# Tartalom

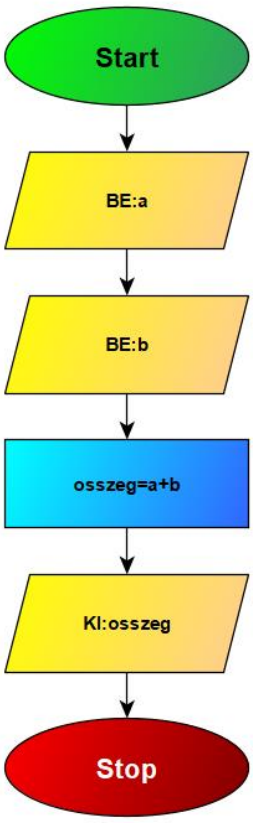
# Vezérlési szerkezet típusok

- **Szekvencia** (követés)
- **Szelekció** (elágazás)
  - *Egyirányú* (egyszerű elágazás, if)
  - *Kétirányú* (összetett elágazás, if-else)
  - *Többirányú*
    - Switch
    - If, else if, else
- **Iteráció** (ciklus)
  - *For* (előírt lépésszámú ciklus, számláló ciklus)
  - *While* (elől tesztelő ciklus)
  - *Do…While* (hátul tesztelő ciklus)
  - *Foreach* (bejáró ciklus)

# Szekvencia

A szekvencia a legegyszerűbb vezérlési szerkezet, amit a különböző programozási nyelvben használunk, használata igen egyszerű, ugyanis, ha szekvenciát használunk a megadott utasítások, sorban, egymás után hajtódnak végre.

Például, két szám összeadása, ahol a két szám megadása után egyszerűen elvégezzük az összeadás műveletét, majd kiíratjuk az eredményt, de természetesen nagyon sok féle egyszerű feladat létezik ennek a nem túl nehéz vezérlési szerkezetnek a bemutatására (gyakorlására).



# Szelekció

Az elágazás vezérlési szerkezetnek a programozási nyelvekben legtöbbször három formája van:

- egyszerű (egyirányú) feltételes elágazás
- összetett (kétirányú) feltételes elágazás
- többirányú feltételes elágazás

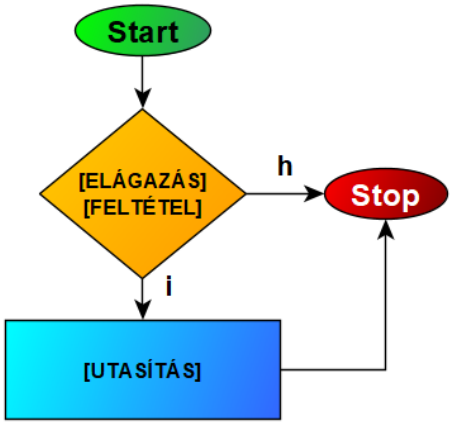
Minden elágazási szerkezet közös abban, hogy valamilyen feltétel teljesülése esetén (pl. egy változó adott értéket tartalmaz, vagy rendelkezik a feltételben megadott tulajdonságokkal) végrehajt egy ez esetre tartogatott utasításblokkot, mely egyszerű esetben egyetlen utasítást, gyakrabban azonban több utasítást (utasítássorozat) tartalmaz. A feltétel azt írja, le mely esetben kell ezt az utasításblokkot végrehajtani - ily módon védelmezi ezen utasításblokkot.

Ez a védelmezés akkor is fontos lehet, ha az utasításblokk végrehajtása csak adott körülmények között biztonságos. Pl. az utasításblokk egy **x** változó nézetgyökét számolja ki - mely csak akkor nem vezet hibára, ha az **x** aktuális értéke nem negatív. Ezért az utasításblokk végrehajthatóságát ehhez a feltételhez kötjük.

## Egyirányú (egyszerű elágazás)

Az elágazás létrehozásának kulcsszava **if**, amely létező angol szó, és magyar jelentése **ha**. Az **if** után kerek zárójelek között áll a **feltétel**, mely **logikai kifejezés** kell legyen - vagyis kiértékelésének eredménye vagy **igaz** vagy **hamis** kell legyen. A kerek zárójelek után áll a feltételhez tartozó utasítás vagy utasításblokk.

```
if(feltétel)
{
        //utasítás(ok) más néven utasítás blokk
}
```



## Kétirányú (összetett elágazás)

Az összetett elágazás kétirányú. Mivel ezt a logikai típusú kifejezések lehetővé teszik, így továbbra is őket használjuk.

Az összetett (kétirányú) elágazás két utasításblokkot tartalmaz. A kettőt az **else** kulcsszó választja el egymástól. Az **else** egy létező angol szó, magyar jelentése **különben**.

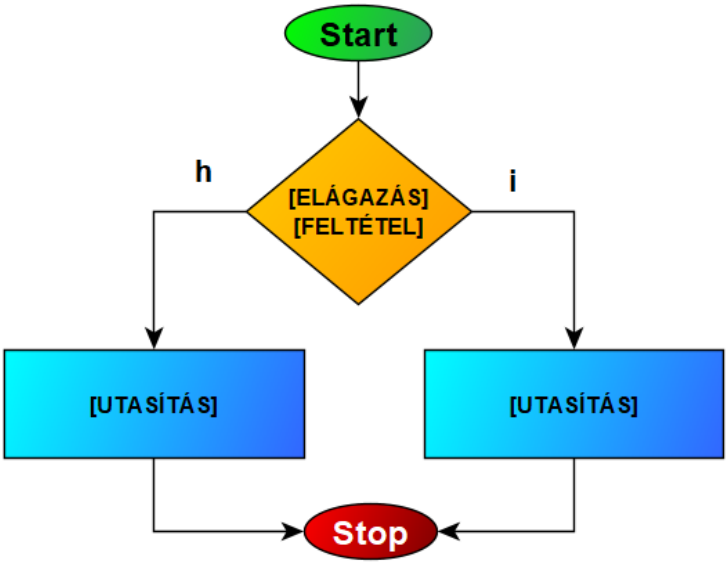
Az egyéb nyelveken, pl pascal, python, c# is létezik az **else** kulcsszó, mivel ott is szükséges elválasztani a két utasításblokkot egymástól. Az **else** nem csak elválasztja a két blokkot egymástól - de azok összetartozását is jelzi. Vagyis **else** hiányában egyszerű elágazást kapunk, az **if** és **else** együttese összetett elágazást jelent.

Az összetett elágazás szintén egyetlen utasításnak tekinthető a szekvencia elv szempontjából. Vagyis bármi is történik az **if-else** belsejében - a program fut tovább az **if-else** utáni utasítások végrehajtásával.

Az **if-else** végrehajtása a feltétel kiértékelésével kezdődik. Ha annak értéke **true**, úgy végre lefut az **if** utáni utasításblokk, és az **else** blokk nem fut végre. Ha a kifejezés értéke **false**, úgy az **if** blokk nem hajtódik végre, de az **else** blokk igen.

Ilyen módon az **if-else** két utasításblokkja közül az egyik blokk mindenképpen lefut. Hogy melyik - az a logikai kifejezés értékén múlik. Természetesen mindkét blokk tartalmazhat egyetlen utasítást vagy több utasítást is.

```
if(feltétel) //ha a feltétel igaz
{
        //utasítás(ok) más néven utasítás blokk
}
else //ha a feltétel NEM igaz
{
        //utasítás(ok) más néven utasítás blokk
}
```

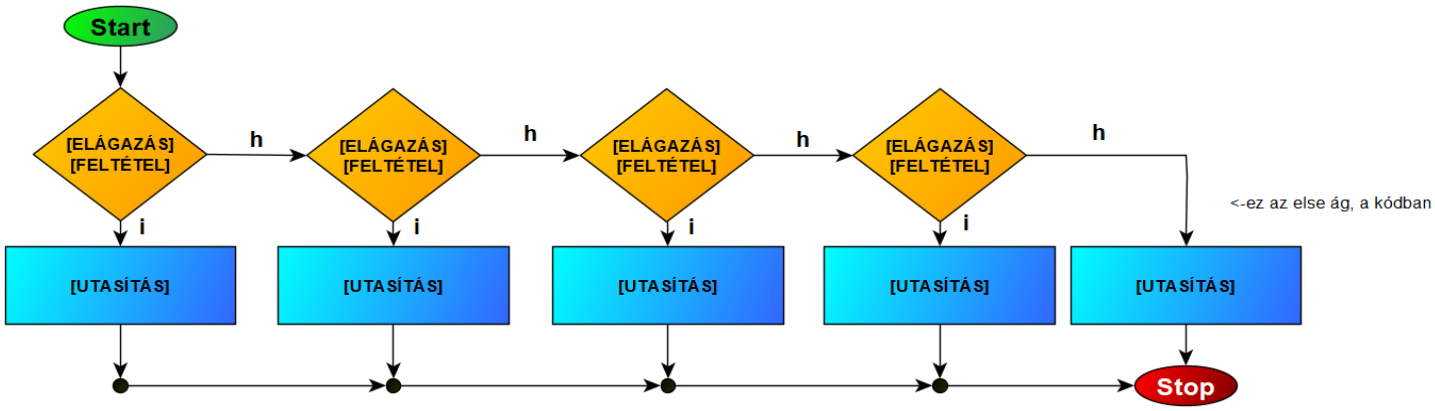


## Többirányú elágazás (if, else_if,else – switch)

Az **if** elágazás esetén a kulcs szerepkört egy logikai kifejezés tölti be. A logikai kifejezés két lehetséges értékkel bírhat - **igaz** vagy **hamis**. Ennek segítségével két elágazási irányban folytatódhat tovább a program végrehajtása.

Érdekes, de a két elágazási irány a programok tervezése során sokkal gyakrabban előfordul, mint első pillantásra látszik. A három irányú elágazást leggyakrabban például két darab összekapcsolt kétirányú elágazással oldjuk meg.

```
if(feltétel) //ha a feltétel igaz
{
        //utasítás(ok) más néven utasítás blokk
}
else if (feltétel) //ha az if feltétel NEM igaz
{
        //utasítás(ok) más néven utasítás blokk
}
else //ha EGYIK FELTÉTEL SEM igaz
{
        //utasítás(ok) más néven utasítás blokk
}
```



## Switch

Ugyanakkor a több irányú elágazás megoldására saját nyelvi konstrukció van - **switch** a neve. Ez angolul **kapcsolót** jelent, mely több irányba is elfordítható. A szintaktikára egy példa.

```
switch(vizsgálandó_változó_neve)
 {
   case vizsgálandó_változó_érték_egy:
    //utasítás(ok)
      break;
   case vizsgálandó_változó_érték_kettő:
    //utasítás(ok)
      break;
   case vizsgálandó_változó_érték_három:
    //utasítás(ok)
      break;
   default :
    //utasítás(ok)
 }
```

A **switch** után zárójelben meg kell adni az elágazás alapját jelentő kifejezést (mely legegyszerűbb esetben egyetlen változó). A kifejezés (vagy változó) típusa egész szám (number) vagy szöveg (string) kell legyen!

Az esetek az utasítássorozat belépési pontjai, tehát nem önálló programblokkok. Ezért is nem tartozik hozzájuk önálló kapcsos zárójeles blokk. Amennyiben mégis azt szeretnénk, hogy valamely esethez tartozó utasítássorozat végrehajtása után már egyéb switch belsejében szereplő utasítás ne hajtódjon végre, úgy ezt kérhetjük a **break** segítségével. A **break** hatására a switch hátralévő utasításai nem hajtódnak végre, hanem ugrunk a switch mögötti (rákövetkező) utasításra.

# Iteráció

A programok megírása során eddig felhasználtuk a **szekvencia** és **szelekció** programvezérlési szerkezeteket. Az előbbi segít abban, hogy az utasításaink ugyanabban a sorrendben kerüljenek végrehajtásra, mint amilyen sorrendben őket a program szövegében szerepeltettük. A második (elágazás) segítségével érhetjük el, hogy egyes utasításaink csakis akkor hajtódjanak végre, ha azokra szükség lenne. Egy feltétellel, vagy feltételrendszerrel írhatjuk le, melyekre van szükség, melyek hajtódhatnak végre a program futásának adott pontján.

A ciklusok akkor szükségesek, mikor valamely utasítást, vagy utasítások sorozatát nem csak egyszer, hanem többször is végre szeretnénk hajtani.

A ciklusok két jellemző részből állnak. Egyrészt definiálni kell ezt a bizonyos utasításblokkot, melyet ismételni szeretnénk. Ezt a részt **ciklusmagnak** nevezzük. Ezen felül szükséges megadni azt a részt, amely befolyásolja a ciklus ismétlését. Ehhez egy logikai feltételt használunk, mely értelemszerűen **igaz** vagy **hamis** értékű lehet. Ezt a feltételt **ciklus vezérlő feltételnek** nevezzük.

## For ciklus

Elsősorban a vektorok kezelésével kapcsolatosan gyakoriak azok a ciklus-alakok, ahol valamilyen változót végig kell futtatni **0..n-1** értékek között.

Az ilyen ciklusok közös ismérve, hogy valamilyen ciklusváltozó (jelen esetben az **i**) kezdőértékének beállításával indulunk (ez jellemzően 0 értékről indul), a ciklus vezérlő feltételében az **i** értékét nem engedjük, hogy elérje az **n** értékét, és minden ciklusmenet végén növeljük az **i** értékét 1-el.

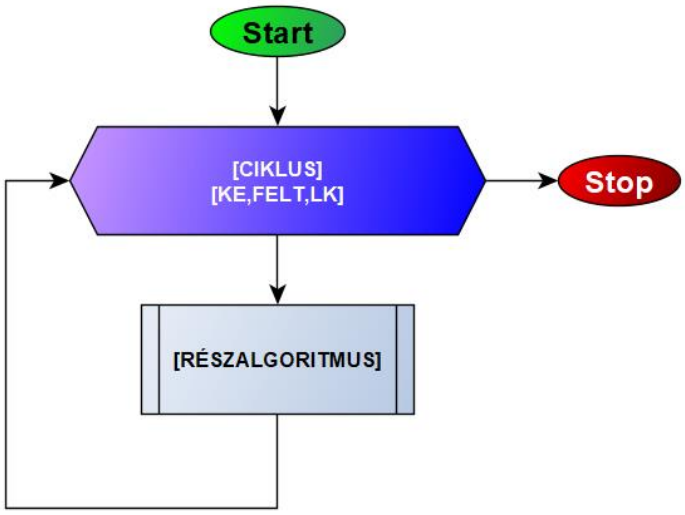
Az ilyen alakú ciklusok speciális alakja miatt saját ciklusfajtája alakult ki. Ez az alak annyira gyakori a programozásban, hogy ez a speciális ciklus valójában gyakrabban szerepel programkódokban, mint az eredeti őse, a **while** ciklus.

Mint láthatjuk, a **for** ciklusban pontosan a fenti három rész kerül kiemelésre és hangsúlyozásra, a **for** ciklus fejrészében. A **for** ciklus fejrésze e miatt mindig három részből áll:

- ciklusváltozó kezdőérték beállítása (ke)
- vezérlő feltétel (felt)
- léptető utasítás, vagy lépésköz (lk)

A három részt **kettő pontosvessző** határolja egymástól. A három rész egyike sem kötött szintaxisú, nyilván bizonyos megkötésekkel: pl. a középső résznek egy logikai kifejezést (feltételt) kell leírnia, míg az első és utolsó résznek egyszerű végrehajtható utasításnak kell lennie. Szokás a ciklusváltozót a for ciklus fejrészében deklarálni, de nem kötelező:

```
for(let i=0; i<n; i++)
{
  //utasítás(ok) jelen esetben ciklusmag
}
```



## While és do while ciklus

A **while** kulcsszó után az **if**-hez hasonlóan a logikai feltételt zárójelben kell megadni. A **while**-ról az előző fejezetben leírtak szerint tudnunk kell, hogy

- pozitív vezérlésű,
- elől tesztelő

ciklus, vagyis legelső lépésben máris a vezérlő feltétel kiértékelésével kezd, majd amennyiben az igaz értékű, úgy végrehajtja a ciklusmag utasításait. Ezek után újra a ciklus vezérlő feltételének kiértékelése következik. A két fázis egymást követve ismétlődik, amíg egyszer a vezérlő feltétel végre **hamis** értékű nem lesz. Ekkor a ciklus futása megáll (a program futása természetesen nem). A program futása a ciklust követő következő utasítás végrehajtásával folytatódik (a szekvencia szabályban foglaltaknak megfelelően).

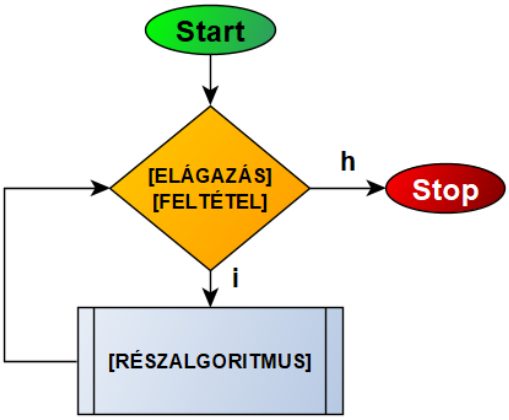
```
let i=0;
while ( i<n )
{
        //utasítás(ok) jelen esetben ciklusmag
         i++;
}
```

A **while** ciklus ennek megfelelően lehet, hogy egyetlen alkalommal sem hajtja végre a ciklusmagját, míg a másik véglet is előfordulhat: a vezérlő értéke minden kiértékeléskor **igaz** értékű lesz. Ez utóbbi esetben **végtelen ciklusról** beszélünk.

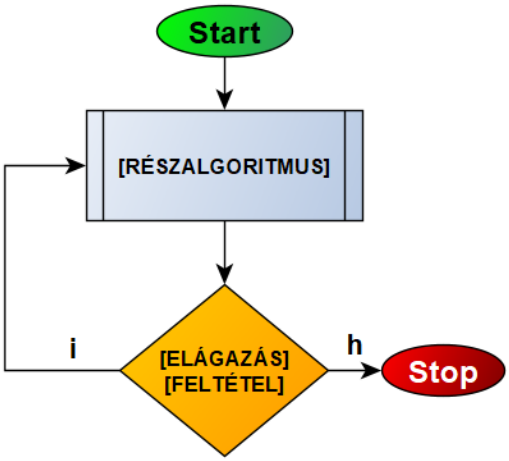
A do…while ciklus ezzel ellentétben hátul tesztelős, ami azt jelenti a feltétel vizsgálat az utasítás végrehajtása után hajtódik csak végre, így előfordulhat az, hogy az érték a feltéltelnek nem megfelelő, de az utasítás egyszer legalább végrehajtódik.

```
let i=0;
do
{
        //utasítás(ok) jelen esetben ciklusmag
         i++;
} while ( i<n );
```

**Előltesztelős (While) ciklus**




**Hátultesztelős (Do-while) ciklus**




## Foreach ciklus

A vektorok(egy dimenziós tömb) esetén tapasztalhatjuk, hogy gyakoriak azok a feldolgozások, amikor a vektor minden elemét egyesével elő kell venni, meg kell vizsgálni.

```
let sum = 0;
const tomb = [5, 3, 7, 9];

tomb.forEach(fuggvenyNeve);

function fuggvenyNeve (item) {
  sum += item;
}

document.write(sum);
```

A foreach ciklus nagyon egyszerűen használható, mindössze annyit kell tennünk, hogy megadjuk melyik tömböt járja be, és biztosak lehetünk benne, hogy ő a tömb minden elemén végigmegy!

***Üres összetett adatszerkezet viszont neki semmiképp sem adható.***

Hátránya egyedül az, hogy a megfelelő kihasználás érdekében külön függvényeket érdemes készíteni az elemek feldolgozására, mint ahogy az a mintában is látható, a bejárt függvény elemeit a mintában megadott módon akár össze is adhatjuk, a függvényekről a következő fejezetben olvashattok részletesebben!