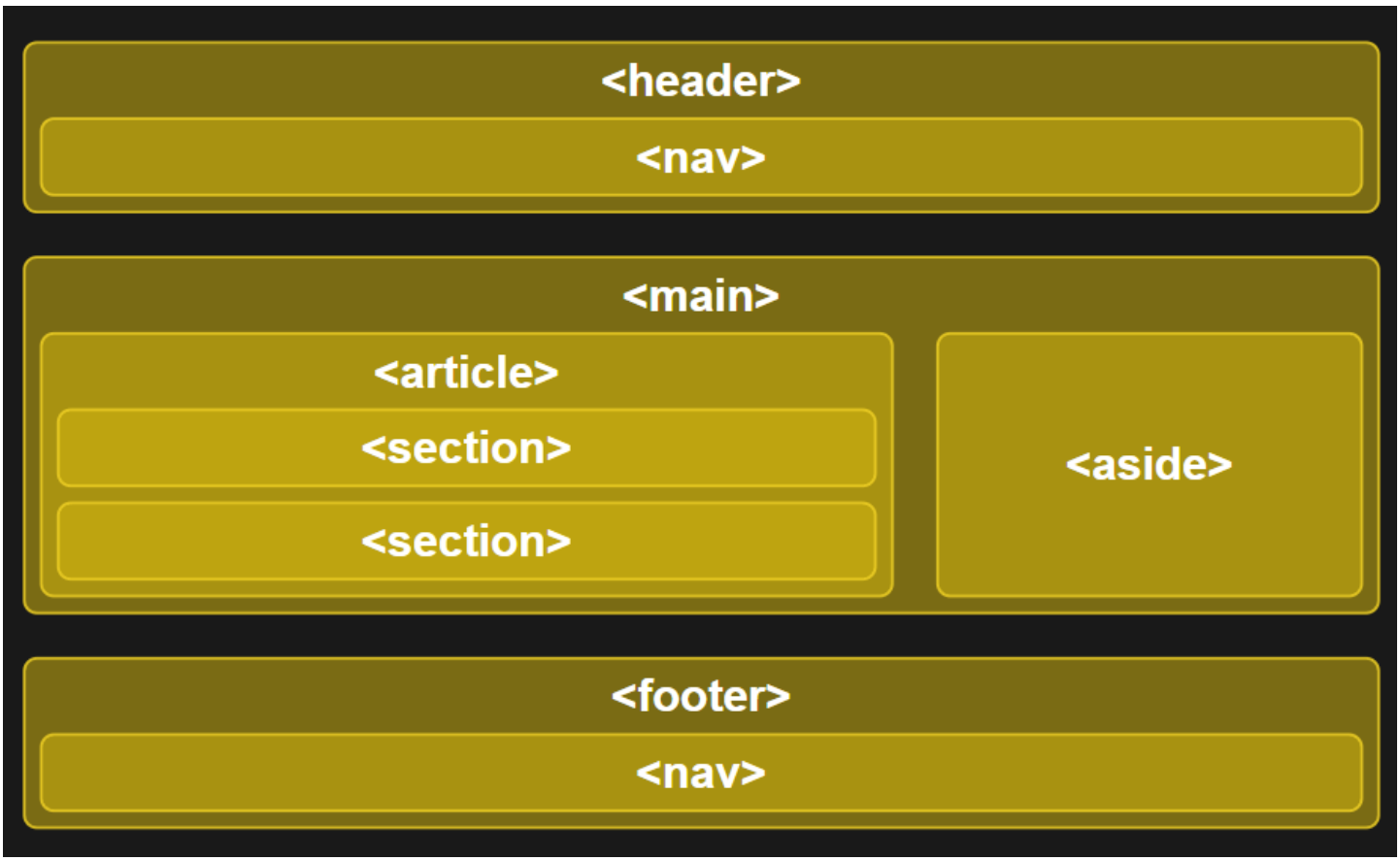
Jedním z cílů HTML5 bylo přidat tzv. sémantické tagy, které popisují různý obsah na stránce. Používání divu a podobných blokových prvků je fajn, ale velmi to limituje to, co může robot přečíst ze stránky. Když stránku rozdělíme pomocí níže uvedených tagů, tak může robot jednodušeji rozdělit stránku tak a vypreparovat to, co potřebuje.

Tagy jsou blokové a nemají většinou žádné základní styly. Tagy jsou následující:

- `header` — hlavička — obsahuje logo, navigaci a další
- `nav` — navigace — odkazy, mapy a další
- `article` — příspěvek — ucelený celek informací
- `section` — sekce — sekce příspěvku (můžeme však využívat i bez `article`)
- `main` — hlavní část — obsahuje informace, které se na stránkách často mění
- `aside` — boční část — nejsou hlavní informace, např. autor článku a podobně
- `footer` — patička — navigace, copyright a další

Často jsou stránky rozděleny například následovně:





Historie rozložení

Jak jsem si dříve říkali, tak se zejména pro rozložení v minulosti používali tabulky. Práce s tabulkami je špatná, protože nejsou responzivní a v CSS neumíme přesměrovávat různé věci. Obecně seřazení je v CSS špatné a proto vznikli nové technologie - flexbox a grid. A náhodou se obě naučíme!

Stránky se často taktéž umisťovali na levou část stránky a úplně se ignorovala šířka zařízení.

Flexbox

Flexbox se rozděluje na kontejner a předměty. Kontejner je takový obal okolo prvků a určuje jim právě to, jak se budou umisťovat. Tam taktéž zapínáme flexbox. Předměty jsou uvnitř flexboxu a můžeme jim určovat nějaké vlastnosti, které bude flexbox respektovat.

Flexbox zapneme tedy tím, že mu v CSS určíme:

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

Flexbox se dělí na dvě různé osy a to hlavní a sekundární osu. V základu je hlavní osa ta horizontální a sekundární vertikální. Flexbox však není dvourozměrné rozložení a osy slouží spíše pouze k zarovnání.

Na hlavní ose můžeme prvky zarovnávat pomocí vlastnosti `justify-content`. Na vedlejší pomocí `align-items`. Na hlavní ose máme k dispozici hodnoty:

- `center` - zarovnání doprostřed
- `flex-end` - zarovnání na konec flexboxu
- `flex-start` - zarovnání na začátku flexboxu
- `space-between` - zarovnání tak, že prvky budou mít co nejvíce místa mezi sebou
- `space-evenly` - zarovnání tak, že prvky budou mít mezi sebou stejné místa a to i okolo
- `space-around` - zarovnání tak, že prvky mezi sebou budou mít stejné místa a okolo budou poloviční

Na vedlejší ose můžeme zadat:

- `center` - zarovnání doprostřed
- `flex-end` - zarovnání na konec flexboxu
- `flex-start` - zarovnání na začátku flexboxu
- `stretch` - natáhne všechny prvky na danou dimenzi flexboxu

Jednotlivé osy můžeme prohodit pomocí vlastnosti `flex-direction` a hodnot `row` a `column`. Jednotlivé osy však můžeme i zalomit, to znamená, že když se přidá prvek, který se na osu nevejde, tak bude přesunut na "další řádek" osy. To ale neznamená, že máme dvourozměrné rozložení (přemýšlejte proč). Zalomení uděláme pomocí vlastnosti `flex-wrap` a hodnot `nowrap` a `wrap`.

Při tvorbě flexu je důležité přemýšlet jinak a v začátcích silně doporučuji jak jsou jednotlivé flexy propojené. Často zjistíte, že vlastně máte velmi mnoho flexů uvnitř sebe.

Pokud tato lekce není dostatečná pro nacvičení si flexboxu, můžete použít stránku ZDE, kde je flexbox velmi hezky vysvětlen.

Nové vlastnosti

Přetékaní

Novou vlastnost, kterou si ukážeme, je vlastnost `overflow`, která slouží k určení toho, jak se bude prvek chovat při tom, kdy jeho obsah je tak velký, že se do něj nevejde a přeteče. Vlastnosti `overflow` můžeme nastavit hodnoty:

- `hidden` - přetečený obsah nebude viditelný
- `visible` - přetečený obsah bude viditelný
- `auto` - zapne se skrolovací bar právě tehdy, když obsah přeteče
- `scroll` - skrolovací bar bude vždy viditelný

Stín

Vlastnost `box-shadow` slouží k vytvoření stínů okolo prvku. Jednotlivé stíny oddělujeme čárkou a každému stínu nastavujeme (v tomto pořadí):

- horizontální posun (povinné)
- vertikální posun (povinné)
- velikost rozmazání
- šířka nebo šíření stínu
- barva (povinné)
- pozice (pouze hodnota `inset`)

```
.fancy {
    box-shadow: 0 0 0 2em #b0f4aa,
                0 0 0 4em #66CCFF;
}
```

Filtr

Vlastností `filter` můžeme na celý prvek aplikovat nějaký styl. Jako ukázku můžeme použít rozmazání. Do hodnoty dáme funkci blur s hodnotou, o kolik se má prvek rozmazat. Hodnot existuje velmi mnoho, veškeré naleznete ZDE.

```
.blur { /* just visually! user can still copy the text and informations */
    filter: blur(20px) grayscale(1);
```

```
}
```

Transformace

Vlastností `transform` slouží k různým posunům a obecně úpravám prvků. Můžeme tedy prvky natáčet, zvětšovat, naklonit a podobné. Jednotlivé hodnoty můžeme spojovat a aktivovat zároveň. Veškeré hodnoty naleznete ZDE.

```
.item.transformed-2 {
    transform: rotate(1deg) scale(0.4) skew(30deg) scaleY(4);
}
```

Mezery

V rozložení často potřebujeme, aby jednotlivé prvky měly různé mezery mezi sebou. V CSS máme proto k dispozici vlastnost `gap`, které v rozložení flexbox (a nám neznámem gridu) dovoluje nastavit mezery mezi prvky jak v hlavní tak i vedlejší ose. Dovnitř můžeme dát jakoukoliv hodnotu a jednotku.

Mimo těchto vlastností můžeme mezery na jednotlivých osách ovlivňovat rozdělně a to díky vlastnosti `row-gap` a `column-gap`.

Formátování kódu

Pro jakékoliv programy a soubory v informačních technologiích je důležité, aby nám psaný kód či program byl zcela čitelný i pro ostatní lidi. Správný kód, syntaxe a návyky si může však určovat každá skupina lidí (open-source, práce, projekty, ...) určovat sama a je důležité dodržovat jejich pravidla. V zásadě však platí níže uvedené.

Každý soubor by měl použít pouze jeden typ odsazení a to buď mezery (většinou 4 na tab) a nebo jenom taby. Tímto způsobem by měl být celý soubor odsazen. K formátování můžeme používat např. vestavěnou funkci VSCode F1 > Format document a nebo si nainstalovat jiný plugin.

Pro WBA platí, že by třídy a identifikátory měly být anglicky, krátce a výstižně. Podporují se různé formáty, nejčastěji se však používá camelCase, snake_case a PascalCase. HTML by mělo být tak, aby bylo velmi sémanticky čitelné a HTML by se nemělo ohýbat kvůli formátování v CSS. V CSS používáme co nejvíce volné selektory, které půjdou dobře editovat. Např. tedy neděláme třídu pro každou věc, ale používáme vnořování a další selektory.

Pomocné vlastnosti pro flexbox

Pořadí prvků

Každému prvku ve flexboxu můžeme nastavit pořadí, v jakém se vykreslí nehledě na to, kdy byl definován v HTML. Čím větší hodnota, tím "později" budou. Vlastnost se jmenuje `order`. Pokud mají prvky stejnou hodnotu, tak má přednost prvek takový, který byl definovaný jako první (tj. první až posledně definovaný).

```
.first {
    order: -1;
}
```

Zarovnání na sekundární ose

Každý prvek v flexu poslouchá svého rodiče k tomu, jak se má na sekundární ose zarovnat pomocí vlastnosti align-items. Každému prvku však můžeme říct, aby toto nastavení ignoroval a pozicoval se dle sebe. Vlastnost se jmenuje `align-self` a hodnoty má stejné jako jeho ekvivalent v rodiči.

```
.special {
    align-self: center;
}
```

Zvětšení/zmenšení prvku

Každý prvek se ve flexu zvětšuje a zmenšuje stejně dle toho, co mu určí rodič. Tuto vlastnost zvětšení a zmenšení můžeme nastavovat každému prvku rozdílné a to pomocí vlastnosti `flex-grow` a `flex-shrink`. První určuje jako moc (kolikrát více) se bude moci prvek zvětšovat a druhý jak zmenšovat. Základní hodnotou je hodnota 1. Hodnota 0 říká, že se prvek nesmí zvětšovat a/nebo zmenšovat.

```
.donottouch {
    flex-grow: 0;
    flex-shrink: 0;
}
```

Základní velikost

Často potřebujeme ve flexu nastavovat velikost před tím, než na ni "někdo šáhne". Na to používáme vlastnost `flex-basis`. Vlastnost taktéž určuje automaticky, zda se nastavuje velikost na horizontální či vertikální ose. Flexbox se při rozdělování místa podívá na tuhle vlastnost a pomocné `flex-grow` a `flex-shrink`. Hodnota 0 na `flex-basis` znamená nejmenší možnou velikost.

```
.largeboi {
    flex-basis: 1000px;
}
```

Ikony

Velmi důležitá část webových stránek jsou tzv. ikony. Ikony nám dovolují zjednodušovat zobrazení tlačítek a obdobného na webových stránkách tak, aby uživatel nemusel číst hromady textu pro pochopení, kam má přejít. Ikony se používají všude, protože často jsou přehledné a designově zabírají méně místa.

Pro účely WBA budeme používat zejména knihovnu Font Awesome, která nabízí různé ikony zdarma a je velmi jednoduchá na použití. Budeme knihovny přidávat pomocí CSS, které naleznete v studijních materiálech zde na webové stránce. Veškeré ikony a použití naleznete na stránce ZDE.

Po přidání `link` u s URL, můžeme ikonu kamkoliv vložit pomocí:

```
<i class="fa-solid fa-trash"></i>
```

Ikony se chovají jako text (reálně se jedná o písmo) a proto je stylujeme zejména pomocí vlastnosti `color`.

Grid

Grid, společně s flexem, patří mezi moderní rozložení webů. Flex se spíše zaměřuje na řádkové rozložení a na vedlejších osách dělá minimum. Grid je spíše taková moderní tabulka. Jak si pamatujete, tak nevýhody tabulek jsou takové, že prvky uvnitř nemůžeme přesouvat tak, aby aby na telefonu byly jinde, než tabulka říká. Grid nám toto dovoluje a věci můžeme různě prohazovat.

Grid má tedy dvě dimenze - řádky a sloupce. Grid zapneme na prvku tak, že mu nastavíme vlastnost `display` na hodnotu `grid`. Grid dělíme, jak flex, na obal a jednotlivé prvky.

Gridu musíme minimálně definovat řádky a nebo sloupce. Řádky nastavujeme pomocí `grid-template-rows` a sloupce pomocí `grid-template-columns`. Hodnoty mohou být jakékoliv a jednotlivé řádky/sloupce rozdělujeme mezerou.

```
.container {
    display: grid;


    grid-template-columns: 50px 30px 40px; /* tři sloupce s velikosti 50px, 30px, 40px */
    grid-template-rows: 10px 60px 20px; /* tři řádky */
}
```

Jednotlivé předměty vkládáme dovnitř. Prvně se plní sloupce (tj. první řádek) a poté řádky. Pokud chceme, aby prvky zabrali celé místo, které jim gridem bylo přiděleno, musím nastavit výšku a šířku na 100%. HTML například vypadá:

```
<div class="container">
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
```

```
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
    <div class="item"></div>
</div>
```

Když přidáme do gridu další věci, tak se již mimo definice budou přidávat na nové řádky. Definice se potom zvětší na `auto`, což znamená maximální hodnotu na dané dimenzi. Mezery mezi jednotlivými řádky a sloupci můžeme nastavovat pomocí vlasto `gap`, `row-gap` a `column-gap`.

Pro jednotky definice řádků a sloupců můžeme taktéž používat jednotku `fr`, která reprezentuje část místa v gridu. Když máme např. 1fr 2fr 1fr v definice, máme 4 místa a prvnímu a třetímu dáme 1/4 a druhý bude mít 2/4. Například tedy:

```
.container {
    display: grid;
    width: 300px;

    grid-template-columns: 4fr 1fr 2fr;
    grid-template-rows: 10px 60px 20px;

    gap: 30px;
}
```

Věci v gridu můžeme zarovnávat (když nepoužijeme 100% na výšku a šířku) pomocí vlastností `align-items` a `justify-items`. Obě mají zvyklé hodnoty.

Samotný grid můžeme pozicovat pomocí `justify-content` a `align-content`. Tyto hodnoty se používají tehdy, když je možné, že grid je menší než to, co mu rodič dal.

V definicích můžeme jednotlivé hodnoty obalovat do funkcí `repeat` a `minmax`. `repeat(n, value)` způsobí, že se hodnota `value` bude `n`-krát. Když v definici použijeme např. hodnotu `12px repeat(4, 1fr) 3fr`, reálně tam bude hodnota `12px 1fr 1fr 1fr 1fr 3fr`.

`minmax(min, max)` nám zapne buď hodnotu `min` a nebo `max`, dle toho, kolik místa mu dá kontejner. Hodnota nikdy však dané místo nepřesáhne. Často používáme např. `minmax(0, 1fr)`, který nám řekne, že každý sloupec musí mít stejné místo.

Každému prvku kontejneru můžeme ručně nastavit v jakém řádku a sloupci se má zobrazit. Dokonce můžeme říct, od jakého do jakého řádku či sloupce se zobrazí. Na to máme složené vlastnosti `grid-row` a `grid-column`. Například:

```
.item.longer {
    grid-column: 1 / 3; /* zobrazí se na sloupci 1 a 2 */
}
.item.higher {
    grid-row-start: 1;
    grid-row-end: 3; /* stejné jako grid-row: 1 / 3; */
}
```

Responzivita

Responzivitu lze chápat jako schopnost stránky se přizpůsobit různým velikostem zařízení, na kterých je stránka zobrazena. Obecně to tedy znamená, že stránka, která funguje na desktopu, bude fungovat na mobilním zařízení a dalších zařízeních. Námi vyvíjené stránky toto neumí - jakkoliv zoomneme, posuneme či přeženeme velikosti prvků či velikost prohlížeče, stránka se rozbije a začne přetékat.

Responzivita stojí na tzv. responzivních jednotkách, mi je známe pod jménem relativní jednotky. Jedná se o jednotky, které se váží k jiné věci na stránce. Pro responzivitu budeme samozřejmě tedy používat %, em, rem, vw, vh a další jednotky.

Klíčové pro vytvoření správné responzivity je to, mít již hotový nějaký základ webové stránky, který zcela využívá výše uvedených jednotek, gridu a flexu. V případě, že budete používat pevné jednotky, stane se pro vás responzivita peklem. Při správně navrhnutých stránkách stačí změnit pouze pár věcí.

Otázku, kterou si budete pokládat je to, pro jaké zařízení vlastně máte responzivitu zařizovat. Nejjednodušší odpověď je pro všechny, protože chcete co nejvíce zákazníků. Pro účely WBA (a třeba i závěrečnou práci), počítáme s tím, že minimální velikost zařízení je 360px na šířku. Jako maximální nemáme žádnou hodnotu, ale bude praxí, že po velikosti 1980px na šířku se stránka přesune doprostřed stránka tam velikostně zůstane stejná.

Posunutí na střed stránky je velmi častá praktika nejen v responzivitě. Danému chování říkáme, že používáme kontejner (container v angličtině). Jedná se o prvek, který na celé stránce má stejnou velikost a určuje, jaký obsah se vykreslí uprostřed stránky. Uvnitř již používáme jen responzivní jednotky - samostatný kontejner však často má i absolutní jednotky. V responzivitě poté určujeme velikost daných kontejnerů a tím se stránka automaticky zmenšuje.

Pro responzivitu si definujeme nové vlastnosti a to:

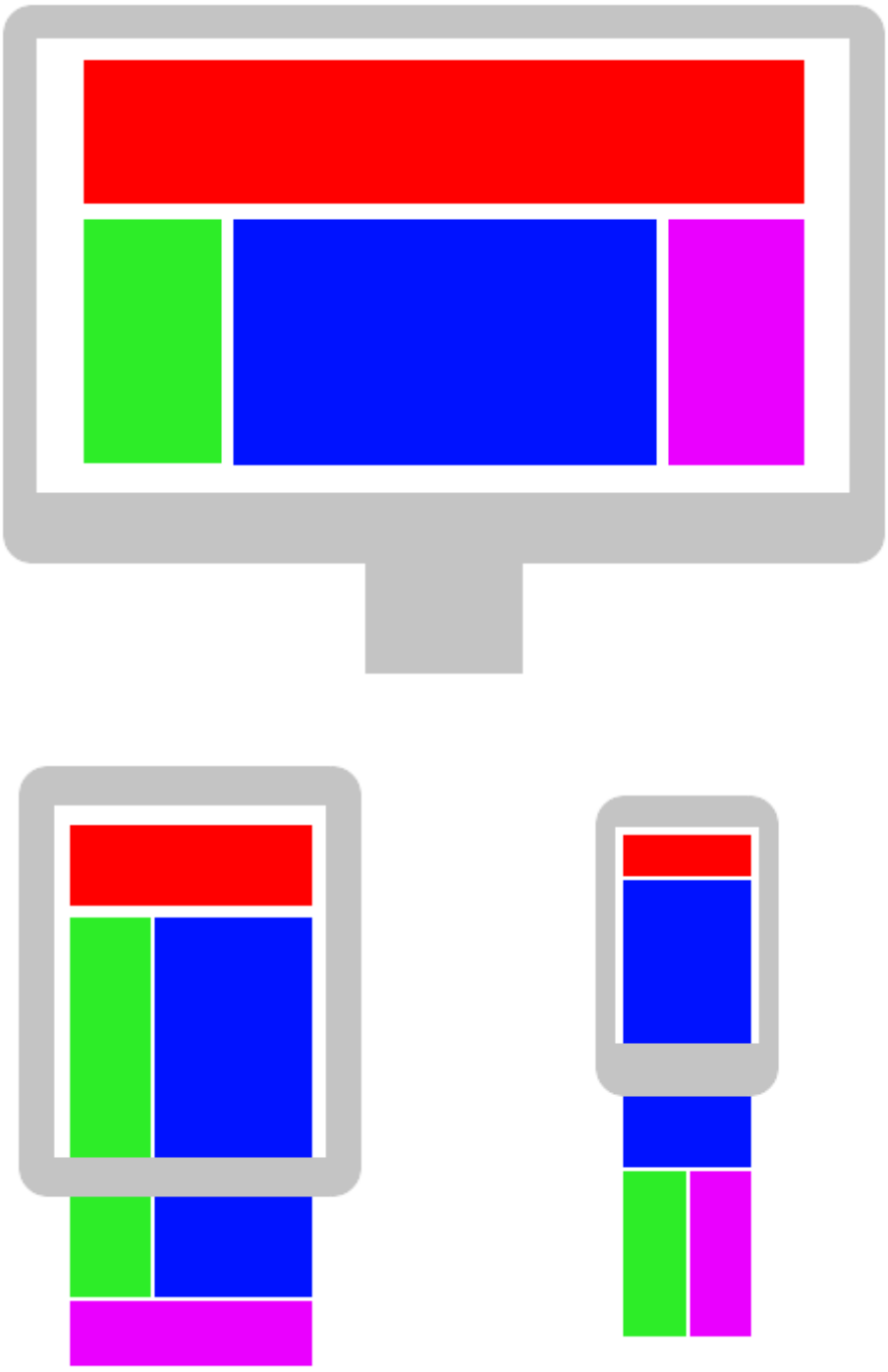
- `min-height` - určuje minimální výšku prvku
- `min-width` - určuje minimální šířku prvku
- `max-height` - určuje maximální výšku prvku
- `max-width` - určuje maximální šířku prvku

Tyto vlastnosti s absolutními jednotkami používám často v kombinaci s relativními jednotkami, abychom přesně definovali, kdy je prvek už příliš velký/malý.

Společně a nebo i mimo responzivitu se pro podobné účely používají další technologie, resp. způsoby, jak udělat stránku funkční na dalších zařízených. Často se dělá to, že desktop má vlastní stránku (vlastní html a styly) a pro mobilní verze existuje jiná verze stránky (vlastní html a styly), které se vykreslí právě tehdy, když stránku zobrazíme na telefonu. Často je taková stránka třeba na jiné subdoméně.

Návrh responzivity

Pro nejlepší pochopení responzivity je dobré si uvědomit, že responzivita je vlastně takové vlévání stránek (resp. jejich prvků). My se tedy snažíme prvky seřadit a upravit tak, aby aby se do dané velikosti vešli. Obrázek níže to názorně prezentuje:



Aby nám responzivita fungovala, je nutné v hlavičce nastavit / přidat následující meta tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Tento metetag říká, že zobrazení stránky má být takové, že šířka stránky je rovna šířce zařízení a to, že dovolujeme zoom, který je nastaven na hodnotu 1.

Obecně responzivitu řešíme pouze na šířku zařízení, protože většina webů funguje skrolováním na vertikální ose. Obrázky uděláme responzivní pomocí již známých vlastností:

```
img.responsive {
    width: 100%; // vyplň rodiče
    height: auto;
}
```

Jak jsme si již řekli, často se používá kontejnarizace stránky pro responzivitu, často to tak ale není, a používáme celostránkové stránky a nebo zcela kontejnery ignorujeme.

Když tvoříme responzivitu, je dobré začít od jednoho zařízení a přejít do druhého maxima. Nejčastěji se začíná na desktopu (desktop - tablet - mobil) a nebo telefonu (telefon - tablet - desktop). Vždy tedy stránku uděláme funkční pro jednu šířku / zařízení a poté zmenšujeme.

Media queries

Responzivita se na webových stránkách tvoří v CSS pomocí tzv. media queries. Media queries si můžeme představit jako podmínky - pokud platí nějaká vlastnost, zapni nějaké pravidla. A přesně tak se chovají. Zapisují se pomocí `@media () {}`, kde do kulatých závorek dáváme vlastnosti a do složených samotné pravidla (ty obyčejná, co známe z CSS), například tedy:

```
@media (max-width: 400px) {
    .container {
        flex-direction: column;
        width: 300px;
    }

    a { color: green; }
}
```

Výše uvedené pravila se aktivují právě tehdy, když stránka bude mít menší velikost než 400px, tedy např. pro hodnoty 200, 300, 350, ...

Mimo uvedené vlastnosti můžeme použít i klíčová slova:

- `print` - zaktivuje se právě tehdy, když se stránka tiskne
- `screen` - zaktivuje se právě tehdy, když je web zobrazený na obrazovce

Místo vlastností či klíčový slov můžeme dokonce uvádět i kombinace. Nejčastější je kombinace and, tedy, že obě dvě vlastností musí platit, například tedy:

```css
@media (min-width: 500px) and (max-width: 700px) {
    ul > li {
        text-decoration: line-through;
    }
}
```

Výše uvedené pravidlo bude aktivováno právě tehdy, když velikost zařízení bude od 500 do 700 pixelů.

Media queries kvůli kaskádě (pozdější pravidla mají větší prioritu) definujeme až ke konci všech CSS. Je nutné taktéž používat stejné selektory (a nebo vědět, co děláte), aby nedošlo ke snížení priority pravidla uvnitř media query. Pořadí pravidel lze vidět např. v DevTools.