LINQ METÓDUSOK

ForEach körbejárja egy lista elemeit. Használatához előbb ki kell adni a ToList-et! **NAGYON TUDNI!!!**

Distinct Egyedivé teszi a listát. Eltünteti az ismétlődéseket/redundanciákat. Nem lehet neki lambda (x=>) kifejezést adni paraméterként. Tehát üres peraméteres

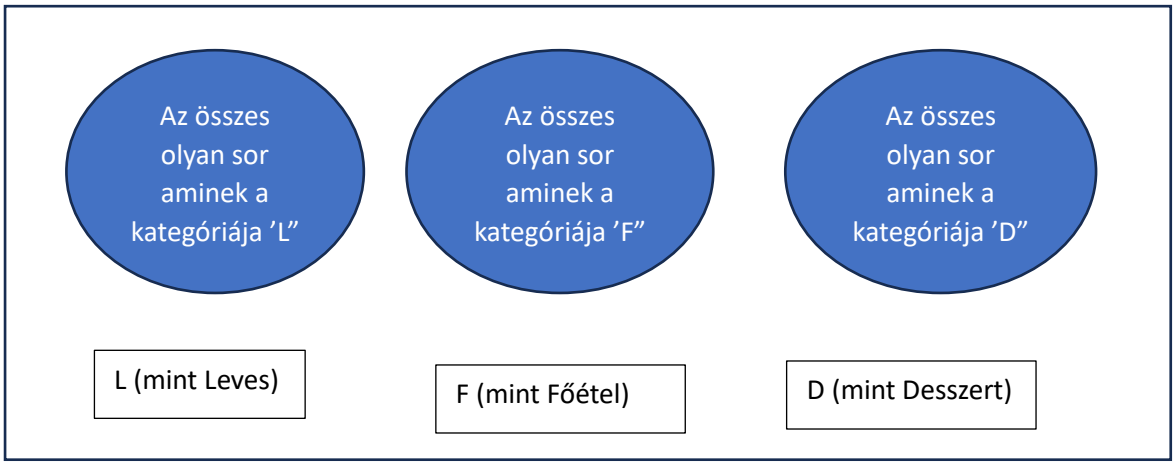
Select Transzformálja/átalakítja az eredeti listát, képezhet egyszerű listát (x=>valami) vagy akár egy teljesen új struktúrát (x=>new{prop1=.., prop2=..}) **NAGYON TUDNI!!!**

Average Segít kiszámolni az elemek átlagát, paraméterül meg kell mondani, hogy ha összetett elemekből áll a lista, melyik property-jére vonatkozva számoljon átlagot. Double a visszatérési értéke. **NAGYON TUDNI!!!**

OrderBy a lista elemeit növekvő sorrendbe rendezi. Meg kell mondani neki, paraméterben, hogy melyik property-je szerint rendezek. **OrderByDescending** ugyan ez csak csökkenő sorrend **NAGYON TUDNI!!!**

GroupBy csoportosítást csinál, elég absztrakt tud lenni, de a redundáns (ismétlődő) értékeket lehet csoportosítani. Pl hónapok, vagy kategória stb.

etelek.GroupBy(x=>x.kategoria) ezzel csoportok képződtek. Annyi a hossza amennyi csoport van. Azon belül pedig minden csoportnak van egy listája, ami a hozzá tartozó elemeket listázza. Így tudjuk például megszámolni, hogy mennyi elem van a levesben. Mennyi a főételben és az összes kategóriákban. **NAGYON TUDNI!!!!**



Any nem jellemző, hogy előfordulna vizsgán, de ha egy eldöntendő dolog a kérdés, azaz igaz-e/van-e ez vagy az a listában, akkor az Any fogja azt megmondani, mert vagy igaz (true) vagy hamis (false) értékkel tér vissza, a feltétel, a lambda (x=>x.neve.Contains(„csirke")) fogja megmondani, hogy van-e olyan sor amire pl igaz, hogy a neve tartalmazza a „csirke" szót. **A Contains-t is nagyon tudni (string típusoknál előszeretettel szokás kérdezni, hogy milyen sorokra igaz, hogy a neve pl tartalmazza ezt vagy azt)**

LINQ METÓDUSOK

Count Nagyon egyszerű, a lista hosszát, db számát adja vissza. **NAGYON TUDNI!!**

First kiszedi a lista első elemét. **Last** pedig az utolsót, ezeknek nem lehet lambda kifejezést adni paraméterként, azaz paraméter nélküli metódusok. PARAMÉTER NÉLKÜLI

Sum összeadja az elemeket, ha összetett adatszerkezetű a lista, ahol egy elem több property-ből áll, akkor meg kell mondani, hogy melyik elemét szeretnénk összeadni. pl Sum(x=>x.Ara) **NAGYON TUDNI!!**

Min a legkisebb **értéket** adja vissza. Paraméterként kell megadni, hogy melyik prorperty alapján szedje ki a legkisebb értéket. Ha viszont kell pl a legolcsóbb ételnek a neve, nem csak, hogy mennyibe kerül, akkor előbb rendezzük pl növekvőbe a listát, és szedjük lista első elemét, majd annak egy pont lenyomásával válasszuk ki a neve property-t. A **Max** is hasonló csak fordítva. **NAGYON TUDNI!!**

Where LECSÖKKENTI a lista elemeit a feltételnek megfelelően, mondhatnám úgy is, hogy mint egy szűrő működik, csak azokat a sorokat hagyja meg amelyekre igaz, amit paraméterül írtunk. pl etelek.Where(x=>x.Ara>500) azaz azokat a sorokat szűri ki, amikre igaz, hogy 500-nál drágábbak **NAGYON TUDNI!! NAGYON TUDNI!!**

ToList általában akkor használjuk, ha a ForEach-et meg akarjuk hívni. Listává alakítja a Linq-val manipulált szerkezetet. PARAMÉTER NÉLKÜLI!! **A FOREACH miatt úgy is kell!!**

Skip kihagy annyi elemet a listából, amilyen értéket írunk bele a paraméterébe. pl etelek.Skip(3) az első hármat átugorja, a többit pedig visszaadja, azzal tudnunk dolgozni, de abban már az első három elem nem lesz benne. **NAGYON TUDNI!!**

Take az előzőnek az ellentétje, a LIMIT SQL módosítónak felel meg. Szóval ha azt mondjuk, hogy etelek.Take(3) akkor az első három elemet adja vissza, a többit nem fogok látni, pl listázd az első három legdrágább leves nevét: **NAGYON TUDNI!!**

etelek.**Where**(x=>x.kategoria==”L”) .**OrderByDescending**(x=>x.ara) .**Take**(3) .**ToList**() .**ForEach**(x=>Console.WriteLine(x.neve));	**Where**→ leszűröm azokat a sorokat amire igaz, hogy a kategória property értéke L **OrderByDescending**→mivel az első három legdrágábbat kell ezért ár szerint csökkenő sorrendbe helyezem a listát, azaz a legdrágábbak legfelül lesznek **Take**→az első 3 legdrágábbra van szükségem ezért kiszedem az első hármat **ToList**→azért kell, hogy megtudjam hívni a ForEach-et, mert jelenleg már van egy listám, ami most már 3 elemű a Take miatt, már csak ki akarom írni a lista elemeinek a nevét. **ForEach**→ az összes elemét körbejárom és minden elemét, sorát, x-ét bele teszem a Console.WriteLine()-ba, de minden sorából csak a név propety kell, ezért x.neve kerül a WriteLine-ba