FLEXBOX MODELL

1. Bevezetés:

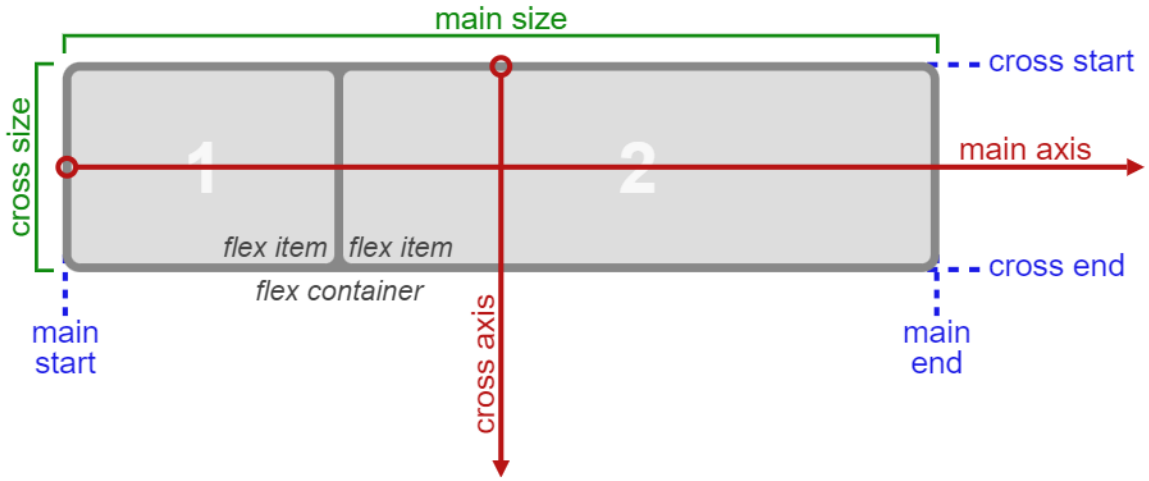
Mindenki, aki először találkozott a HTML oldalon található elemek pozícionálásának feladatával, emlékezhet, hogy mennyire nehezen lehetett megérteni a különböző pozícionáló módszereket, azok egymásra hatását. A probléma gyökere abban keresendő, hogy egy HTML oldal feldolgozásakor a böngésző alapértelmezetten sorra veszi az egyes sor– és blokkszintű elemeket, és vagy egymás mellé helyezi el azokat (inline), vagy egymás alá (block). Persze ezeket lehet aztán keverni is, de az biztos, hogy a kódolás során a megjelentetni kívánt elrendezést sokszor igen nehéz átültetni a HTML kódba.

A problémát persze részben meg lehet kerülni olyan keretrendszerek használatával, mint a Bootstrap, ahol kész CSS osztályok állnak rendelkezésünkre, melyek használata már sokkal rugalmasabb elrendezést tesz lehetővé.
Természetes módon született meg az igény arra, hogy ez a rugalmasság már a CSS szintjén megjelenjen, ezért jelent meg a CSS3– ban a Flexbox elrendezés.

2. Fogalmak:

A Flexbox elnevezés a rugalmas dobozmodell (Elastic Boksz Model) rugalmas doboz moduljának (Flexible Box Modul) rövidítése. Segítségével a HTML elemek elhelyezkedését, rugalmas méretét és a közöttük lévő távolságokat könnyen állíthatjuk be.

A modell két alapeleme a konténer (container) és az elem (item). Értelemszerűen a konténer tartalmazza az elemeket, és képes azoknak a szélességét és magasságát (sőt a sorrendjét is) úgy változtatni, hogy azok a lehető legjobban töltsék ki a rendelkezésre álló helyet. Képes akár az elemek szélességének növelésére, vagy akár csökkentésére, az esetleges átfedések elkerülése érdekében.



Fontos megjegyezni, hogy a flexbox elrendezés irányfüggetlen. (A block elrendezés vertikális, míg az inline horizontális alapú volt.)

- **main axis (főtengely)** – A konténer elsődleges tengelye, amely mentén az elemek elrendeződnek. Csupán a fenti ábrán helyezkedik el vízszintesen, a **direction** tulajdonságon keresztül függőlegesen is állítható.
- **main – start, main – end** – A konténer kezdetét és végét jelzi, ezen belül helyezkednek el a konténer elemei.
- **main – size** – Meghatározza a konténer méretét a főtengely mentén. Ez a főtengely irányától függően lehet a szélessége vagy a magassága.
- **cross – axis (keresztirányú tengely)** – A főtengelyre merőleges tengely. Iránya természetesen a főtengely irányától függ.
- **cross – start, cross – end** – Egy konténer elemekkel van feltöltve. Ezek az elemek a sortörés miatt akár több sorban illetve több oszlopban is elhelyezkedhetnek. Keresztirányban (tehát a keresztirányú tengely mentén) az elemek kezdeti és végpontját jelölik.
- **cross – size** – Meghatározza a konténer méretét a keresztirányú tengely mentén. Ez a főtengely irányától függően lehet a magassága vagy a szélessége.

3. Konténer és elemek:

Példa: `container példaprogram:`

Alapértelmezetten a HTML a ** ** elemeket (mivel blokk szintűek) egymás alatt helyezi el. Kapcsoljuk át a szülő ** ** elemet **flex** megjelenítésre. Ehhez készítünk egy container osztályt, melyet szülő ** ** elemhez rendelünk. (A **basic** stílus tartalmát már nem tüntetjük fel.) Annak érdekében, hogy lássuk a konténer méretét, keretezzük is be.

Példa: `container_2 példaprogram:`

Mivel a flexbox főtengelyének alapértelmezett iránya vízszintes, a konténer elemei is így rendezkednek el. Azt is láthatjuk, hogy az elemek nem töltik ki automatikusan a rendelkezésre álló teret, viszont a konténer igen. Amennyiben ez utóbbira nincsen szükség, a **display** tulajdonságot **inline – flex** -e kell állítani.

Példa: `container_3 példaprogram:`

Mi a továbbiakban a display: flex tulajdonsággal dolgozunk. Az elemek főtengely menti méretét a dobozmodell alapján számolja ki. Ezt mi fölülbírálhatjuk az elemekre alkalmazott **flex – basis** tulajdonsággal. (Ennek alapértelmezett értéke az auto, ilyenkor számolja ki maga, de megadhatjuk a konkrét méretet is.) Az alábbi példában a 2. elem kezdeti megjelenített méretét 100px –re állítjuk. (A flexbox ekkorára nyújtja az elemt.) Később még látni fogjuk, hogy ez a méret rugalmasan változhat, de ebből a kiindulási méretből.

Példa: `container_4 példaprogram:`

Azt már látjuk, hogy a főtengely mentén hogyan méretezik a flexbox, de mi van a keresztirányban? Növeljük meg a konténer függőleges méretét **300** pixelre.
Példa: `container_5 példaprogram:`

Azt tapasztaljuk, hogy a keresztirányú tengely mentén az elemek méretét hozzáilleszti a konténer méretéhez. Természetesen ha valamelyik elemre konkrét méretet (jelen esetben magasságot) állítunk be, akkor azon az átméretezés nem történi meg.

4. **A tengelyek iránya:**

Példa: `flex_direction példaprogram`

Ahogyan láttuk, két tengelyünk van, a főtengely és a rá merőleges keresztirányú. Alapértelmezetten a főtengely vízszintes, így a keresztirányú függőleges. A főtengely iránya a **flex – direction** tulajdonsággal állítható. Négyféle értéke lehet, melyet elnevezése önmagáért beszél:

- **row:** alapértelmezett, a konténerben az elemek sorban, balról jobbra jelennek meg.
- **row – reverse:** a konténerben az elemek sorban, jobbról balra, vagyis fordított sorrendben jelennek meg.
- **column:** a konténerben az elemek felülről lefelé jelennek meg.
- **column – reverse:** a konténerben az elemek alulról felfelé jelennek meg.

5. Az elemek sortörése, elemek sorrendje:

Példa: `flex_wrap példaprogram`
Ehhez beállítjuk a konténer szélességét 300px -re, hogy jobban látszódjanak a különbségek.

Mi történik, ha a főtengely mentén nem férnek el az elemek az oldalon? Méretezzük át most úgy az oldalt, hogy az előző példában a konténer elemei vízszintesen (pontosabban a főtengely mentén) ne férjenek el. Figyeljük meg, mi történik a „kilógó" elemekkel! Azt tapasztaljuk, hogy jelen esetben elég csúnyán kilógnak az oldalból. Az elemek sortörése a **flex – wrap** tulajdonsággal tudjuk befolyásolni:

- **nowrap:** nem történik sortörés
- **wrap:** a kilógó elemeket új sorba helyezi
- **wrap – reverse:** a kilógó elemeket új sorba helyezi, de fordított sorrendben

Példa: `flex_flow példaprogram.`

A főtengely irányától függően természetesen a sortörés függőleges irányban is történhet.

Lehetőségünk van a főtengely irányát és a sortörést egyetlen tulajdonságon keresztül is beállítani, ez a tulajdonság a **flex – flow**. Két értéket lehet megadni, az első a főtengely iránya, a másik pedig a sortörés módja. (Ezek az értékek megegyeznek a korábban látottakkal.) Az alábbi példák most a függőleges főtengely menti sortörés eseteit mutatják meg, hiszen a vízszinteset a fentiekben már láttuk, igaz 2 külön tulajdonságon keresztül. A konténer magasságát 300px -re állítottuk, hogy legyen sortörés. (Ne lepődjünk meg, hogy **nowrap** esetén az oldal elég érdekes képet mutat.)

A konténeren belüli elemek sorrendjének meghatározására rendelkezésünkre áll egy másik lehetőség is. Az elemeket sorszámmal azonosított csoportokba sorolhatjuk az **order** tulajdonsággal (a sorszám negatív, nulla vagy pozitív lehet). Azok az elemek, melyek kisebb sorszámú csoportokba vannak sorolva, a megjelenítésben előrébb kerülnek, mint a nagyobb sorszámúak. Ügyeljünk arra, hogy az alap sorrendet a tengelyek iránya és a sortörés iránya határozza meg először, a csoportokba tartozást ezek után értékeli ki a böngésző. Alapértelmezetten minden elem a 0 -s csoportba tartozik.

Ahogyan látjuk 4 csoportot hoztunk létre:

- **-1 – es csoport:** 4. és 5. elem

- **0 -s csoport:** 1., 2. és 3. és 6. elem
- **1 – es csoport:** 9. és 10. elem
- **2 – es csoport:** 7. és 8. elem

Az elemek sorrendje ennek megfelelően úgy alakul, hogy először a -1 – es csoport elemei jelennek meg (hiszen ez a legkisebb sorszámú), azután a 0 –s, majd az 1 –es, végül a 2 –es csoporté. (Az azonos csoportba tartozó elemek sorrendjét természetesen az határozza meg, ahogyan egymás után jönnek.) Nézzük meg most, hogy ez a sorrend hogyan változik, ha megfordítjuk a tengely irányát.

A fenti sorrend nem változik, csak most nem balról jobbra, hanem jobbról balra kell olvasni azokat.

6. Az elemek igazítása:

Láthatjuk, az elemek szorosan egymás mellett, balról jobbra helyezkednek el a vízszintes főtengely mentén. Igazítsuk most ezeket az elemeket. Négy tulajdonság is rendelkezésünkre áll:

- **justify – content:** a főtengelyen állítjuk a boxunk tartalmának elhelyezkedését.
- **justify – self:** A főtengely mentén állíthatjuk be, hogy az adott eleme hol jelenjen meg, az adott elemre kell beállítani és nem a tárolóra.
- **align – self:** az adott elem hol jelenjen meg a kereszttengely mentén a tárolóban, ez a beállítás az adott elemre kell beállítani!
- **align – content:** a kereszttengely mentén igazítjuk a tartalmat.

Kezdjük a **justify – content** tulajdonsággal. Példa: `justify_content példaprogram`.

Az első három érték hatásához nem szükséges magyarázat. A második három az elemek közötti teret szabályozza, közülük a **space – around** minden elem bal és jobb oldalára (vagy fölé és alá, ha függőleges a főtengely) azonos térközt helyez el, a **space – between** az elemek között azonos távolságot állít be úgy, hogy az elemek a konténer elejéhez és végéhez illeszkednek, míg a **space – evenly** a konténer elejéhez és végéhez is hozzárendeli az azonos szélességű térközt.

A **justify – self** értékei lehetnek **flex – start**, ami a tároló elejére; a **center**, ami a tároló közepére és a **flex – end**, ami a tároló végére helyezi az elemeket. A stretch beállításnál az elemek kitöltik a teljes tárolót, a baseline beállításnál az alapvonalhoz igazít a legnagyobb távolság alapján a kereszt irányú kezdővonaltól.

Az **align – self** tulajdonsággal az egyes elemeknek külön – külön állíthatjuk az igazítását. Annak érdekében, hogy jobban megfigyelhessük a hatását, ha az egyes elemek magasságát is állítjuk.

Ahogyan a fenti példában látható, az elemeket a keresztirányú tengely mentén a tengely elejéhez, közepéhez vagy végéhez illeszti, nyújtja (ez az alapértelmezett), illetve az elemben található szöveg alapvonala lesz az illesztési pont.

Az **align – content** tulajdonsággal befolyásolhatjuk, hogy amennyiben sortörés miatt több sorban jelennek meg az elemek, a sorok milyen messze legyenek egymástól. Hogy láthassuk a hatását, a konténer szélességét csökkentsük le az elem szélességére, így azok egymás alatt fognak megjelenni.

Ahogyan látjuk, az elemek sorait a konténer tetejéhez, közepéhez vagy aljához igazítja, vagy kinyújtja azokat a teljes magasságra, illetve kétféle módon térközt helyez el közéjük.

Összevonás: **place – content**

Az **align – content** és **justify – content** összevonása.

Összevonás: **place – self**

Az **align – self** és a **justify – self** összevonása.

7. **Az elemek rugalmas méretezése:**

Korábban már találkoztunk a **flex – basis** tulajdonsággal, melynek segítségével a konténeren belül az elemek kezdő méretét lehet beállítani. Ez a méret azonban rugalmasan változtatható a **flex – grow** és a **flex – shrink** tulajdonságokkal. A **flex – grow** tulajdonság azt határozza meg, hogy az egyes elemek mekkora részt kapjanak a konténer maradék térközéből. (A maradék térköz az, amit a kezdeti méretükkel nem töltenek ki.) Amennyiben minden elemnél a **flex – grow** tulajdonság 1, akkor közöttük ez a maradék térköz egyenletesen lesz elosztva. Ha valamelyiknél 2 -re állítjuk a tulajdonság értékét, akkor számára 2 -szer nagyobb térköz jut, mint a többiek számára. (Tehát nem 2 -szer nagyobb lesz, hiszen a kezdeti méretük nem biztos, hogy megegyezett.) Ez a tulajdonság nem egész értéket is felvehet.

A **flex – shrink** tulajdonság akkor kap szerepet, amikor az elemek nem férnek el a konténer főtengelye mentén, és nincsen bekapcsolva a wrap. Amennyiben definiáljuk a **flex – shrink** tulajdonságot az elemeken, a böngésző megnézi, hogy mennyivel nyúlnak túl az elemek a konténeren, és mindegyikük méretét a **flex – shrink** tulajdonságban megadott értéknek

megfelelő arányban csökkenti. A következő példában a 2. elem 3 -szor akkora mértékben kerül rövidítésre, mint a másik kettő.

A **flex** tulajdonsággal egyetlen lépésben állíthatjuk be a **flex – grow**, **flex – shrink** és **flex – basis** értékeit ebben a sorrendben. (Alapértelmezett értéke a **0 1 auto**)