WEB02-css

- Kulcs-érték párokat állítunk be fix értékkészletből olyan magától értetődő tulajdonságokra, mint szín és háttér
- *Deklaratívan* adhatjuk meg a dokumentum adott részeire vonatkozóan.

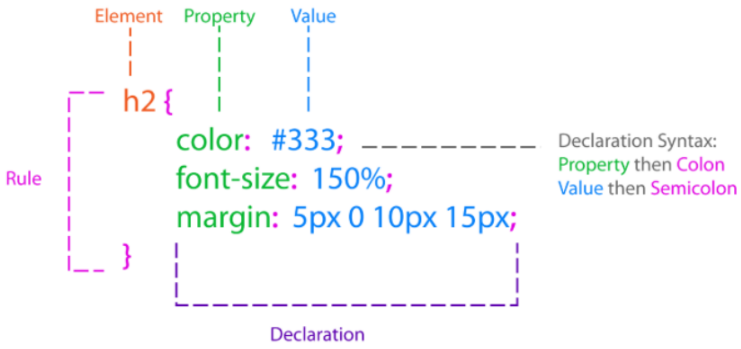
Inline stílusok

A HTML `<head>` tagjében megadhatunk stílusokat. Ezek adják majd az oldal designját. `<style>` tagek közt direkt megadhatjuk a kulcs-érték párokat is.

👎 Nem szeretjük, mert nehezen karbantartható, a HTML a tartalomért felelős, nem a designért!

- Szervezzük ki a stílusokat külön fájlba, és a head-hez adjuk őket hozzá:
  `<head><link rel="stlyesheet" href="style.css"></head>`

> 📋 Alapok
>
> A HTML oldal egy vagy több CSS fájlt is hivatkozhat.
> A szabályok **selector**ral kezdődnek, amik megadják, hogy milyen elemekre kell őket értelmezni.
>
> - Egyes selectorok magasabb prioritásúak másoknál, és felülírhatják egymás szabályait.



Selectorok

`*` : Univerzális selector. Az oldalon levő **összes elem**et kijelölhetjük vele.
`a {}` - tetszőleges HTML **tag**
`.osztaly {}` - tetszőleges HTML **osztály**
`#egyedi_azonosito` - HTML **ID** alapján
`a[href] {}` - ha létezik az adott attribútum
`a href=[*|$]` ha:
- *: Tartalmazza / egyenlő?
- $: azzal végződik, hogy:

Szóközök nékül teljes egyezés kell, egyébként pedig:
`nav > .main{}` - Tag *alatti* közvetlen osztály
`nav .main{}` - Tagen *belüli* osztály (összes leszármazott)
`img ~ p {}` - Utána következő *Testvér* **elemek** (ugyanazon DOM szinten)
`img + p {}` - Utána következő *Testvér* **elem**

Vizsgálat állapot alapján:
`:visited`: meglátogatott link
`:disabled` : letiltott inputok
`:readonly`: Csak olvasható input
`:first-child`: első gyerek
`:hover`: felette van az egér

Pszeudo elemek

`section::before`: Új HTML elem létrehozása első gyerekként
`section::after`: Új HTML elem létrehozása utolsó gyerekként
`::selection`: Kiválasztott elemek formázása
`::first-letter`: Első betűre egyedi megjelenés
stb...
Természetesen tetszőlegesen lehet ezeket kombinálni.

CSS reset

A böngészőknek van egy alapértelmezett stíluslapjuk. Vagyis **Egy elem, amit stílusozni akarunk, sosem "üres"**, mindig minden CSS tulajdonság be van rajta állítva. A spec nem definiálja, mi az alapértelmezett stíluslap tartalma.

- ha a HTML dolga, hogy a tartalom szerkezetét írja le, akkor nem helyes, hogy bárminek is legyen alapértelmezett stílusa.
- Erre jött ötletnek a reset CSS, ami igyekszik lehetőleg mindent "kinullázni"

```
html, body, div, h1, h2, h3, /* ... */ video {
margin: 0;
padding: 0;
border: 0;
font-size: 100%;
font: inherit;
vertical-align: baseline;
}
/* Ez nagyon basic, ennél lehet sokkal részletesebb is */
```

👍 Egy "üres vásznat" ad, ahol minden stílus, margó és egyéb tudatos döntésen alapul
👍 Nem kell azon gondolkodni, hol különböznek a böngészők stíluslapjai.
👎 A reset szó nem teljesen jó rá. Felülírja a böngésző alap stílusait egy üres érzetet keltő stílussal, amit aztán újra felülírunk.
👎 Ágyúval verébre: A komplexebb reset-ek olyan elemekre is rátehetnek stílusokat, amiket nem is fogunk használni.

CSS Normalize

Csak elfedni próbálja a böngészők közti alapértelmezéseket. Kiindulási alapként szokták használni, amit a dizájn igényeknek megfelelően módosítanak.

Blokk vs. Inline elemek

Az oldalon minden elem egy doboz. **A befogaló négyszög határozza meg, hogy mekkora helyet foglal el egy elem az oldalon.**

⚠ Az egyes típusok által felvehető tulajdonságok

Az, hogy egy elem block vagy inline, **nem csak a dokumentumfolyambeli viselkedésére van kihatással!**
- meghatározza azt is, hogy milyen doboz modell érvényesíthető rá,
- befolyásolja az általa felvehet CSS tulajdonságok *halmaz*át.

CSS mértékegységek

- Többféle CSS mértékegység islétezik, pl: `px, %, em, rem, vw, vh, cm, mm, pt, vm...`
- `%`: A szülő szélességének %-a
- `em`: A szülő betűméretéhez mérve, `rem`: a gyökér betűméretéhez mérve relatív.
- `vw, vh`: A viewport aktuális méreteihez képest relatív
- `vm`: A nézőpont szélessége és magassága közül a kisebbik
- `pt`: Pont
- `cm, mm`: centiméter, milliméter

Blokk elemek

- Mindig új soron kezdődik, és a szülője teljes szélességét kitölti.
  - az egymás után következő blokk elemek – ha a stílus felül nem írja – egymás alá rendeződnek. Ez a *Block flow*, iránya fentről lefelé.

👎 a `text-align, line-height, vertical-align` tulajdonságok nem hatnak rá.

- De ha megadunk neki ilyet, akkor az *inline gyerekelemek megöröklik*, vagyis közvetett hatása van.

📌 `width`

- Alapértelmezetten `auto`, ennek hatására a blokk kitölti a szülője szélességét.
- Többféle CSS mértékegységben is meghatározható.

📌 `height`

- Szintén `auto` az alapértelmezése. Ez itt azt jelenti, hogy a **tartalmazott elemek megjelenítéséhez szükséges magasság!**
- mindenféle CSS mértékegységgel állítható.
- **A % a szülő magasságának %-ában értendő**, nem mindig adja a várt eredményt!
  - A dokumentumfán felfelé **minden elemnek meg kell adni a magasságot, hogy jól működhessen!**

Inline elemek

- Egy soron belül követik egymást
  - leginkább szövegfolyamra illő elemekről van szó. Belesímulnak a szövegbe, nem kezdenek új sort.

👎 Nem reagálnak a szélesség, a magasság és a függőleges padding / margó beállítására.

👍 Reagálnak a:

- *vízszintes rendezés*re (`text-align`)
- *sormagasság beállítására* (`line-height`)
- *és az azon belüli rendezésre* (`vertical-align`)

📌 Egyes CSS property-k kiszakítják az inline elemeket a dokumentumfolyamból, *blokkszerűvé* alakítva őket. Beállíthatóvá válik így rajtuk a `width, height` érték.

Inline-block elemek

Inline:

👍 Az elemek szépen egymás mellé kerülnek, és reagálnak a vertical align-ra

Block:

👍 Játszhatunk a margókkal, paddinggel, méretekkel
Nem lehetne mindkettőből egy kicsit?

❗ Az elem az inline flow része marad, de emellett blokk tulajdonságok is beállíthatóak lesznek rajta!

- pl: menüben az elemek egymás mellé helyezése

❗ A **button** és **select** (egyes böngészőkben az *input*) alapból inline-block.

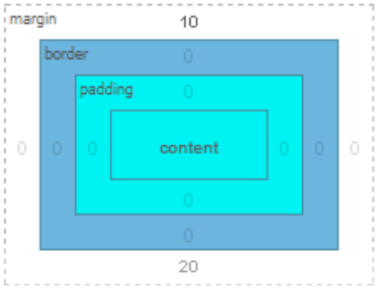
Blokkosítás

Bármely inline elem blokk elemmé alakítható a `display` tulajdonság `block`–ra állításával.

Margin, padding, box model

A **margó** a blokk *köré*, a **padding** a blokk *külső körvonala és a tartalma közé* helyez térközt.

⚠️ Ha vízszintes margók találkoznak, akkor azok összeolvadnak és csak a nagyobb lesz látható! -> **margin collapse**
- Megakadályozható pl azzal, hogy bevezetünk közéjük bordert (pl átlátszót)
- Vagy csak meghatározunk szabályokat saját magunknak, mondjuk hogy *margót csak lefele, paddinget csak felfele* rakunk.



Border-box, és content box méretezés

Nem egyértelmű, hogy minek a mérete a `width x height`.

Border-box: ha egy dobozra gondolunk, akkor a `width x height`ba annak széleit is beleértjük.
Content-box: a `width x height` csak a tartalom mérete kell legyen, és a padding sem számít bele.

Ha belszámoljuk a bordert is és a paddinget is, akkor túlmehetünk a 100% szélességnél! Ilyenkor egymás alá töri őket a layout motor. A megoldás css3-ban a `calc()`: `width: calc(50%-2px)`;

Támogatottság

Nincs mindent támogató böngésző, biztosítani kell a lehető legjobb tartalommegjelenítést mindenhol. Megoldás: `@supports` blokk:

@supports blokk

- Ha egy `property: érték` pár támogatott, érvényre jutnak a szabályok, ha nem támogatott, akkor nem jutnak érvényre.
- Ha egy böngésző egyáltalán nem ismeri a `@supports` blokkot, a feltételtől függetlenül figyelmen kívül hagyja a tartalmát.
- Fontos figyelni arra, hogy a `@supports`-os szabály legyen később (erősebb).

```
@supports (display: grid) {
        .main {
                display: grid;
        }
}
```

vendor prefixek: a böngészőgyártók megegyeztek abban, hogy *minden új propertyt prefixszel látnak el, amíg nem véglegesedik a specifikációjuk*, hogy elkerüljék a névütközést a később "szabványra emelkedő" verzióval.

👎 Redundánsak, könnyű őket elrontani
👎 Nehéz debugolni a viselkedést

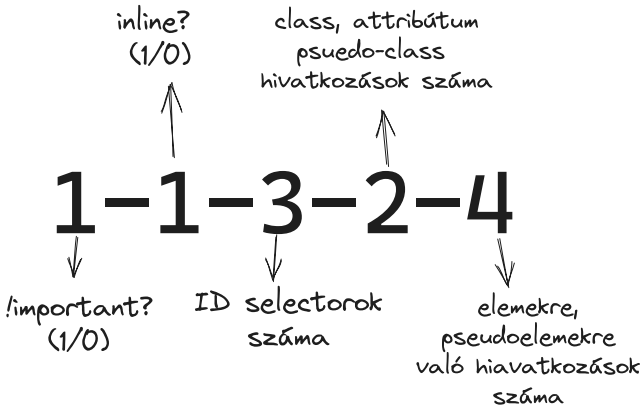
Stíluslapok forrásai

- Rengeteg helyről érkezhetnek a stíluslapok
  1. <link> tag
  2. `@import` egy másik css fájlra hivatkozva
  3. <style> tag (*HTML*-ben megadott CSS szabályok)
  4. `style` attribútum (tag-re téve közvetlenül)

  Valamilyen súlyozást kell alkalmazni, hogy az ütközések esetén eldöntsük, melyik stíluslap a fontosabb.

Súlyozás

1. Fontosság
   - Fontos, ami `!important`, minden más nem fontos.
   - ❗ Karbantarthatósági problémákat okozhatunk vele
   - Lehetőleg ne használjuk
2. Származás
   - A **fontosság** és a származási helye együttesen meghatároznak egy fontossági sorrendet
3. **Specifikusság**
   - a selector szintjén dől el a fájlon belül, a nagyobb „specifikusságú” selector az erősebb.
   - Négy (igazából 5) helyiértékből álló számsor:



Példa:

```
header#myhead ul#main-nav li
  > .sub-menu li.child
  a:not(.unimportant):hover {
     color: red;
}
```

0-0-2-4-5

```
!important? 0
    inline? 0
        ID: 2 # main-nav, # myhead
     class: 4 .sub-menu, .child, .unimportant, :hover
    elemek: 5 header, ul, li, li, a
```

4. Forrás sorrend
   - Ha azonos a súly és a specifikusság, akkor a "később" (lejjebb a fájlban) definiált fog érvényesülni.

Öröklés

Bizonyos tulajdonságok megörökölhetik a szülő elemen beállított értéküket.

- Egyes tulajdonságok eleve így viselkednek, de mi is előidézhetjük az `inherit` érték beállításával.
- Ha pedig esetleg a tulajdonság CSS ajánlás szerinti alapértelmezett értéket szeretnénk visszaállítani, az `initial` kulcsszóval tehetjük meg.

Dokumentum folyam

A HTML alapvetően egy dokumentumformátum. Ezért a HTML-ben szereplő tartalom dokumentumszerűen próbál viselkedni, balról jobbra, fentről lefele folyó szövegként. Ez a **normál dokumentumfolyam**.

Webalkalmazások készítésekor azonban gyakran kell dolgokat egymás mellé, vagy „Z irányban” egymás fölé tenni. Ezek egyrészt kitörtnek a folyamból, másrészt befolyásolják a folyam működését.

Blokk folyam viselkedése

Mivel a blokk elemek **egymás alatt** helyezkednek el, a **box-modell** tulajdonságok változtatása (margó, magasság stb.) **nem befolyásolja** az előttük levő elemek elhelyezkedését, de **szinte mindig befolyásolja az utánuk jövő elemek** elhelyezkedését!

- Az elemek folyambeli viselkedését a már említett `display: block | inline` beállításával tudjuk testre szabni.
- De sok egyéb más értéke is lehetséges, pl: `table-column, table-cell, flex, list-item`
- a **none** is a display propertyn állítható be.

A `position` tulajdonság átállításával kivehetjük az elemet a normál folyamból, és általunk megadott koordináták mentén helyezhetjük el.

- Lehetséges értékei: `static, relative, absolute, fixed`
  - abban különböznek egymástól, hogy mihez képest értelmezzük a koordinátákat.

`position: static`

Ez az alapértelmezett. Ekkor **az elem a normál dokumentumfolyam része**.

- A `top, right, bottom, left, z-index` tulajdonságok hatástalanok
  - Akkor szoktuk beállítani kézzel, ha *felül akarunk írni* valamit.

`position: relative`

Az elem **eltolása a normál folyam szerinti helyéhez képest**

- Az eredetileg neki szánt hely **üresen marad**, a többi elem elhelyezkedése így nem változik.
- Az elem új z-rétegre kerül, az eredeti folyambeli elemek *fölé*. Az új réteghez új koordinátarendszer is dukál, az elem gyerekelemei abszolút pozícionálása ehhez az elemhez képest lesznek abszolútak.

Mikor használjuk:

- Ha egy rossz assetet (pl kép) 1-2 pixellel arrébb akarunk tolni, anélkül, hogy bármi is változna az oldalon
- Ha szeretnénk egy normál folyambeli elemet *normál folyamon kívüli elem fölé helyezni anélkül, hogy a helyét megváltoztatnánk.*
- Ha *a gyerekelemeket abszolút pozícionálni szeretnénk, de nem a dokumentumhoz, hanem ehhez az elemhez képest.*

`position: absolute`

Az elem tetejének, bal oldalának stb. precíz elhelyezése az aktuális koordinátarendszerben.

- *Aktuális koordinátarendszer*: a legközelebbi szülő, ami `position: relative`, ha nincs ilyen, akkor a <body>
- Teljesen kikerül a normál folyamból, a többi elemet úgy rendezi a böngésző, mintha ő nem is létezne.
- Új z-rétegre is kerül
- **blokkszerű viselkedés**t is felvesz. Reagál a `width, height` tulajdonságokra.

- de **nem tölti ki a szülő szélességét**! ehelyett a gyerekelemek számára minimálisan szükséges méretet veszi fel, ha mást nem mondunk.
  => gyakran kézzel végzünk `width` és `height` beállítást rajtuk.

👍 **full control**

👎 rugalmatlan, csak fix dokumentumméretre működik jól.

👎 Nekünk kell figyelni, hogy ne takarja ki a normál flow elemeit (margó vagy padding használatával)

> ⓘ Mikor használjuk?
>
> Kisebb komponenseken belül használjuk:
> - **Több rétegre van szükségünk** vagy pontos elhelyezésre
> - De **nincs sok más elem, amire figyelnünk kell menet közben**.
> Ha ügyelünk arra, hogy a komponens gyökere helyesen működjön a normál folyam részeként, és arra alapozzuk a koordinátarendszert, többnyire megússzuk a karbantartási problémákat.

`position: fixed`

Mint az absolute, de itt az elem akkor sem mozog az oldallal, ha elpörgetik.
A top, left stb. koordinátarendszere: a legközelebbi szülő, amin a transform bármire be van állítva, ha nincs ilyen, akkor a böngészőablak belső széle.

`z-index`

❗ `position: static`-tól eltérő beállítás esetén minden elem új rétegre kerül.
Ezeknek a sorrendjét kezeli a `z-index`. **A nagyobb értékű kitakarja a kisebb értékűt**.
- Megadhatunk negatív értéket is.
- Alapkértelmezetten: `auto`: a HTML-ben később következő elem kitakarja a korábban szereplőt.

- **Nem globális számról van szó, a szülők (és egyéb felmenők) értékei is számítanak!**

> 🖉 Új stacking context gyökér jön létre:
>
> - Ha 1-nél kisebb opacity-t állítunk be,
> - Bármilyen nem none transform érték
> - z-index-et állítunk be relatív, vagy abszolút pozícionált, vagy flexboxban szereplő elemen.

Normál folyam megtörése float-tal

Eredeti célja: **kép** (vagy más, dobozszerű tartalom) **körbefolyatása** jobbról vagy balról.

- Bár a folyamból kikerülnek, nem kapnak saját réteget, kivéve ha állítottunk a `position` tulajdonságon is

❗ Ha két elemet is ugyanabba az irányba „float-olunk”, egymás mellé kerülnek. Amint nem férnek ki egymás mellé, akkor *sortörés* következik. Kiválóan használható összettt layoutokra.
`clear` property:

- letiltja a körbefuttatást egyik, másik, vagy mindkét oldalon.