Algoritmizálás és blokkprogramozás

Bevezetés és ismétlés

Előzetes tanulmányainkban a micro:bit mikrovezérlő számos funkcióját használtuk már. Megtanultunk animációt készíteni, a LED-kijelző pontjait felkapcsolni, lekapcsolni. Játékokat fejlesztettünk, melynek során gesztusokat és gombokat is használtunk, sőt, a rádiókapcsolat segítségével adatokat küldtünk, illetve fogadtunk.

Nemcsak mikrovezérlőt programoztunk, hanem robotjárműveket is. Megtanultuk, hogyan haladhat a jármű egyenes vonalban, hogyan foroghat egy helyben, hogyan kanyarodhat, illetve követhet vonalat. Munkánk során többféle szenzort használtunk (fényérzékelőt, távolságérzékelőt, színérzékelőt), önállóan terveztünk programot valós, gyakorlati probléma megoldására.

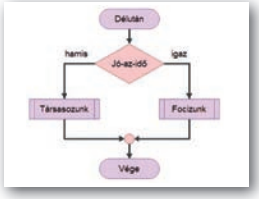
Folytatásként elmélyítjük tudásunkat, illetve új, izgalmas eszközökkel és lehetőségekkel is megismerkedünk.

Feladat

Párosítsuk a fogalmakat a képekkel!

1. változó
2. egyszerű elágazás
3. végtelen ciklus
4. kétirányú elágazás
5. gesztusok
6. szekvencia
7. logikai műveletek
8. számlálós ciklus
9. feltételes ciklus

A

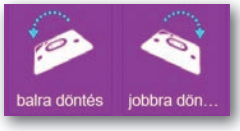


B

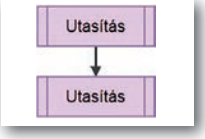
C

D

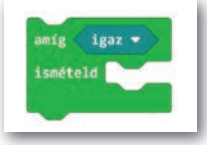
E



F



G



H

```
Ha feltétel akkor
   utasítás(ok)
elágazás vége
```

I

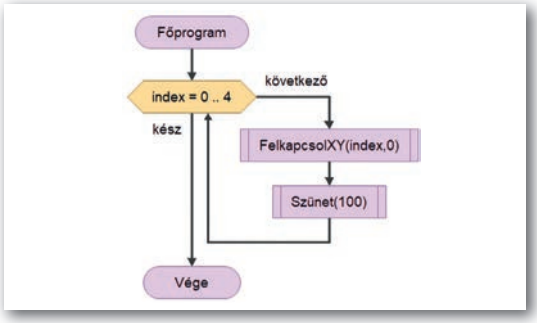
Játsszunk az algoritmusokkal! Egyszerű algoritmusok tervezése

Ismerkedés a Flowgorithm alkalmazással

Korábban számos példát láttunk arra, hogyan lehet az algoritmusokat szöveges, mondatszerű leírással, illetve folyamatábrával megadni.

```
Program
 Ciklus index=0-tól 4-ig
    FelkapcsolXY(index,0)
    Szünet(100)
 Ciklus vége
Program vége
```

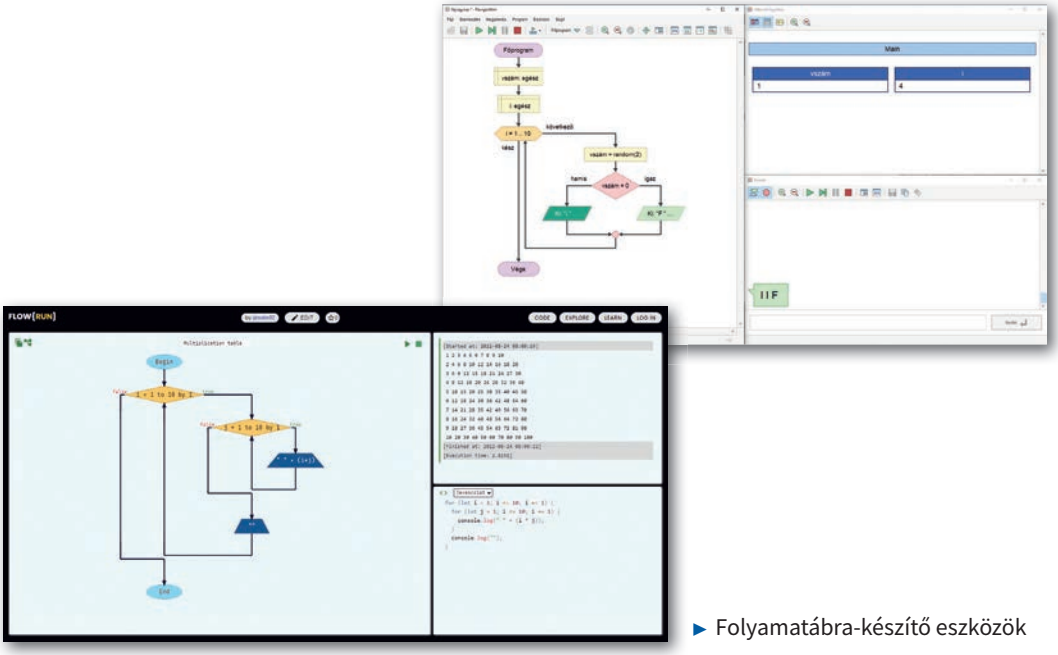
► Egy algoritmus mondatszerű leírása



► A bal oldali algoritmus leírása folyamatábrával

Milyen jó volna, ha lenne olyan eszközünk, amellyel ezeket a folyamatábrákat egyszerűen elkészíthetnénk, sőt akár ki is próbálhatnánk a működésüket! Ezzel ugyanis meggyőződhetnénk arról, hogy helyesen működik-e az algoritmus, vagyis tesztelni tudnánk. A tesztelés során megvizsgálhatnánk az egyes változók értékeit anélkül, hogy ki kellene őket írni. Sőt, egy korszerű eszközzel a folyamatábrát akár futtatható programkóddá is alakíthatnánk.

A jó hír, hogy van ilyen eszköz (például a *Flowgorithm*), a következőkben ezzel foglalkozunk.

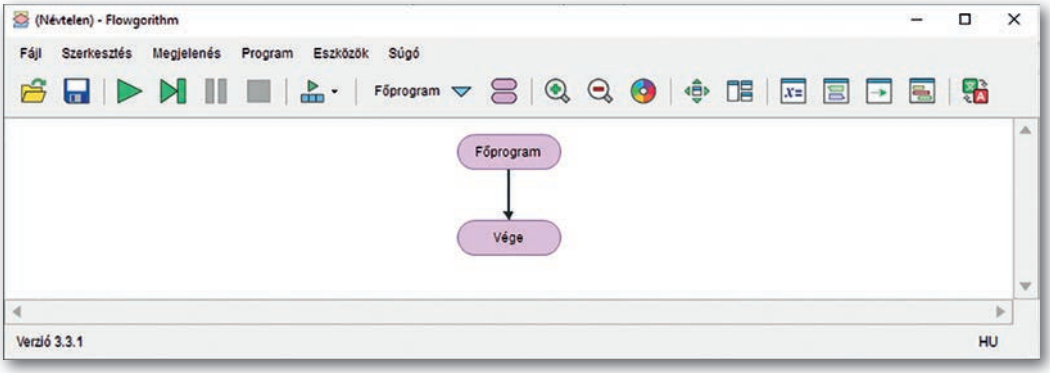


► Folyamatábra-készítő eszközök

A Flowgorithm alkalmazás felépítése, használata

Az algoritmusok készítését és tesztelését a *Flowgorithm* alkalmazással mutatjuk be. Ennek telepítőkészlete a http://www.flowgorithm.org/ webcímről tölthető le Windows operációs rendszerre. Más operációs rendszereken való futtatáshoz is találunk segítséget az oldalon.

Az alkalmazás neve a *flowchart* (folyamatábra) és az *algorithm* (algoritmus) szavakból származik. A program indítása után egy üres program folyamatábrája jelenik meg a képernyőn, amely két dobozból *(Főprogram, Vége)* és egy nyílból áll.

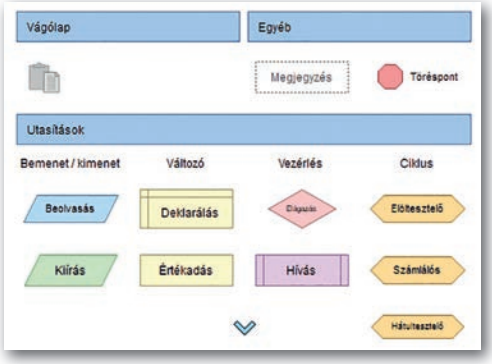


▶ A Flowgorithm alkalmazás kezdőképernyője

Láthatjuk, hogy a különböző típusú blokkok más-más színekkel jelennek meg a folyamatábrában. A programban számos színséma beállítható, amely befolyásolja az egyes blokkok színét. Érdemes kezdetben az alapértelmezett színsémát használni, és csak akkor átállítani másikra, ha már megfelelően tudjuk használni a programot.

A folyamatábrát úgy bővíthetjük új elemmel, hogy a két blokk közti nyílra kattintunk. Ekkor az alábbi elemekből választhatunk:

- A *Vágólap* segítségével a korábban vágólapra helyezett elemeket illeszthetjük be.
- Az elhelyezhető *utasítások* – algoritmuselemek – különböző csoportokba (például változó, vezérlés) vannak sorolva.
- Az *Egyéb* kategóriában található elemek segítségével pedig megjegyzést fűzhetünk a folyamatábrához, illetve töréspontot helyezhetünk el, amely a tesztelés során lehet fontos.

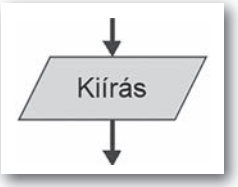


▶ Választható elemek és blokkok

Ha beillesztettünk egy elemet, akkor a jobb egérgombbal rákattintva előhívhatunk egy hasznos menüt. Itt megtaláljuk a *Kivágás,* a *Másolás* és a *Törlés* műveleteket.

Törölni a *Del* billentyűvel is tudunk, kivágáshoz a CTRL + X, másoláshoz a CTRL + C billentyűkombinációt is használhatjuk Windows operációs rendszerben.

Írjunk ki egy szöveget!

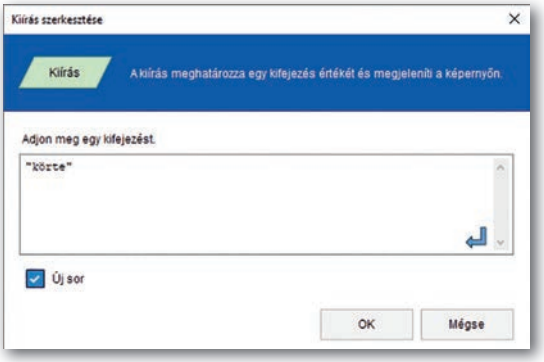


Kezdetnek készítsünk el egy egyszerű algoritmust, amely kiírja a kedvenc gyümölcsünk nevét! Ehhez használjuk a *Kiírás* utasítást! Azt tapasztaljuk, hogy a blokk a beszúrása után szürke színnel jelenik meg, az alapértelmezett színsémát használva.

Ez a szürke szín arra figyelmeztet bennünket, hogy a blokkot még módosítani kell ahhoz, hogy használható legyen. Ha ebben az állapotban próbálnánk futtatni az algoritmust, hibaüzenetet kapnánk.

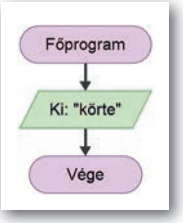
Módosítsuk tehát a *Kiírás* blokkot! Ehhez kattintsunk rá duplán!

A megjelenő ablakban megadhatjuk a kifejezést, amely legyen most a kedvenc gyümölcsünk neve. Vigyázzunk arra, hogy ha szöveget szeretnénk kiírni, akkor azt minden esetben **idézőjelek közé** kell tennünk!



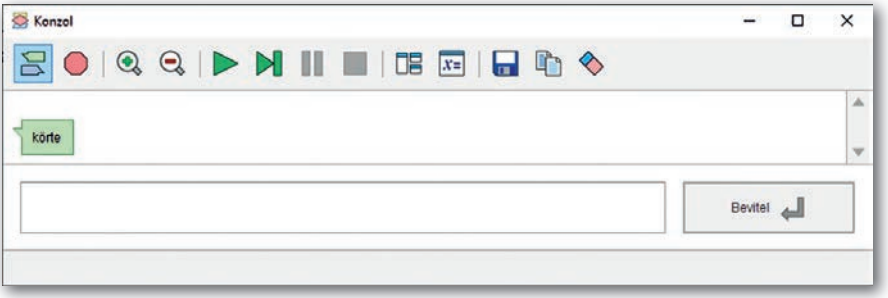
► A Kiírás szerkesztési ablaka, benne egy szöveggel

Az *OK* gomb megnyomása után megváltozik az algoritmus, most már zöld színnel jelenik meg a blokk, és láthatjuk benne a kiírt szöveget is.



Ezek után kipróbálhatjuk az algoritmust. Ehhez kattintsunk az eszköztáron a *Futtatás* (▶) ikonra!

A futtatás után a képernyőn megjelenik a *Konzol* nevű ablak, benne pedig a kiírt szöveg, illetve ha a program adatokat kér be, itt gépelhetjük be azokat is.



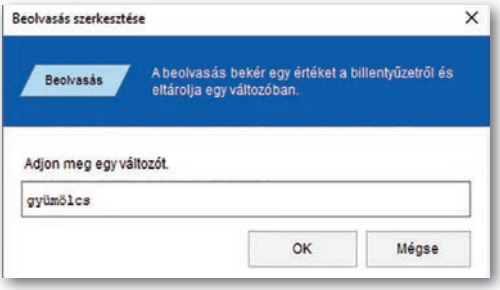
► A konzolablak, benne a kiírt szöveggel

Olvassunk be adatot változóba, és írjuk ki!

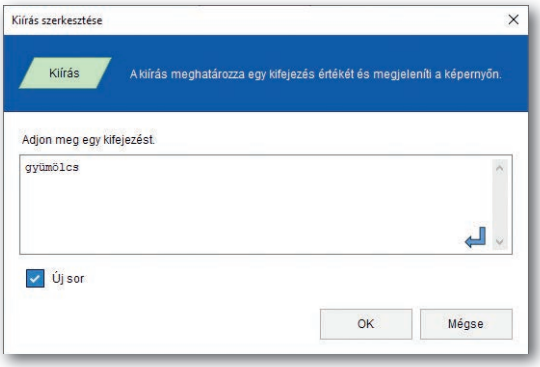
Módosítsuk úgy az algoritmust, hogy egy változóba kérjük be a kedvenc gyümölcsünk nevét, majd írjuk azt ki!

Az adat beolvasásához helyezzük el a *Beolvasás* blokkot! Kattintsunk rá duplán, és adjuk meg annak a változónak a nevét (például `gyümölcs`), amelybe a beírt szöveg kerül majd!

Módosítsuk úgy a kiírás blokkot, hogy a `gyümölcs` változó értéke kerüljön kiírásra! Ehhez adjuk meg a változó nevét! Fontos, hogy **a változónevet ne tegyük idézőjelek közé.**



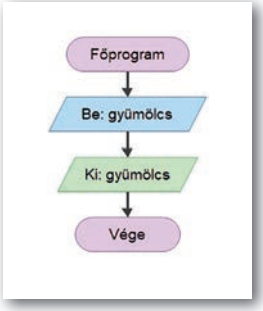
▶ Beolvasás változóba



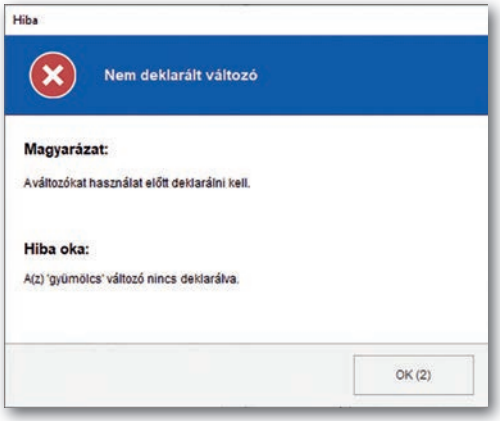
▶ Változó értékének kiíratása

Ezzel elkészült az algoritmusunk. Ne felejtsük el elmenteni a *Mentés* ( ) ikon megnyomásával!

Próbáljuk meg futtatni az algoritmust! Sajnos ekkor azt tapasztaljuk, hogy egy hibaüzenet jelenik meg.



▶ A futtatandó algoritmus



▶ A megjelenő hibaüzenet (Nem deklarált változó)

Vajon mi okozza a hibát? Mit jelent az, hogy nem deklarált változó?

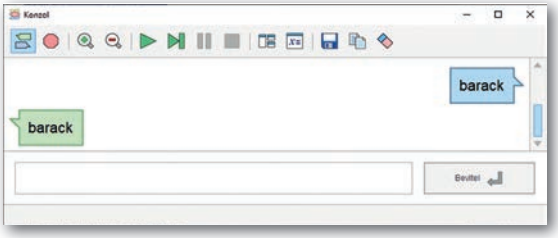
Azokban a blokkprogramozási nyelvekben, amelyekkel korábban dolgoztunk (*Scratch, MakeCode* stb.), ahogy a változóknak kezdő értéket adtunk, már használhattuk is őket. Más programozási nyelvekben viszont először **deklarálni** kell a változót, ami azt jelenti, hogy még az első használat előtt meg kell adnunk a változó nevét és típusát. A típus arra utal, hogy milyen jellegű adatot tárolunk el benne. Ez lehet például szöveg, egész szám, valós szám vagy logikai (igaz/hamis) érték.

Folytassuk tehát a változó deklarálásával! A változó értékének beolvasása előtt helyezzük el a *Deklarálás* blokkot!

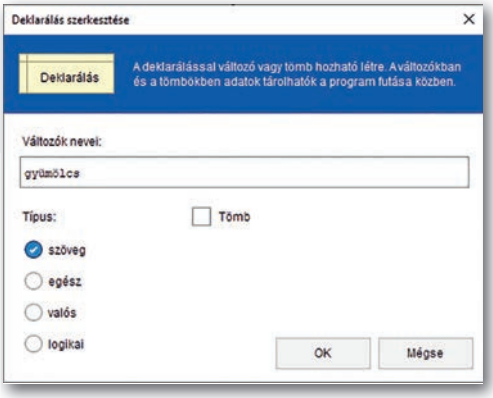
Kattintsunk rá duplán, majd adjuk meg a változó nevét (`gyümölcs`) és típusát (`szöveg`)! Vigyázzunk arra, hogy a változó nevét pontosan úgy írjuk le, mint a beolvasásnál! Ha például `gyümölcs` helyett `gyumolcs`-öt írunk, továbbra is hibaüzenetet kapunk.

Ezek után futtassuk le az algoritmust!

A konzolablakba begépelhetjük a szöveget, amely aztán a kiírás miatt újra megjelenik. Láthatjuk, hogy a beolvasott szöveg jobbra, a kiírt szöveg balra igazítottan jelenik meg, és a színük is különbözik.



▶ Beolvasás és kiírás a konzolablakban



▶ Deklarálás szerkesztése

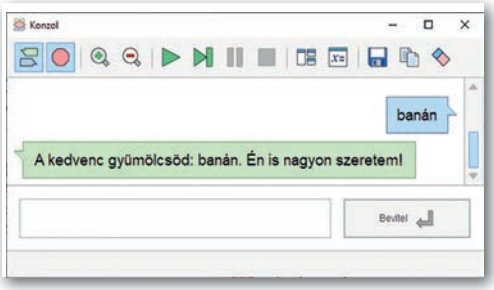
Most módosítsuk a kiírást úgy, hogy ne csak a gyümölcs nevét írjuk ki, hanem az alábbi szöveget:

„A kedvenc gyümölcsöd: *gyümölcs neve*. Én is nagyon szeretem!”

Kattintsunk duplán a *Kiírás (KI)* blokkra, és adjuk meg a kifejezést! Mivel most egymás mellett, egy sorban szeretnénk kiírni *„A kedvenc gyümölcsöd:”* szöveget, a változó tartalmát és az *„Én is nagyon szeretem!”* szöveget, ezért egy speciális jelet kell használnunk ezek összefűzéséhez. Ez nem más, mint az **&** jel. Vagyis írjuk a következőt:

```
"A kedvenc gyümölcsöd: "& gyümölcs & ". Én is nagyon szeretem!"
```

Futtatáskor a képen látható eredményt kapjuk, amennyiben a banán szót gépeljük be.

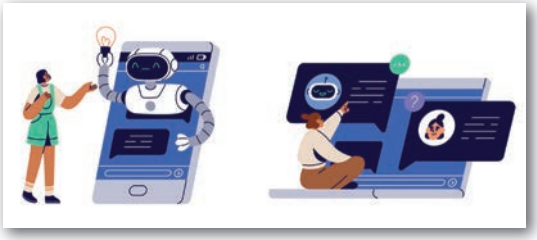


▶ A megjelenő szöveg a konzolablakban

Feladat

Pármunkában készítsünk egy chatbotot! A chatbot olyan alkalmazás, amellyel chatelni (beszélgetni) lehet. A chatbot adjon szöveges visszajelzést arra, amit írtunk (például *„Aha, értem. Miben tudok segíteni?”*). Az algoritmus legalább öt kérdés-válasz lehetőséget biztosítson! Próbáljunk olyan általános visszajelzéseket adni, amelyekkel hihető lehet, hogy a chatbot arra válaszol, amit beírtunk!

Gondolkodjunk fordítva is! Előre tudjuk, hogy mit fog válaszolni a chatbotunk, hiszen mi programoztuk be. Írjuk be olyan mondatokat, amelyekkel vicces párbeszédek születhetnek! Próbáljuk ki egymás chatbotjait úgy, hogy ne nézzük meg előtte az algoritmust!



Ciklusok és elágazások

Fej vagy írás? Használjunk ciklust és elágazást!

Pénzfeldobással egyszerűen eldönthetünk kérdéseket úgy, hogy a véletlenre bízzuk magunkat. Az eredmény kétfajta lehet: fej vagy írás. A következőkben olyan algoritmust készítünk, amely a pénzfeldobást szimulálja.

A *Flowgorithm* alkalmazásban vannak beépített függvények, amelyeket a kifejezésekben használhatunk.

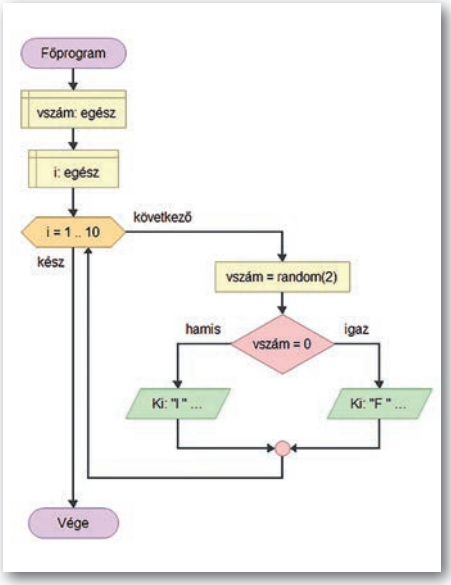
Véletlenszám generálására a `random(n)` függvény alkalmas, amely `0` és `n-1` között ad vissza véletlenszámot. Vagyis a `random(3)` a `0, 1, 2` számokat adja eredményül, véletlenszerűen.

Készítsük el az ábrán látható algoritmust! Ez tíz alkalommal írja ki véletlenszerűen az `F` és az `I` betűt, egy sorban, szóközzel elválasztva.

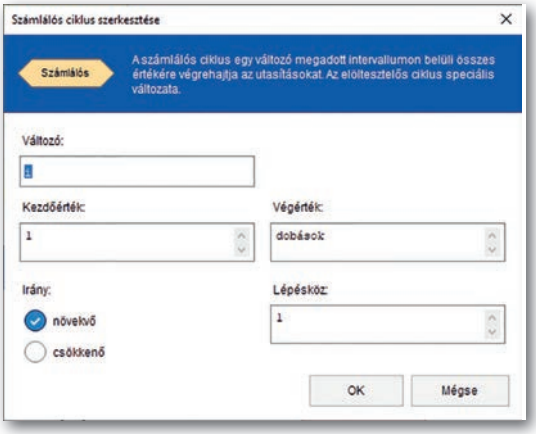
Magyarázat: Mivel tudjuk, hogy tíz alkalommal kell feldobni a „pénzérmét”, számlálós ciklust érdemes használnunk.

A `vszám` változót a véletlenszám tárolására használjuk, az `i`-t pedig a számlálós ciklus ciklusváltozójának. Mindkét változót deklarálnunk kell használat előtt, mégpedig egész típust megadva.

A számlálós ciklus szerkesztésekor meg kell adnunk a ciklusváltozó nevét (`i`), kezdő értékét (`1`) és végértékét (`10`). Ezenkívül megadhatjuk még a lépésközt, vagyis azt, hogy hányasával növekedjen a ciklusváltozó. Ez most `1` lesz. A számolás iránya szintén beállítható, ami lehet növekvő, illetve csökkenő.

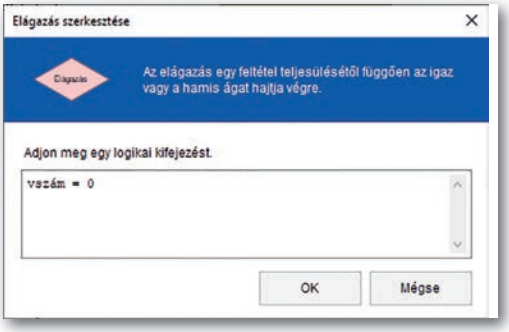


► A fej vagy írás dobás algoritmusa



► A számlálós ciklus beállítása

► Az elágazás feltételének megadása



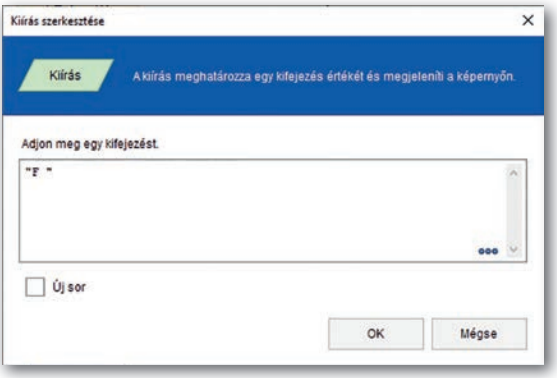
Az elágazásnál egy logikai feltételt kell megadnunk. Most azt vizsgáljuk, hogy a kapott véletlenszám nullával egyenlő-e. Ehhez a `vszám = 0` feltételt kell beírnunk.

Amennyiben a feltétel teljesül (igaz), akkor az F betűt írjuk ki, ha nem, akkor az I betűt.

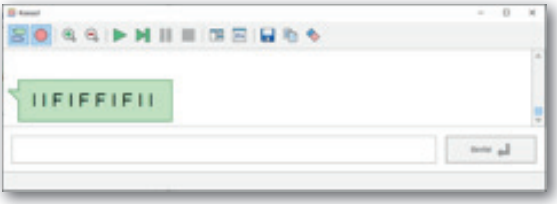
Fontos még, hogy most egy sorba, szóközzel elválasztva szeretnénk kiírni a kapott eredményt, ezért az F és I kiírásánál ne válasszuk ki az *Új sor* jelölőnégyzetet! Ekkor a kifejezés jobb alsó sarkában a ●●● ikon jelenik meg, nem pedig az ↵ ikon.

Ne felejtsük el beírni az F, illetve I betű kiírásánál a betű utáni szóközt sem!

Ezzel készen is vagyunk. Futtassuk az algoritmust! Eredményül a képen látható kimenethez hasonlót kapunk a konzolablakban.



► Kiírás sortörés nélkül

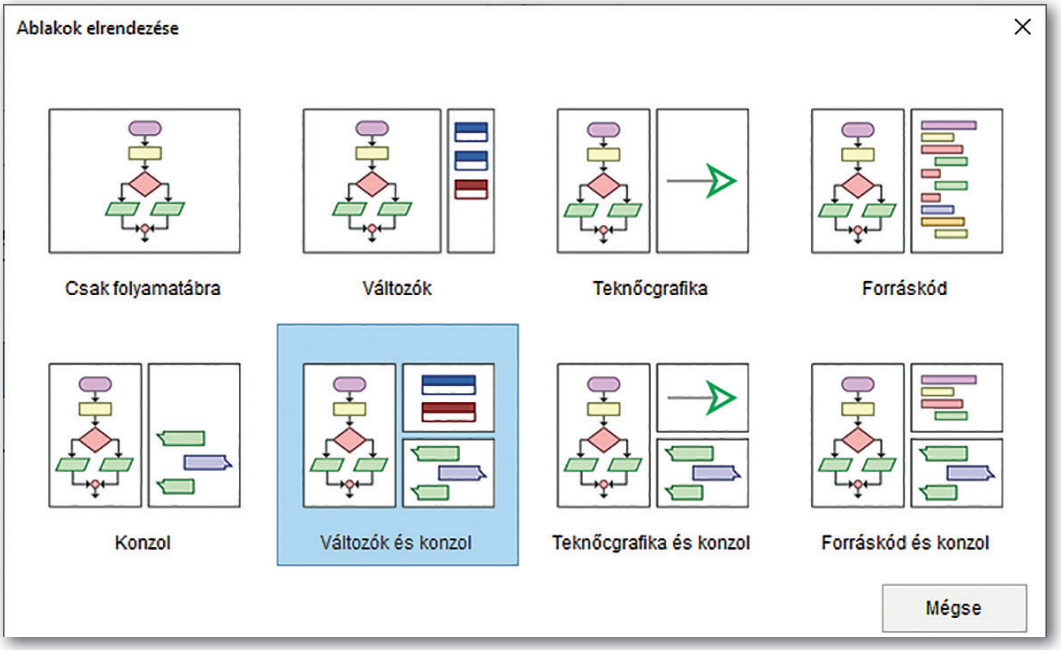


► Egy futtatás eredménye

Az algoritmus lépésenkénti futtatása

Ahhoz, hogy jobban megértsük, mi történik az algoritmus végrehajtásakor, érdemes a *lépésenkénti végrehajtást* választani, és közben ellenőrizni a változók értékét.

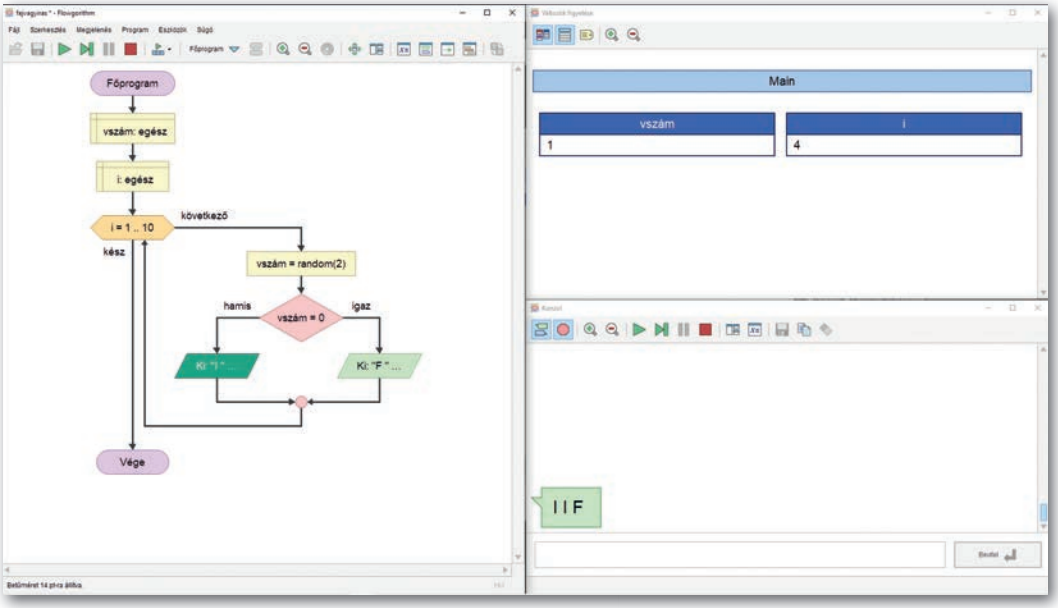
Ahhoz, hogy jól lássunk minden adatot, válasszuk ki a megfelelő ablakelrendezést! Ehhez kattintsunk az *Ablakok elrendezése* (▭) ikonra, majd válasszuk ki a *Változók és konzol* elrendezést!



► „Változók és konzol” elrendezés

Ezek után elkezdhetjük a lépésenkénti futtatást. Kattintsunk a léptetés ikonra ( )! Ezzel az algoritmust lépésenként (vagyis blokkról blokkra) tudjuk végrehajtani. A program kiemeli a következő végrehajtandó blokkot, közben pedig láthatjuk a változók értékeit is.

Az alábbi képen azt láthatjuk, hogy az `i` változó értéke 4, vagyis a negyedik betű kiírásánál tartunk. A `vszám` változó értéke `1`, vagyis az elágazás hamis ága fog végrehajtódni. Sötétzöld színnel a `KI: "I "` blokk van megjelölve, vagyis ez hajtódik végre a léptetés gomb következő megnyomásakor. A lépésenkénti végrehajtást a leállítás ( ) ikonnal állíthatjuk meg. Lépésenkénti végrehajtásból bármikor átválthatunk a normál végrehajtásba a futtatás ( ) ikon megnyomásával.



▶ Az algoritmus lépésenkénti végrehajtása

Típusalgoritmusok használata

Számoljunk! Használjunk típusalgoritmust!

Amikor a fej vagy írás játékról van szó, általában arra gondolunk, hogy a kétféle kimenet közel azonos számban fordul elő. De vajon valóban így van ez? Biztosan nem forulhat elő, hogy tízszer egymás után fejet dobunk? Győződjünk meg róla! Számoljuk meg, hányszor dobtunk fejet, illetve írást!

Az, hogy meg kell számolnunk adott tulajdonságú elemeket, nagyon gyakran fordul elő a különböző problémák megoldása során. Azt is mondhatnánk, hogy ez egy úgynevezett **típusalgoritmus,** mivel adott típusú problémára nyújt megoldást. A típusalgoritmusokat más néven **programozási tételeknek** is nevezzük. (Későbbi tanulmányaink során még több ilyennel fogunk találkozni.)

Nézzük meg először, hogy mi a típusalgoritmusa a **megszámlálásnak!** Ha összesen *N* darab elemünk van, amelyek közül meg akarjuk számolni, hogy hány adott tulajdonságú elem van, akkor az algoritmus a következő lesz:

```
Eljárás Megszámlálás
  DB:=0
  Ciklus I=1-től N-ig
    Ha az I. elem adott tulajdonságú akkor DB:=DB+1
  Ciklus vége
Eljárás vége
```

Az algoritmus lényege, hogy egy cikluson belül megvizsgáljuk az összes elemet. Ha adott tulajdonságú elemet találunk, akkor megnöveljük a darabszám (DB) értékét eggyel. Természetesen a ciklus előtt gondoskodnunk kell a darabszám változó lenullázásáról is. Ha ezt nem tennénk, akkor hibás eredményt kaphatnánk a megszámolás során.

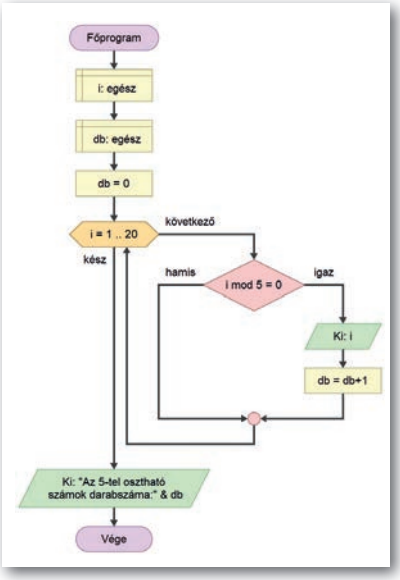
Példa: Öttel osztható számok 1 és 20 között

Nézzünk egy példát a típusalgoritmus használatára! Számoljuk meg, hogy 1 és 20 között hány olyan szám van, amely osztható öttel, és ezeket írjuk ki a képernyőre!

Tudjuk, hogy egy szám akkor osztható öttel, ha az öttel való osztás után maradékul nullát kapunk.

Az osztási maradék művelete a *Flowgorithm* programban (és sok más programozási nyelvben is) a *mod* vagy % művelet. Vagyis a 10 *mod* 5 művelet eredménye 0, mert a 10 osztható 5-tel. A 10 *mod* 3 művelet eredménye pedig 1, mert a 10-ben a 3 háromszor van meg, a maradék pedig 1. A 10 *mod* 3 műveletet így is írhatjuk: 10 % 3.

Próbáljuk ki a képen látható algoritmust a gyakorlatban is!



► Öttel osztható számok kiírásának algoritmusa

Feladat

A megszámlálás típusalgoritmusa alapján módosítsuk a korábbi (fej vagy írás) algoritmust úgy, hogy megjelenjen a fej dobások száma is! Használjuk az itt látható mintát a kiírásra!

```
F F F F F F I F I F     A fej dobások száma: 8
```

► Egy lehetséges kimenet

Legyen áttekinthető a kód! Csoportosítsuk a kódrészleteket!

Érdekes lenne tudni, hogyan alakul a fej dobások száma, ha sokszor (például százszor) megismételjük a pénzfeldobás-sorozatot. Lesz-e esetleg olyan, hogy az összes érmedobás fej lesz?

Ehhez a korábban elkészített, teljes algoritmusunkat egy olyan számlálós ciklusban kellene elhelyeznünk, amely százszor hajtódik végre. Ezzel viszont már eléggé hosszú, kevésbé átlátható algoritmust kapnánk. Mi lehet erre a jobb megoldás?

Ha egy algoritmusrészletet többször szeretnénk végrehajtani, érdemes azt a részletet egy önálló blokkban elhelyeznünk. Ezt a blokkot pedig el is nevezhetjük. Ez hasonló ahhoz, mintha különböző építőkockáink lennének, amelyekből aztán összeállíthatjuk a kész építményt, a teljes algoritmust (vagy programot). Egy fajta építőkockát akár többször is felhasználhatunk.

A különböző programozási nyelvek eltérő lehetőségeket biztosítanak *(eljárások és/vagy függvények)* a kódrészletek létrehozására. A *Flowgorithm* alkalmazásban (és sok más programozási nyelvben) **függvényeket** használhatunk erre a célra. A következő leckében ezekkel ismerkedünk meg.

Függvények használata

Mire jók a függvények?

Beépített függvényt már korábban is használtunk. Ilyen volt a `random()` függvény. A függvénynek meg kellett adnunk az *n* egész **paramétert** is a kerek zárójelen belül. A megadott paraméter határozza meg azt, hogy a függvény belsejében milyen adatokkal dolgozunk. A `random()` függvény esetén a paraméter azt jelenti, hogy mely számnál kisebb véletlenszámot szeretnénk előállítani. Vannak azonban olyan függvények is, amelyeknek több paraméterük van. Ekkor a kerek zárójelben több adatot kell felsorolni.

A függvénnyel az a célunk, hogy elvégezzen egy műveletsorozatot, vagy hogy kiszámoljon valamit. A kiszámolt értéket fogja eredményül adni a függvény hívásakor. Ezt az értéket **visszatérési értéknek** nevezzük. Ezt az értéket kapjuk eredményül a függvény hívásakor. Például a `random(3)` függvény visszatérési értéke olyan egész szám, amely `0`, `1` vagy 2 lehet.

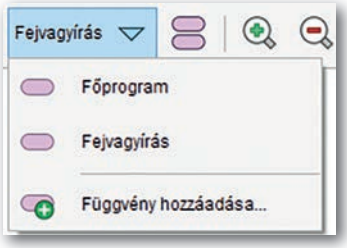
Nagyon hasznos, hogy nemcsak beépített függvényt használhatunk, hanem mi is létrehozhatunk függvényt. A függvény nevét mi adhatjuk meg, de a programozási nyelvben már foglalt nevet nem választhatunk.

Példa: A Fejvagyírás nevű függvény létrehozása

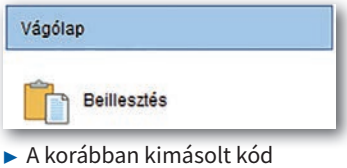
Hozzuk létre a `Fejvagyírás` nevű függvényt a következő lépésekkel!

1. Kattintsunk a függvénykezelő ( ) ikonra az eszköztáron!
2. A megjelenő ablakban kattintsunk a *Létrehozás* gombra!
3. A függvény neveként adjuk meg a `Fejvagyírás` szöveget! Visszatérési típust most nem kell beállítanunk, vagyis a *nincs* nevű választási lehetőség legyen kijelölve. Később majd megnézzük, hogy milyen előnyei lehetnek a visszatérési értéknek.
4. Kattintsunk az *OK* gombra, majd a megjelenő ablakban a *Kész* gombra!
5. Ezek után az eszköztáron már átválthatunk a *Főprogram*ra, illetve visszaválthatunk a `Fejvagyírás` nevű függvényre is.
6. A következőkben a *Főprogram*ból fogjuk a blokkokat átmozgatni a `Fejvagyírás` függvénybe. Válasszuk ki a *Főprogram*ot!
7. Egérrel kerítsük körbe az összes blokkot! Ennek hatására a kijelölt blokkok kék színűek lesznek. Amennyiben minden blokkot sikerült kijelölnünk, kattintsunk a jobb egérgombbal, és válasszuk ki a *Kivágás* műveletet! Ennek hatására a *Főprogram*ból eltűnnek a kijelölt blokkok.

▶ Függvény létrehozása



▶ Váltás a *Főprogram* és a függvények között

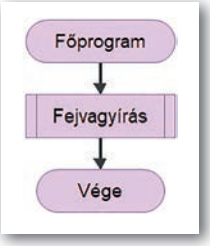


▶ A korábban kimásolt kód beillesztése

8. Váltsunk át a `Fejvagyírás` nevű függvényre! Kattintsunk a `Fejvagyírás` nevű blokkból kiinduló nyílra, majd a megjelenő ablakban válasszuk ki a *Vágólap* kategóriában a *Beillesztés* opciót! Ezzel minden korábban kivágott blokk ebbe a függvénybe kerül.

Példa: A Fejvagyírás nevű függvény meghívása

1. Váltsunk vissza a *Főprogram*ra! Kattintsunk a `Főprogram` nevű blokkból kiinduló nyílra, és a megjelenő ablakban válasszuk ki a *Hívás* utasítást! Ezt az utasítások kategória *Vezérlés* csoportjában találjuk.
2. Kattintsunk a *Hívás* blokkra, és adjuk meg a meghívni kívánt (vagyis végrehajtandó) függvény nevét: `Fejvagyírás`. Ha mindent jól csináltunk, a *Hívás* blokkban megjelenik a függvény neve.
3. Hajtsuk végre az algoritmust! Eredményül ugyanazt kell kapnunk, mint korábban, vagyis egymás után megjelennek a fej és írás dobások, illetve a fej dobások száma.



▶ A módosult *Főprogram*

Feladatok

1. Módosítsuk önállóan a főprogramot úgy, hogy a `Fejvagyírás` függvény 100 alkalommal hajtódjon végre, vagyis egymás alatt 100 alkalommal jelenjenek meg az érmedobások eredményei! Ezután futtassuk le az algoritmust!

▶ Részlet az eredményből. Látszik, hogy volt olyan eset, amikor 9 fej dobás volt

2. Vizsgáljuk meg az eredményt! Volt-e olyan eset, amikor 10 alkalommal is fejet dobtunk? Ha nem, akkor mi volt a legnagyobb szám? Beszéljük meg, hogy kinek a programja dobta a legtöbb fejet az osztályban!
3. Hallottunk-e már a cinkelt érméről? Ezt az érmét szándékosan úgy módosították, hogy sokkal többször lehessen vele fejet dobni, mint írást (vagy fordítva). Hogyan lehetne módosítani az algoritmust úgy, hogy sokkal többször legyen fej az eredmény? Beszéljük meg közösen a lehetséges megoldásokat!

Jó tudni!

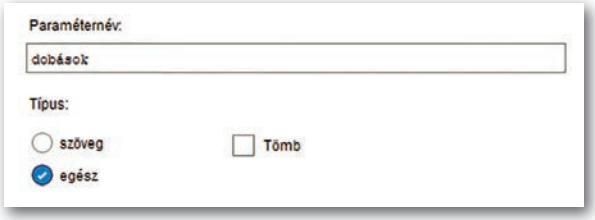
1. A legnagyobb érték meghatározására szintén létezik típusalgoritmus, amelyet *maximumkiválasztásnak* neveznek. Ezzel (és több más) algoritmussal későbbi tanulmányaink során fogunk megismerkedni. De érdemes már most elgondolkodni a problémán! Vajon milyen algoritmussal lehetne meghatározni a legnagyobb számot sok (például 100) szám közül?
2. Bizonyos programozási nyelveken nemcsak függvényt, hanem *eljárást* is lehet készíteni. Az eljárás nagyon hasonló ahhoz, ahogy a függvényt készítjük el (saját névvel rendelkezik, és paraméterei is lehetnek), azonban a függvénytől eltérően az eljárásnak nincs visszatérési értéke. A későbbiekben találkozhatunk olyan programozási nyelvekkel, amelyekben használhatunk ilyeneket.
3. A *Flowgorithm* programmal az algoritmust át lehet alakítani különböző programozási nyelvekre. Ehhez a *Forráskód megtekintése* () ikont kell megnyomni, majd a legördülő menüt használni. Sőt, mondatszerű leírássá is átalakíthatjuk a folyamatábrát, amihez a *Magyar pszeudokód* nyelvet kell kiválasztani. Próbáljuk ki!

Példa: A Fejvagyírás függvény paraméterezése

Korábban láttuk, hogy a beépített függvénynek **paraméterben** adjuk át a szükséges adatokat. Ugyanezt a saját függvények esetén is megtehetjük.

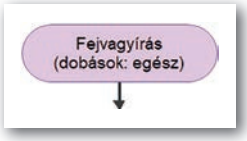
Most a `Fejvagyírás` függvényünk 10 alkalommal dob véletlenszerűen fejet vagy írást. De a 10-es szám lehetne akár a függvény paramétere is, a paraméter megváltoztatásával pedig a dobások száma is változna. Folytassuk ennek megvalósításával!

1. Nyissuk meg a függvénykezelőt (), és kattintsunk a `Fejvagyírás` függvényre!
2. Kattintsunk a *Szerkesztés* gombra!
3. A megjelenő ablakban a *Paraméterek* csoportban kattintsunk a *Hozzáadás* gombra!
4. A *Paraméternév* mezőben adjunk nevet a paraméternek! A név legyen: `dobások`. A típusa legyen egész!



► Paraméternév megadása

5. Kattintsunk az *OK* gombra (két ablakban is kell), majd a *Kész* gombra!
6. Ezek után már látszik a paraméter neve a függvénynél.



► Függvény neve a paraméterrel

7. Most válasszuk ki a `Fejvagyírás` függvényt, és módosítsuk a *számlálós ciklus*t úgy, hogy a végértéke ne 10 legyen, hanem a `dobások` változó értéke.

► Ciklus végértékének beállítása

8. Térjünk vissza a főprogramhoz! Módosítsuk a függvényhívást úgy, hogy a paramétert lehessen megadni! Vagyis írjuk például ezt: `Fejvagyírás(15)`
9. Futtatáskor azt tapasztaljuk, hogy most már 15 alkalommal történik érmedobás.

Függvény használata visszatérési értékkel

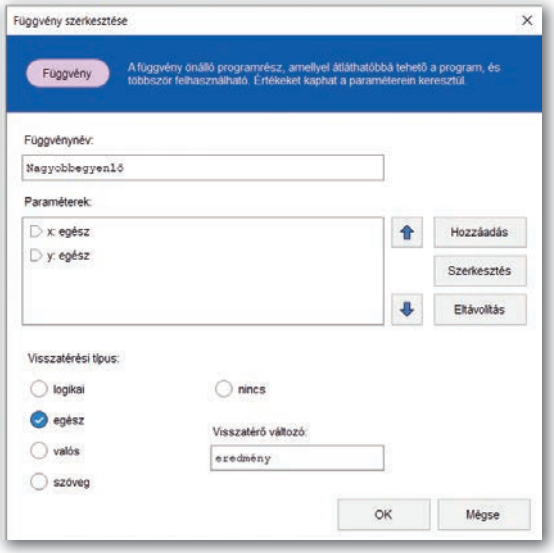
Most nézzük meg azt, hogy hogyan állíthatunk be a függvénynek visszatérési értéket! Ezt a tudásunkat több blokkprogramozási nyelvben felhasználhatjuk, például a micro:bitek programozásánál is.

Készítsünk egy `Nagyobbegyenlő` nevű függvényt, amely két egész paraméterrel rendelkezik! Ha a paraméterként kapott két szám különböző, a függvény a nagyobbat adja vissza, ha viszont egyformák a paraméterek, a függvény ezt a számot adja vissza. Vagyis a függvény a táblázatban láthatók szerint működjön.

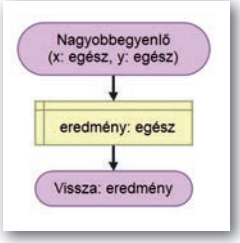
Függvény	Eredmény
`Nagyobbegyenlő(2,8)`	8
`Nagyobbegyenlő(-1,-6)`	-1
`Nagyobbegyenlő(5,5)`	5

A létrehozás lépései:

1. Hozzuk létre a `Nagyobbegyenlő` függvényt, és adjunk hozzá két paramétert! Az *x* és *y* paraméter lesz a két számunk, amelyek legyenek *egész* típusúak!
2. A függvény visszatérési értékének típusa legyen egész! Meg kell adnunk a visszatérő változó nevét is, amely legyen: `eredmény`.
3. Az *OK*, majd a *Kész* gombok megnyomása után megjelenik a függvény folyamatábrája. Láthatjuk, hogy a függvény a *Vége* blokk helyett a *Vissza* blokkal zárul. Ez utal arra, hogy a függvénynek van visszatérési értéke.

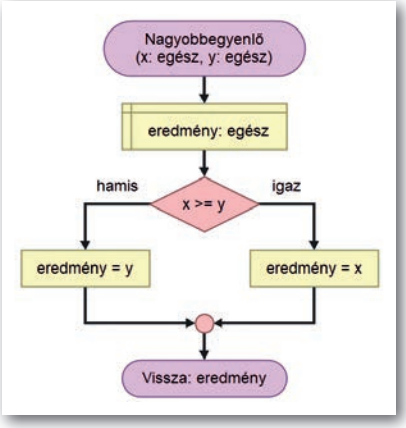


► A függvény szerkesztési ablaka



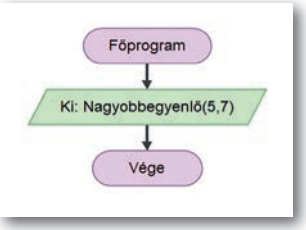
► A létrejött függvény

4. Módosítsuk úgy a függvényt, hogy az `eredmény` változó az *x* és *y* számok közül a nagyobbal legyen egyenlő! Ha az *x* és az *y* megegyezik, akkor az *x* szám legyen az eredmény! Ehhez használjunk elágazást!



► Egy lehetséges megoldás a függvényre

5. Készítsük el a *Főprogram*ot is! Írassuk ki a `Nagyobb-egyenlő` függvény eredményét!
6. Teszteljük az algoritmust különböző paraméterek megadásával!



► A *Főprogram*

Kérdés

Az elágazásnál most az x >= y feltételt helyeztük el. Mi történne, ha az x > y feltételt írnánk be? Változna-e az eredmény? Miért?

Jó tudni!

A fenti algoritmus végrehajtása során három eset állhat elő. Vagy az első szám a nagyobb, vagy a második, vagy a két szám egyenlő. A tesztelésnél mindig törekednünk kell arra, hogy a lehetséges eseteket megfelelő paraméterekkel kipróbáljuk.

Ha hibát találunk, használjuk a lépésenkénti végrehajtást, amely segít abban, hogy a változók értékét figyelemmel kísérjük, illetve lássuk, hogy milyen utasítások hajtódnak végre, és hányszor.

Feladatok

1. Próbáljuk ki, hogy mi történik akkor, ha elhelyezünk az algoritmusban egy vagy több töréspontot! Milyen hatása van a töréspontnak a futtatáskor, illetve a lépésenkénti végrehajtáskor?
2. Lássuk el az algoritmust magyarázó megjegyzésekkel! Ezek a magyarázatok segíthetnek nekünk abban, hogy később is megértsük, mit miért csináltunk. Később látni fogjuk, hogy megjegyzéseket a programok forráskódjában is elhelyezhetünk.
3. Pármunkában készítsünk algoritmusokat a következő feladatok megoldására!
   a. Készítsünk egy sormintát, amely legfeljebb 10 karakterből (betű, szám, egyéb karakterek) áll. Például: >-----<. Kérjük be a felhasználótól, hogy hányszor szeretné megjeleníteni egymás mellett a sormintát, majd jelenítsük meg azt!
   b. Fejlesszük tovább az algoritmust úgy, hogy több sorból álló mintát is ki lehessen rajzolni, és a felhasználó adhassa meg a sorok számát! Az egy sorban megjelenő sorminta kiírását paraméterezett függvény segítségével készítsük el!

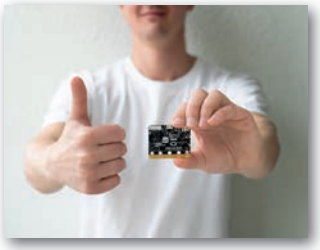
Korábbi tanulmányainkban a programok készítésekor számos műveletet használtunk (például `és`, `vagy`, `nem`). A *Flowgorithm* programban is használhatók műveletek. A következő feladatok megoldása során ezekre szükség lehet. Összetettebb műveletek esetén a kerek zárójelekkel csoportosíthatjuk a műveleteket.

	Művelet	Példa
egyenlőség vizsgálata	= *vagy* ==	x = 10 x == 10
nem egyenlőség vizsgálata	!= *vagy* <>	x != 10 x <> 10
és művelet	&& *vagy* and	x > 5 && x < 10 x > 5 and x < 10
vagy művelet	|| *vagy* or	x = 5 || x = 10 x = 5 or x = 10
tagadás művelet	! *vagy* not	! (x < 10)
összefűzés művelet	&	"Az életkorom: " & kor
maradék számítása	% *vagy* mod	x % 5 x mod 5

4. Jelenítsük meg a naptár egy hónapját a következő formátumban: a napok 1-től 31-ig jelenjenek meg, és a sorszámok után legyen kiírva a *nap:* szöveg is. A héttel osztható számok esetén a *Szabadnap* szöveg jelenjen meg, a többinél a *Munkanap*. Vajon hogyan lehetne megoldani azt, hogy a héttel osztható szám előtti napnál is a *Szabadnap* szöveg jelenjen meg? Készítsük el ezt a változatot is!

5. Készítsük el az `Ellentett` függvényt! A függvénynek egy paramétere legyen, amely egy egész szám! A függvény adja vissza a szám ellentettjét, vagyis a –1-gyel való szorzatát! Teszteljük le az eredményt pozitív, negatív és nulla paraméterre is!

Programozzunk micro:bitet!



Az előzőekben megismertünk egy olyan eszközt, amellyel algoritmusokat készíthetünk, kipróbálhatjuk őket, és akár lépésenként is nyomon követhetjük, hogy mi történik a végrehajtásuk során.

Ez a tudás a későbbiekben is hasznos lesz, amikor új algoritmusokat kell kitalálnunk. Az is előfordulhat, hogy mások által írt algoritmust kell megértenünk, alkalmaznunk, és tesztelnünk a helyes működését.

Használtunk függvényt, amellyel csoportosíthattuk a többször felhasználandó kódot, illetve paramétereket adtunk a függvénynek. A következőkben visszatérünk a micro:bitek programozásához, melynek során kihasználjuk a függvényekben rejlő lehetőségeket is.

Használjunk függvényeket!

A játékprogramok gyakran visszaszámlálással indulnak, hogy a játékos felkészülhessen a kezdésre. Készítsünk olyan függvényt, amely elvégzi a visszaszámlálást! A függvény paramétere legyen az az egész szám, amelytől indítani szeretnénk a visszaszámlálást egészen nulláig!

Készítsünk egy új projektet a https://makecode.microbit.org/ felületen! A függvényeket a *Haladó funkciók* között találjuk, ezért kattintsunk a Haladó legördülő menüre!

A *f(x)* Függvények kategóriában kattintsunk a Függvény létrehozása... gombra!

A *Függvény szerkesztése* ablakban adjunk nevet a függvények (`Visszaszámol`), majd kattintsunk a *Paraméter hozzáadása* sorban a *Szám* paraméterre! Ha készen vagyunk, nyomjuk meg a *Kész* gombot!

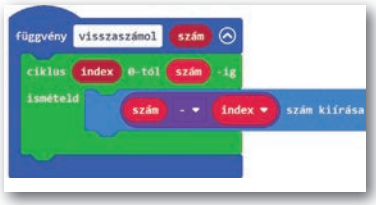
► A függvény beállításai

A Visszaszámol függvény tartalmának megadása

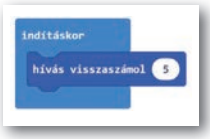
Ezzel létrejött a függvény! Helyezzük el a visszaszámláláshoz szükséges utasításokat! Például használhatunk számlálós ciklust erre a célra. A paraméterként megjelenő szám változót úgy tudjuk elhelyezni ciklusváltozóként, hogy az egérrel megragadjuk, és a megfelelő helyre vonszoljuk.

Ügyeljünk arra, hogy vissza kell számolnunk, ezért a kiírásnál nem a ciklusváltozót kell kiíratnunk, hanem megfelelő matematikai művelettel még elő kell állítanunk a kiírandó számot. Korábbi tanulmányainkban volt már erre példa.

Most már csak gondoskodnunk kell a függvény meghívásáról, és tesztelhetjük is az eredményt.



► A visszaszámlálás egy lehetséges megvalósítása



► A függvény meghívása

Feladatok

1. Egy szöveg a micro:bit kijelzőjén jobbról balra gördítve jelenik meg. Pármunkában készítsünk olyan függvényt, amely a szöveget betűnként jeleníti meg, gördítés nélkül! A függvény neve legyen: `Betűzd`! Minden betű kiírása után teljen el egytized másodperc, majd a kijelzőt letörölve, az újabb betű megjelenése előtt teljen el újabb egytized másodperc! A szöveg feldolgozásával kapcsolatos blokkokat a Szöveg kategóriában találjuk. Gondoljuk át, mely blokkok szükségesek a feladat megoldásához! A függvényt természetesen úgy kell elkészítenünk, hogy tetszőleges hosszúságú szöveget képes legyen betűzni.
2. Módosítsuk a fenti függvényt úgy, hogy ne csak a szöveg legyen paraméterként átadva, hanem az időzítési érték is, amely az előző feladatban egytized másodperc volt!

Függvény visszatérési értékének beállítása

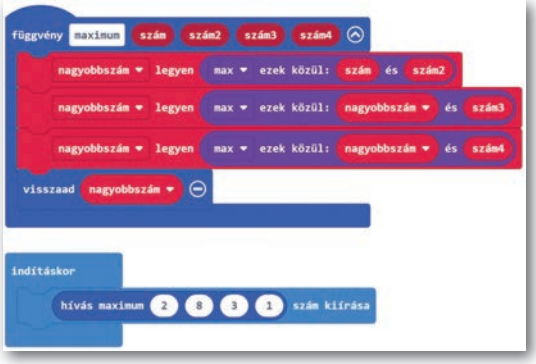
Az imént létrehoztunk függvényeket, de azoknak nem volt visszatérési értéke. Készítsünk egy paraméterezhetőt is, hiszen a későbbiekben azt sokféle célra felhasználhatjuk.

A *MakeCode* blokkprogramozási felület Matek blokkjában találunk egy beépített függvényt (max ezek közül: 0 és 0), amely két különböző szám közül a nagyobbat adja vissza. Egyenlő számok esetén az értéke a megadott szám lesz. Készítsünk olyan függvényt, amely hasonlóan működik, de nem két szám közül adja meg a legnagyobbat, hanem négy szám közül!

Hozzunk létre `maximum` néven egy függvényt, amelynek legyen négy darab szám paramétere! Ha négy számból akarjuk meghatározni a legnagyobbat, a következőképpen tehetjük meg. Először határozzuk meg, hogy az első két szám közül melyik a nagyobb! Hasonlítsuk össze ezt a számot a harmadik számmal, és a nagyobbat tároljuk el egy változóban! Végül ezt a számot hasonlítsuk össze a negyedik számmal, és ismét tároljuk el a nagyobbat!

Ahhoz, hogy a függvény visszaadja a legnagyobb értéket, a *Függvények* kategóriából a *Visszaad* blokkot kell elhelyeznünk a függvény végén. Látható is, hogy a *Visszaad* blokknak olyan a kialakítása, hogy alá már nem helyezhetünk el másik blokkot.

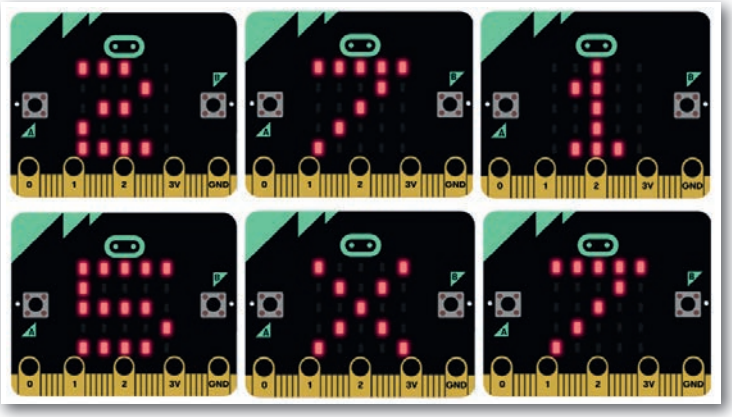
Ezek után próbáljuk ki a függvény működését! Írassuk ki a legnagyobb számot! A *Kiírás* blokkban lekerekített blokkot tudunk csak elhelyezni. Mivel olyan függvényt készítettünk, amelynek van visszatérési értéke, a *Függvények* kategóriában megjelenik a függvényhívás olyan változata is, amely egy lekerekített blokkban helyezkedik el. Használjuk ezt a szám kiírásánál!



► A feladat egy lehetséges megoldása

Feladatok

1. Korábban elkészítettük a fej vagy írás dobás algoritmusát. Valósítsuk meg ugyanezt a micro:biten is! A függvény neve legyen: `Fejvagyírás`. A függvénynek egy paramétere legyen, amely jelentse azt, hogy hányszor dobunk egymás után! Fej és írás esetén más-más ikonok jelenjenek meg a kijelzőn! Számoljuk meg a függvényben azt is, hogy hányszor dobtunk fejet! A fejek száma legyen a függvény visszatérési értéke! A micro:bit megrázásával hívjuk meg 10-es paraméterrel a függvényt, majd utána jelenjen meg a fej dobások száma!
2. Készítsünk függvényt `Kockadobás` néven! A függvény paramétere jelentse azt, hogy hány oldalú dobókockával dobunk. Egy hagyományos, hatoldalú kocka esetén a paraméter értéke 6. Ha a paraméter 12, akkor már 1 és 12 közötti értékeket kaphatunk. A dobott szám legyen a függvény visszatérési értéke!
   Végezzük el négyszer a kockadobást úgy, hogy jelenjenek meg a micro:bit kijelzőjén a dobott számok! Ezt követően töröljük le a kijelzőt, majd jelenítsük meg az X ikont! Ezután írjuk ki, hogy melyik szám volt a legnagyobb a négy dobás közül! Használjuk fel a korábban készített `Maximum` függvényt!



► A program egy lehetséges eredménye

Készítsünk játékokat micro:bitre!

Az alábbiakban ötleteket találunk arra, hogy milyen játékok, alkalmazások készíthetők önállóan, párban vagy akár csoportmunkában a micro:bit eszközre. Törekedjünk arra, hogy a kód legyen átlátható, ahol lehet, csoportosítsuk a kódot kisebb egységekbe a függvények segítségével! Adjunk megjegyzéseket a kódhoz, hogy társaink is megértsék a gondolatmenetünket! Ezt a jobb gomb megnyomása után, a *Megjegyzés hozzáadása* menüponttal tehetjük meg.

Milyen az időérzékünk?

1. Pármunkában készítsünk olyan programot, amely azt méri, hogy milyen jó az időérzékünk! A játék az *A* gomb megnyomásával induljon úgy, hogy jelenjen meg a kijelzőn egy véletlenszám 5 és 9 között, majd 1 másodperc múlva jelenjen meg egy ikon, amely a játék kezdetét jelzi. A játék lényege, hogy a *B* gombot pontosan annyi másodperc után kell megnyomni, amekkora számot láttunk korábban! A gomb megnyomása után jelenítsük meg a kijelzőn azt, hogy hány ezredmásodperc különbséggel nyomtuk meg a gombot az előírt értékhez képest! Természetesen minél kisebb számot látunk, annál ügyesebbek voltunk. Azt, hogy a program mennyi ideje fut, a *Bemenet* kategória futási idő (ms) blokkjával kérhetjük le.
2. Fejlesszük tovább a programot úgy, hogy ne csak egy szám jelenjen meg a játék végén, hanem egy ikon is, amely utal az eredményre! Legyen három kategória: nagyon pontos, pontos és átlagos. Például akkor nagyon pontos az eredmény, ha a tényleges idő és a lenyomás között kevesebb mint fél másodperc telt el.
3. Készítsük el a játék azon változatát, amikor két micro:bit egymásnak küldi az adatokat! Az első micro:biten lehessen a játékot elindítani és leállítani, de az eredmény a másik micro:biten jelenjen meg!

Vajon tudunk-e blöffölni?

A játék közbeni blöffölés azt jelenti, hogy félrevezetjük a társainkat azért, hogy ne essünk ki a játékból, vagy hogy jobb eredményt érjünk el.

A bás nevű játék eredeti változatát dobókockákkal játsszák. A játék előnye, hogy akár 5-6 fős (vagy nagyobb) csoportokban is játszhatjuk. A játékban minden játékos két dobókockával dob úgy, hogy társai ne láthassák a dobott számokat, vagyis a dobások eredményét el kell takarni. A játék célja, hogy társaink elhiggyék azt, hogy a két dobókockával olyan számot dobtunk, amely nagyobb vagy egyenlő, mint amit az előzőleg dobó társunk dobott.

A két dobott számból speciális szabállyal határozzuk meg a dobás értékét. A nagyobb dobás lesz a kétjegyű számunk első számjegye, a kisebb szám pedig az utolsó számjegye.

Ha két egyenlő számot dobtunk, akkor természetesen a két számjegy megegyezik. Ezt nevezik básnak. Az 1-es bás a 11, a 2-es bás a 22, és így tovább.

Vagyis a dobásértékek a következőképpen alakulnak:

Egyik kockadobás	Másik kockadobás	A dobás értéke
1	3	31
4	1	41
3	3	33

► A két kockadobás eredménye és az abból előállt dobási érték

Van azonban egy nehezítés is, amely miatt nagyon kell figyelni a játék során. A dupla dobások nagyobb értékkel bírnak, mint a különböző számjegyekből állók. De ezeknél is többet ér a 21-es dobás, amely a játékban a legtöbbet éri. Vagyis a dobásértékek sorrendje aszerint, hogy melyik érték ér többet az előzőnél:

31, 32, 41, 42, 43, 51, 52, 53, 54, 61, 62, 63, 64, 65, 11, 22, 33, 44, 55, 66, 21

Kezdetben minden játékosnak 5 pontja (vagy zsetonja) van. A játékosok körben ülnek egy asztal mellett vagy a földön. Minden körben más játékos kezdi a játékot az óramutató járásával megegyezően vagy ellenkezően (amelyik jobban tetszik). A játékot kezdő dobó elmondja a dobásának az értékét. A soron következő játékos ezt vagy elhiszi, vagy nem. Ha elhiszi, akkor folytatódik a játék: dob, és elmondja, mennyit dobott, vagy blöfföl, és egy másik számot mond. Ha viszont nem hitt az előző játékosnak, akkor annak fel kell fednie a dobott kockáit. Ekkor kiderül, hogy igazat mondott-e, vagy blöffölt.

Ha valaki rosszul blöffölt, az veszít egy pontot, aki pedig hazugságon érte társát, az 1 ponttal gazdagodik.

Akkor is veszít a játékos 1 pontot, ha olyan számértéket mondott, amely nem fordulhat elő a játékszabály szerint (5, 9, 40, 13, 24, 46 stb.). Ekkor a dobás sorrendjében következő játékos automatikusan kap 1 pontot.

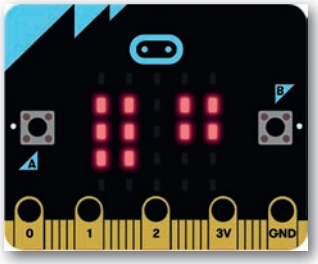
Az új fordulót a következő játékos kezdi. Akinek elfogynak a pontjai, kiesik a játékból. A játék addig tart, míg egy játékos marad, de játszhatunk úgy is, hogy ha valaki elér egy bizonyos pontszámot (például 10), akkor vége a játéknak.

Nézzünk egy példajátékot! Az első játékos 3-at és 4-et dob, ezért a dobás értéke 43. Nem blöfföl, az igazat mondja a társainak. A második játékos 1-et és 3-at dob. Dobásának értéke 31, ami sajnos kevesebbet ér, mint az előző játékosé. Ezért kénytelen blöffölni, és azt mondja, hogy 52-t dobott. A következő játékos ezt nem hiszi el, és kéri, hogy mutassa meg a dobott kockákat. Mivel a játékos hazudott a pontértékről, elveszít 1 pontot, a másik játékos pedig kap 1 pontot.

A bás játék elkészítése

Feladatunk az előző társasjáték algoritmusának elkészítése a *Flowgorithm* alkalmazásban, majd a megvalósítása micro:bit segítségével.

A kockadobás a micro:bit megrázásával történjen! A dobás értékét úgy jelenítsük meg a kijelzőn, hogy egyszerűen leolvasható legyen a két szám! Például a nagyobb dobást megjeleníthetjük a kijelző bal oldalán, a másikat pedig a jobb oldalán. Így a dobásérték egyszerűen leolvashatóvá válik. A kockák képének megjelenítéséhez használjunk függvényeket, a megjelenítendő számot paraméterként adjuk át!



► A két dobás egyidejű megjelenítése (64-es pontérték)

Nyerés esetén a *B* gomb megnyomásával lehessen növelni a pontszámot, vesztés esetén pedig az *A* gombbal csökkenteni. Ha a pontszám eléri a nullát, jelenjen meg egy animáció, illetve halljunk egy hangjelzést, amely arra utal, hogy kiesett a játékos a további fordulókból! Az *A* + *B* gombok megnyomása után jelenjen meg a pontszám, majd kis idő elteltével az utolsó dobás képe. Így akár játék közben is ellenőrizni lehet a pontszámot.

Továbbfejlesztés, ötletelés

3-4 fős csoportokban ötleteljünk arról, hogyan lehetne növelni a játékélményt! Gondolkodhatunk abban is, hogy a micro:bitek rádiókapcsolattal kommunikálhatnak egymással vagy egy játékvezetői micro:bittel. Színesíthetjük a játékot különböző hanghatásokkal is.

Valósítsuk meg ötleteinket! Próbáljuk ki egymás megoldásait, és játsszunk néhány játszmát a legjobban sikerült programokkal!

Ezek után akár osztálykirándulásra is magunkkal vihetjük a micro:biteket, és a szabadidőben is játszhatunk velük.

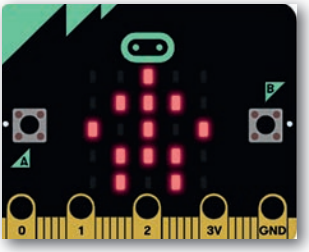
Ha maradt még időnk, fejlesszünk más társasjátékokat is a korábban tanultak felhasználásával!

Összefoglalás

Az általános iskola felső tagozatán sok izgalmas lehetőséggel találkoztunk az algoritmizálás, programozás és robotika területén.

Olyan blokkprogramozási környezeteket használtunk, amelyekben játékosan készíthettük el programjainkat. Ehhez algoritmusokat gyártottunk. Az algoritmusokat mondatszerűen vagy folyamatábrák segítségével írtuk le, sőt azt is megtanultuk, hogyan tesztelhetjük az algoritmusok működését, és milyen lehetőségeink vannak a hibák azonosítására és kijavítására. Megismerkedtünk olyan programozási fogalmakkal, amelyekre a későbbiekben építeni tudunk.

Valós és nem létező (szimulált) robotokat irányítottunk, és megtanítottuk őket különböző feladatok elvégzésére, például vonal mentén történő haladásra. Áttekintettük és kipróbáltuk, hogy a robotok milyen állapotokkal rendelkeznek, illetve hogyan képesek érzékelni a környezetüket.



A programozás során megismerkedtünk számos utasítással, függvénnyel és vezérlési szerkezettel, használtunk ciklusokat, elágazásokat, hogy minél többfajta feladat megoldására képesek legyünk.

Megismerkedtünk a micro:bit vezérlővel, amely játékossá tette a tanulást. Nemcsak az előírt feladatokat készítettük el, hanem pármunkában és csoportmunkában is megvalósíthattuk saját ötleteinket, amelyeket aztán megoszthattunk a többiekkel.

A környezetünkből származó adatokat (például fényerősséget) mértünk és dolgoztunk fel úgy, hogy rádiókapcsolatot is használtunk az eszközök között az egyes feladatok elkülönítésére.

Animációkat és ezekre épülő játékokat fejlesztettünk úgy, hogy a szereplőket különböző gesztusokkal vagy gombnyomássokkal lehetett irányítani. Rajzoltunk a kijelzőre, és valós, életbeli problémákat szimuláltunk.

Megnéztük, hogyan tudjuk hatékonyabbá tenni a programjainkat, illetve milyen lehetőség van a kód újrafelhasználására és átláthatóbbá tételére.

Későbbi tanulmányainkban már nem (feltétlenül) blokkprogramozási környezetet fogunk használni, hanem például betekintést kapunk a modern, objektumorientált programozásba is. De azt, amit eddig megtanultunk, a későbbiekben is tudjuk használni. Addig is, ha találkozunk egy problémával, amelyet meg tudunk oldani az általunk ismert környezetekben, álljunk neki bátran! Ezzel nemcsak a számítógépes gondolkodásunk fejlődik, hanem hasznos alkalmazásokat hozunk létre, amelyek gyorsabbá, hatékonyabbá, esetleg élvezetesebbé teszik a munkát. Ehhez kívánunk sok sikert a továbbiakban!