

EXCEL MAKRÓ TANFOLYAM

2.1 verzió

2018

Készítette:

Zugonicsné Szabó Dóra

NAV KEKI

Lektorálta:

Smolong Edvin

NAV INIT

Lezárva:

Budapest, 2013. február 7.

TARTALOMJEGYZÉK

1. BEVEZETŐ.....	5
2. AZ EXCEL MAKRÓK MEGÍRÁSÁVAL ÖSSZEFÜGGŐ FOGALMAK.....	6
2.1. MAKRÓ	6
2.2. PROGRAMOZÁS, PROGRAM	6
2.3. MODULÁRIS PROGRAMOZÁS	6
2.4. ALGORITMUS.....	6
2.5. FORDÍTÓPROGRAM	7
2.6. SZINTAXIS, SZINTAKTIKAI HIBA	7
2.7. SZEMANTIKAI HIBA.....	7
3. AZ EXCEL VB NÉZETÉNEK MEGISMERÉSE.....	8
3.1. ESZKÖZTÁR.....	9
3.2. PROJEKTTALLÓZÓ ABLAK	10
3.3. TULAJDONSÁGOK ABLAK.....	10
4. MAKRÓ KÉSZÍTÉSE FELVÉTELLEL (VARÁZSLÓVAL).....	11
5. ÖNÁLLÓ EXCEL MAKRÓK MEGÍRÁSÁHOZ SZÜKSÉGES PROGRAMOZÁSI ISMERETEK.....	14
5.1. A PROGRAMFEJLESZTÉS LÉPÉSEI	14
5.1.1. Feladat-specifikáció	14
5.1.2. Algoritmus készítés	14
5.1.3. Kódolás	14
5.1.4. Tesztelés, dokumentáció.....	15
5.2. A STRUKTURÁLT PROGRAM RÉSZEI	15
5.2.1. Rutinfej	15
5.2.2. Deklarációs rész	15
5.2.2.1. Konstansok	16
5.2.2.2. Típusok	16
5.2.2.3. Változók	16
5.2.2.4. Saját, felhasználó által létrehozott típusok.....	20
5.2.2.5. Típuskonverziók.....	21
5.2.3. Törzs.....	23
5.2.3.1. Szekvencia	23
5.2.3.2. Szelekció (Elágazás)	24
5.2.3.3. Iteráció (Ciklus)	25
5.2.3.4. Eljárások, függvények	29
5.2.3.5. Rutinhívó utasítások	32
5.2.3.6. Beépített Excel függvények.....	34
5.2.3.7. Üzenetküldés és válaszkérés VBA nyelven.....	37
5.3. GYAKORLÓ FELADAT	39
5.4. A PROGRAMOZOTT MAKRÓK INDÍTÁSA	39
5.5. ÖSSZETETT PÉLDA A FEJEZETBEN TANULTAK ELSAJÁTÍTÁSÁRA	41

6. OBJEKTUMOK VISUAL BASIC-BEN (VEZÉRLŐELEMELK, FORMOK HASZNÁLATA).....	42
6.1. AZ OBJEKTUMORIENTÁLT PROGRAMOZÁS ALAPJAI	42
6.2. FORM MODULE	43
6.2.1. Vezérlőelemek.....	43
6.2.2. Vezérlőelemek elhelyezése a párbeszédapon	45
6.2.3. Vezérlőelemek részletes ismertetése	45
6.2.3.1. Beviteli mező (TextBox)	45
6.2.3.2. Címke, Felirat (Label)	46
6.2.3.3. Parancsgomb (CommandButton)	46
6.2.3.4. Listamező (ListBox)	46
6.2.3.5. Kombinált Lista (ComboBox)	47
6.2.3.6. Kijelölőnégyzet (CheckBox).....	47
6.2.3.7. Választógomb (OptionButton).....	48
6.2.3.8. Kép (Image).....	48
6.2.3.9. Görgetősáv (ScrollBar)	48
6.2.3.10. Léptetőgomb (SpinButton)	49
6.2.4. Párbeszédapok.....	49
6.2.4.1. Párbeszédap létrehozása	49
6.2.4.2. Párbeszédap tulajdonságai	50
6.2.4.3. A Form objektum metódusai	50
6.2.4.4. Form megjelenítése, elrejtése	51
6.2.4.5. Felhasználói beavatkozások a formon	52
6.3. OSZTÁLY MODUL (CLASS MODULE).....	53
7. ÁLLOMÁNYOK HASZNÁLATA, ESEMÉNYVEZÉRLÉS	60
7.1. ÁLLOMÁNYKEZELÉS	60
7.2. ESEMÉNYVEZÉRLÉS.....	61
7.2.1. Munkafüzet események.....	62
7.2.2. Munkalap események.....	62
7.2.3. Párbeszédap események.....	63
7.2.4. Párbeszédap vezérlőelemeihez kötött események.....	63
7.2.5. Eseményvezérelt metódusok	63
8. ÖSSZETETT PÉLDA FELADAT MEGOLDÁSA	65
1. FELADAT:	65
2. FELADAT:	69
3. FELADAT:	69
4. FELADAT:	70
5. FELADAT:	70
6. FELADAT:	71

1. BEVEZETŐ

Ezt a példatárat azon felhasználók számára készítettem, akik az Excelt már olyan magas szinten elsajátították, hogy rájöttek, nem egyszerűen cellák formázására, összeadására alkalmas csupán ez a szoftver. Vannak ötleteik, amelyeket meg szeretnének valósítani, de nem tudják, hogyan álljanak neki. Az Excel magasabb szintű használata azt jelenti, hogy bizonyos feladatokat automatizálni szeretnénk. Ezekhez az automatizálási feladatokhoz hozták létre a fejlesztők a Visual Basic for Application (VBA) nyelvet – ami minden Office termékben megtalálható – és a Visual Basicre támaszkodó makrógenerátort. A makrók megírásához tehát Visual Basic (VB) nyelvet használhatunk, amely valójában az objektum orientált Basic nyelv kiegészítése grafikus felületekkel. A VB nyelvű utasításokat egy makró-szerkesztő (Visual Basic Editor) segítségével „fogalmazhatjuk meg”, amely a saját igényeiknek megfelelő makrók megírását, hibakezelését, és nyomon követését segíti.

Ez a tankönyv tehát arra hivatott, hogy ezen VBA nyelv alapjait megismertesse az olvasóval. Azt azonban hangsúlyozni szeretném, hogy a téma nehézsége és komplexitása miatt a tankönyv terjedelme *nem elegendő* ahhoz, hogy a makró programozás *minden* fortélyát bemutassa, továbbá a programozói ismeretek elsajátítása is *hosszas gyakorlást* igényel. Arra azonban megfelelő, hogy elindítsa a felhasználó gondolatait és azokat az alapelveket bemutassa, amelyeket összeépítve később önállóan alkalmas lesz bonyolultabb makrók elkészítésére is.

A makró programozáshoz elengedhetetlen, hogy a felhasználónak legyen „programozói vénája”, továbbá, mivel a Visual Basic Editor angol nyelven íródott, szükséges az angol nyelv minimális ismerete is.

JELMAGYARÁZAT

A tankönyv tartalmaz néhány, a programozásban is használatos jelet, melyeket az érthetőség kedvéért megmagyarázok:

[elhagyható]

A szögletes zárójelek közti rész elhagyható

<név>

A kacsacsőrökben levő részt, itt a nevet be kell helyettesíteni.

1 | 2 | 3

A „pipe”-al elválasztott részekből valamelyiket be kell helyettesíteni

2. Az EXCEL makrók megírásával összefüggő fogalmak

2.1. Makró

A makró olyan kisméretű program, mely egyszerű utasítások egymás utáni végrehajtásával képes egy bonyolultabb feladat megoldására.

2.2. Programozás, program

A **programozás** feladata azt jelenti, hogy a hétköznapi nyelven megfogalmazott feladatot átfordítjuk a számítógép számára értelmezhető magas szintű programozási nyelvre. Azt a leírást, mely meghatározza a számítógép számára, hogy milyen műveleteket milyen sorrendben hajtson végre, **forrásprogramnak** nevezzük.

A jó program ismérvei:

- Nem tartalmaz felesleges utasításokat.
- Felhasználóbarát (csak annyi közreműködést kér a felhasználótól, amennyire feltétlenül szükség van)

2.3. Moduláris programozás

Az összetett feladatok megoldását kisebb részfeladatok megoldására vezetjük vissza. Erre kétféle megoldási lehetőségünk van:

- A bonyolult feladatot felbontjuk egyszerűbb, kisebb feladatokra (top-to-down), vagy
- Már korábban megoldott kisebb feladatokból építünk fel egy összetettebb feladatot (down-to-top).

A moduláris programozás előnye, hogy nagyban megkönnyíti a hibakeresést.

2.4. Algoritmus

Az algoritmus olyan művelet sor, mely egy bonyolultabb feladatot véges számú, előre meghatározott sorrendű részfeladatok segítségével ír le. Minden feladathoz tartozik egy kezdőállapot és egy végállapot, az algoritmus feladata pedig, hogy segítségével eljussunk a kezdőállapotból a végállapotba.

Összességében azt mondhatjuk, hogy algoritmusnak nevezzük azt a műveletek tartalmát és sorrendjét meghatározó egyértelmű *utasításrendszert*, amely a megfelelő kiinduló adatokból a kívánt eredményre vezet.

A jó algoritmus jellemzői:

- Egyértelműen megfogalmazott
- Végrehajtható
- Véges számú lépésekből áll
- Elvezet a kitűzött célhoz, azaz a feladat megoldásához.

Ahhoz, hogy megértsük mi is egy algoritmus, gondoljuk végig, hogy mit is teszünk egy teafőzés esetén.

A teafőzés egy lehetséges algoritmus:

- Ha kevés a víz a kannában, vagy üres a kanna akkor
 - Tedd a kannát a csap alá
 - Nyisd meg a csapot
 - Ismételd
 - Engedd a kannába a vizet
 - Figyeld a vízszintet

Amíg a vízszint elég magas nem lesz

 - Zárd el a csapot
- Különben tedd fel a gázra a kannát
- Gyűjtsd meg a gázt
- Ismételd
 - Melegítsd a vizet
- Amíg a víz forrni nem kezd
- Zárd el a gázt
- Öntsd ki a vizet a csészébe

2.5. Fordítóprogram

Feladata, hogy a forrásprogram utasításaiból olyan *gépi utasításokat* (alacsony szintű utasítássorozatok, amit maga a processzor végre tud hajtani) állítson elő, amelyeket végrehajtva a számítógép a kívánt eredményt szolgáltatja.

2.6. Szintaxis, szintaktikai hiba

A programozási nyelv utasításkészletének használatát meghatározó formális (nyelvtani) szabályok összessége. Ha a program írásakor nem tartjuk be a szintaktikai szabályokat, a fordítóprogram hibát jelez, mivel nem érti a leírt utasítást.


2.7. Szemantikai hiba

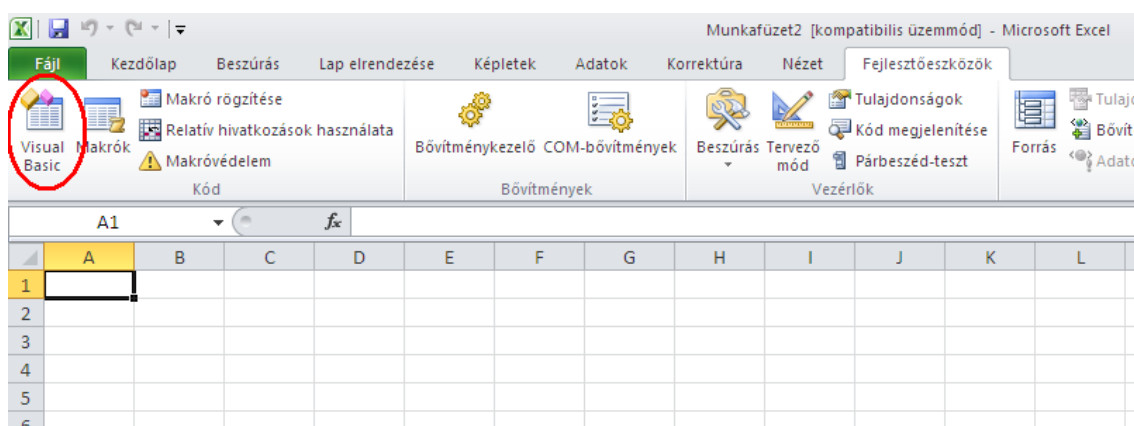
A forrásprogram nem formai, hanem logikai hibát tartalmaz, így a fordítóprogram hibát nem észlel, a forráskód lefut, de nem a várt eredmény születik. Ezen hibák megtalálása és kijavítása sokkal nehezebb, mint a szintaktikai hibáké.

3. Az EXCEL VB nézetének megismerése

Excelben a munkafüzeteket kétféle formában tekinthetjük meg. A felhasználók többsége azonban csak az alapértelmezett E-nézetet ismeri, mivel általában ebben dolgozunk. Ahhoz azonban, hogy hatékonyan tudjunk makrókat készíteni, meg kell ismerkednünk a VB nézettel is. A makrók többsége ugyanis VB nézetben születik.

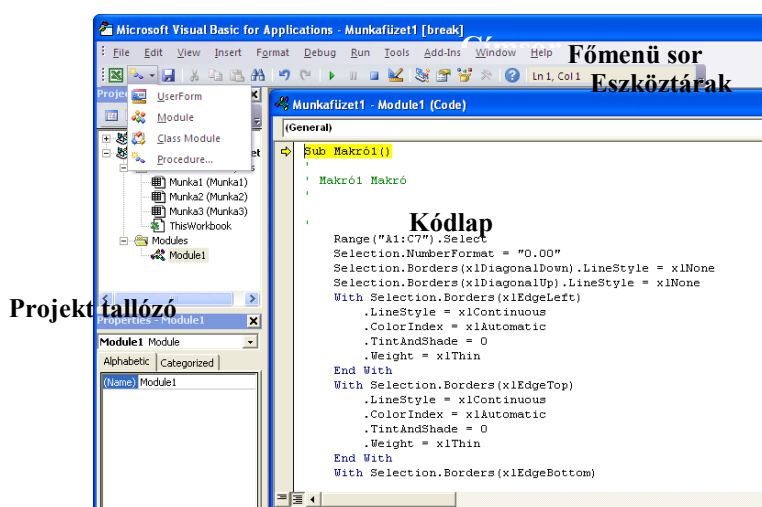
A VB nézetet az Alt+F11 billentyűkombinációval, vagy a *Fejlesztőeszközök szalaglap Visual*

Basic () ikon segítségével érhetjük el. Ha nem látható a **Fejlesztőeszközök** szalaglap, akkor a **Fájl** → **Beállítások** ablak → **Menüszalag testreszabása** részen jelöljük ki a **Fejlesztőeszközök** menüszalagot, majd OK.



1. ábra VB nézet megjelenítése

A VB nézet nagyjában hasonlít az E-nézetre, hiszen legfelül helyezkedik el a címsor, alatta a főmenü sor, ezt követik az eszköztárak. A képernyő területének legnagyobb részét pedig természetesen a munkaterület teszi ki.



Tulajdonság ablak

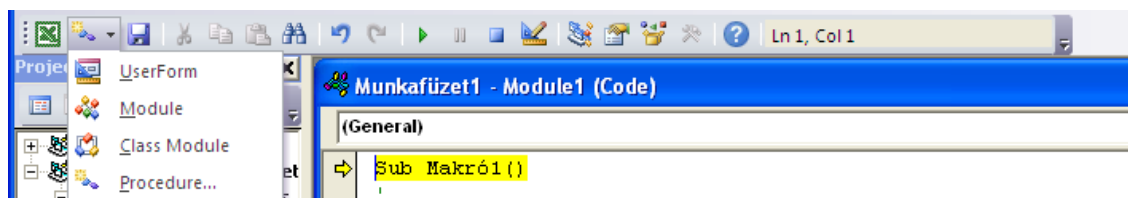
2. ábra VB nézet

Amikor létrehozunk egy munkafüzetet, a VBA *minden munkalaphoz* elkészít egy *kódlapot*. A kódlapra azon makrók kerülnek, melyek közvetlenül az *adott munkalap* egyes objektumaihoz

(parancsgombhoz, választógombhoz, beviteli mezőhöz, stb.) kapcsolódnak. A kódlap tehát a *VB nézet munkalapja*, mert itt olvashatók az adott kódlapon lévő makrók szövegei. Ezen makrók szövegét egy speciális Editorral, a Visual Basic Editorral szerkeszthetjük.

Nézzük akkor részletesen az Editor részeit:

3.1. Eszköztár



3. ábra VB nézet Eszköztár

Az eszköztáron a következő ikonokat találhatjuk meg balról jobbra haladva:

- Váltás Excel nézetre
- Beszúrás: Beszúrhatunk párbeszédlapot (UserForm), modullapot (Module), osztály modult (Class Module), illetve eljárást vagy függvényt (Procedure)
- Mentés, az aktuális objektum mentésére szolgál.
- A következő 6 ikon már ismert lehet más Office-s alkalmazásból, hiszen ezek általános szövegszerkesztésre szolgálnak. Ezen ikonok ugyanis a kivágás, másolás, beillesztés, keresés, visszavonás és az újra.
- Makró indítása: kész makró lefuttatására alkalmas.
- Makró megállítása: futás közben meggondoltuk magunkat és mégsem szeretnénk folytatni a makró futtatását, így a futási folyamatot *felfüggeszthetjük*, és szükség esetén *folytathatjuk*.
- Alaphelyzet: a makró *végleges* leállítása (ekkor már nem folytathatjuk a futást, csak előről kezdhetjük)
- Tervező mód (Design Mode): párbeszédlap létrehozására vagy vezérlőelemek elhelyezésére szolgál.
- Projekt tallózó (Project Explorer): Projekt tallózó ablak megnyitására szolgál.
- Tulajdonságok (Properties Window): a tulajdonságok ablak megnyitására szolgál.
- Objektumtallózó (Object Browser): Objektumtallózó ablak megnyitására szolgál.
- Eszközök (ToolBox): a vezérlőelemek készlete jeleníthető meg segítségével.
- Súgó
- Információs ablak: kurzor aktuális pozíciója (sor, oszlop)

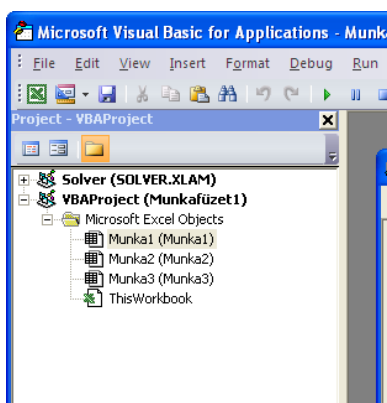
3.2. Projekttallózó ablak

Ez az ablak fastruktúrában jelenít meg *mappákat*. A mappák *tartalma* pedig a munkafüzetek *nevével* azonosítható VBA *projektek*, melyek a következők:

- Microsoft Excel objektumok: az egyes *munkalapokhoz* tartozó munkalap-objektumokat tartalmazzák. Ebben a mappában mindig megtalálható egy *ThisWorkbook* nevű objektum, mely valójában a munkafüzet-objektum, vagyis a munkafüzet maga.
- Modulok mappája: *modullapokat* tartalmaz.
- Űrlapok mappája: *párbeszédlapokat* tartalmazza.

A modulok és az űrlapok mappája csak akkor látható, ha az adott munkafüzethez tartoznak modulok illetve űrlapok. Amíg ilyet nem hozunk létre, a két mappa nem látható.

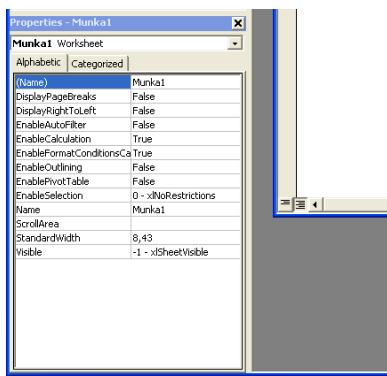
Az ablak eszköztára 3 ikont tartalmaz. A „View Code” a makró(k) kódjának megjelenítésére szolgál; a „View Object” magának az objektumoknak a megjelenítésére, míg a „Toggle Folders” a mappák ki/be kapcsolására szolgál.



4. ábra Projekttallózó ablak

3.3. Tulajdonságok ablak

A tulajdonságok ablakban a kijelölt objektum tulajdonságait lehet megjeleníteni és módosítani. Tulajdonság például az objektum szélessége, a betűtípus-objektum neve, stílusa és a színe.




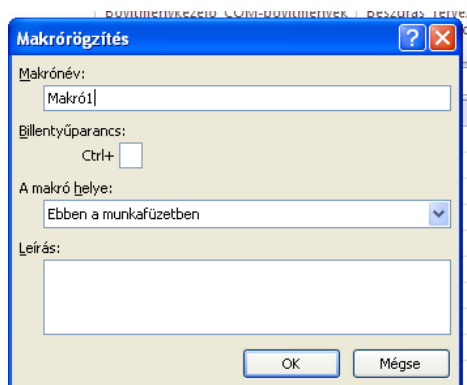
5. ábra Tulajdonságok ablak

4. Makró készítése felvétellel (varázslóval)

A makrógenerátorral (felvétellel) létrehozott makrókat a Visual Basic Editor mindig modulapon helyezi el. A modullap mindig a *Modul<n>* nevet kapja, ahol az n egy sorszám. Az egymás után generált makrók egy modullapra kerülnek, de a munkafüzet minden egyes megnyitásakor a makrógenerátor újbóli elindításával új modullap keletkezik.

Makró létrehozása makrógenerátorral

- 1) Kattintsunk a Fejlesztőeszközök szalaglap „makró rögzítése” (), vagy a *Nézet* → *Makrók* → *Makró rögzítése* ikonra.
- 2) A megjelenő makrórögzítés ablakban töltsük ki a beviteli mezőket. A rutinnak lehetőleg a tevékenységére utaló nevet adjunk. A leírás mezőbe a makró jellemzőit tudjunk felsorolni, például a készítő nevét, a makró funkcióját, stb. A billentyűparancs kijelölő négyzet pedig arra szolgál, hogy ott adhatjuk meg azt a betűt, amelynek a CTRL-al történő egyidejű lenyomása esetén a makró azonnal elindítható (kitöltésekor a CTRL billentyűt nem szabad lenyomni! – pl. ha a makró a CTRL + SHIFT + M billentyűkombinációval szeretnénk elindítani, akkor a mezőben csak a SHIFT + M billentyűt nyomjuk meg).



6. ábra Makrórögzítés

- 3) Az OK gombra kattintva elindul a makrógenerátorral a felvétel. Ez azt jelenti, hogy az Excelben innentől bármire is kattintunk vagy bármit is teszünk, azt rögzíti a generátor. A *Fejlesztőeszközök* szalaglap megváltozik, és két ikonnal egészül ki, ezek a „rögzítés vége”



és a „relatív hivatkozások használata” .

- 4) Végezzük el a feladatokat, majd a „rögzítés vége” ikonra kattintva állítsuk meg a felvételt. FIGYELEM! A makró a rögzítés végén ne felejtsük el leállítani, mert sok fejtörést és programkiakadást megelőzhetünk.
- 5) A felvett műveleteket, vagyis az eredményt megtekinthetjük, ha átváltunk VB nézetre. A projektatló ablakban meg kellett jelenjen egy *Modulok* nevű mappa, azon belül pedig egy *Modul1* nevű modullap. Ha megtekintjük az angol nyelvű forráskódot, láthatjuk, hogy a generátor megjegyzéseket is beszúr. Megfigyelhető továbbá, hogy a VBA kulcsszavak kék, a megjegyzések zöld színnel, míg a hibás sorok piros színnel jelennek meg.

Ezek a színbeállítások azonban tetszés szerint módosíthatók az *Eszközök (Tools → Egyebek (Options) → Editor Format (szövegszínek)* menüpont alatt.

A makrógenerátorral létrehozott makrók legnagyobb hátránya, hogy a varázslónak köszönhetően felesleges sorokkal lehet tele.

Feladat

Makrógenerátor segítségével készítsünk egy makrót, mely a munkafüzetbe beszúr egy munkalapot, majd átnevezi a keletkezett munkalapot „második” névre. Ezután jelöljük ki a D1:D5 tartományt és kettős kék színű kerettel keretezzük körbe.

Egy lehetséges megoldás

```

Sub Elso_gyakorlof()
' Elso_gyakorlof Makró
' Billentyűparancs: Ctrl+g
  Sheets.Add After:=Sheets(Sheets.Count)

  Sheets("Munka4").Select

  Sheets("Munka4").Name = "második"
  Range("D1:D5").Select

  Selection.Borders(xlDiagonalDown).LineStyle = xlNone
  Selection.Borders(xlDiagonalUp).LineStyle = xlNone

  With Selection.Borders(xlEdgeLeft)
    .LineStyle = xlDouble
    .ThemeColor = 4
    .TintAndShade = -0.499984740745262
    .Weight = xlThick
  End With
  With Selection.Borders(xlEdgeTop)
    .LineStyle = xlDouble
    .ThemeColor = 4
    .TintAndShade = -0.499984740745262
    .Weight = xlThick
  End With
  With Selection.Borders(xlEdgeBottom)
    .LineStyle = xlDouble
    .ThemeColor = 4
    .TintAndShade = -0.499984740745262
    .Weight = xlThick
  End With
  With Selection.Borders(xlEdgeRight)
    .LineStyle = xlDouble
    .ThemeColor = 4
    .TintAndShade = -0.499984740745262
    .Weight = xlThick
  End With
  Selection.Borders(xlInsideVertical).LineStyle = xlNone
  Selection.Borders(xlInsideHorizontal).LineStyle = xlNone
End Sub

```

Rutinfej

megjegyzések

Munkalap beszúrása a többi után, az Add metódus ellentéte a Delete, mellyel törölni lehet a kijelölt objektumot.

Az átnevezni kívánt munkalap kijelölése

A kijelölt munkalap átnevezése

D1:D5 tartomány kijelölése

Ez a két sor azt mutatja, hogy a belső átlós vonalakat törli, pedig nem is állítottuk át

*A keretek lehetséges oldaléleit a **Borders** konténer tartalmazza. Ezek közül az indexek megadásával lehet kijelölni a megfelelőt. Indexnek azonban itt nem sorszámot, hanem helyzetükre utaló neveket alkalmazunk.*

- Felső/alsó oldalél: xlTop/xlBottom
- Bal oldali/jobbi oldali oldalél: xlLeft/xlRight
- Belső vízszintes/függőleges választóvonalak: xlInsideHorizontal/xlInsideVertical
- Belső lefelé/felfelé átlós választóvonal: xlDiagonalDown/xlDiagonalUp

Minden élnek 4 tulajdonsága van:

- Stílusa (LineStyle)
- Vastagsága (Weight)
- Szín csoport (ThemeColor)
- Csoporton belüli színárnyalat

Ez a két sor azt jelenti, hogy a kijelölt tartományon belül lévő vízszintes és függőleges választóvonalakat törli, pedig ezeket sem állítottuk át.

5. Önálló EXCEL makrók megírásához szükséges programozási ismeretek

Ahhoz, hogy a makrókról értelmesen tudjunk beszélni, először el kell sajátítanunk bizonyos *programozás-elméleti* kérdéseket. *A makró írás ugyanis nem bemagolható, mechanikus lépések sorozata, hanem komoly háttér-információk meglétét igénylő, gondolkodtató feladat.*

A számítógépek nem rendelkeznek semmilyen intelligenciával, csupán arra képesek, hogy a programozó által meghatározott utasítások (műveletek) sorrendjét a megadott módon végrehajtsák. Nézzük tehát, hogy is készül egy program, vagy jelen esetben egy makró!

5.1. A programfejlesztés lépései

5.1.1. Feladatspecifikáció

A leendő felhasználó által megfogalmazott *feladateleírás* a megoldandó problémáról.

5.1.2. Algoritmus készítés

Az algoritmusok leírására *pszeudokódot* (mondatszerű leírást) szoktunk leggyakrabban használni. Ez a kód átmenet az emberi nyelv és a programozási nyelv között. A pszeudokódban többnyire *hétköznapi szavakkal* írjuk le a feladatokat, de már olyan kifejezéseket használunk, melyek *nagyban hasonlítanak* a programozási utasításokra.

Például:

- **Adatbekérési feladat:** *Be:* adatnév
- **Tartomány kijelölése:** *kijelölés:* objektumnév

Konkrét algoritmus példa:

Határozzuk meg egy szám abszolút értékét!

Megoldás:

Be: Szám

Ha Szám ≥ 0 **akkor**

Ki: Szám

Különben

Ki: $(-1) \cdot \text{Szám}$

Elágazás vége

5.1.3. Kódolás

A *kódolás* az elkészített algoritmus megadása valamilyen *magas szintű programozási nyelven*. Ezt a korszerű programozási technikát *strukturált programozásnak* nevezzük.

5.1.4. Tesztelés, dokumentáció

A programunk működésének helyességét teszteléssel lehet ellenőrizni. Éles használat előtt mindenképpen tesztadatokkal próbáljuk ki programjainkat. A teszteléshez hasznos segítséget nyújtanak a Visual Basic üzenetküldésekre és válaszkérésekre szolgáló lehetőségei is, amelyeket az *Üzenetküldés és válaszkérés* című fejezetben ismertetek részletesebben.

5.2. A strukturált program részei

5.2.1. Rutinfej

A rutinfej a rutin *első* sora. A rutinfej tartalmazza az adott eljárás vagy függvény (későbbiekben részletesen kitérünk mivoltukra) nevét, esetleges paramétereinek (ha vannak) felsorolását.

A VBA nyelvben a név megadása kötelezően betűvel kezdődik, és betűket, számokat és jeleket (pl. aláhúzás) szóközők nélkül tartalmazó, maximum 255 karakter hosszú karaktersorozat. Lehetőleg könnyen megjegyezhető, utaló neveket adjunk alkalmazásainknak, ezáltal sok bosszúságtól kímélhetjük meg magunkat. Az ideális név 5-10 karakter hosszú. A beépített tulajdonságok, utasítások nevét érdemes mindig kisbetűvel írni, mivel a fordító ezeket a felismerés után automatikusan nagybetűsre változtatja, így egy elgépelési hiba könnyen felismerhetővé válik.

Pl.: Sub makrorogzites()

5.2.2. Deklarációs rész

A deklarációs rész tartalmazza az alapértelmezett *beállításokat*, a *konstansok*, *változók nevét*, *típusát*. Tulajdonképpen itt azt fogalmazzuk meg, hogy az adott *értékre* vagy *memóriában tárolt adatokra* milyen *névvel* hivatkozunk és ezen értékek milyen *értéktartományba* esznek, továbbá, hogy *szöveges*, *szám* vagy *logikai* az adott érték.

A deklaráció háromféle lehet:

- *Lokális* (rutin szintű) *deklaráció*: A lokálisan deklarált konstansok és változók csak abban az egy rutinban léteznek, amelyben deklaráltuk őket. A rutin lefutása *után* elvesztik értéküket, kivéve a **Static** kulcsszóval megjelölt statikus konstansok, változók.
- *Globális deklaráció*: Az ily módon deklarált változók *alkalmazásszintűek*, azaz *minden modulban* használhatóak. Globális változókat *csak modullapon* hozhatunk létre!
- *Modális deklaráció*: Az így létrehozott változók mind az aktuális kódapon, mind a modullapon léteznek, és az itt elhelyezkedő rutinok mindegyike használhatja ezeket, de másik kódapon/modullapon elhelyezkedő rutin viszont nem.

Ha különböző szintű (pl.: lokális és globális) változókat hoztunk létre azonos néven (kezdő programozóknak azonban nem ajánlom), akkor futás során mindig a legalacsonyabb szintű deklaráció érvényes.

Deklarálni lehet *konstansokat, változókat és típusokat*.

5.2.2.1. Konstansok

A konstans egy, a program futása során *állandó* érték. Alkalmazása akkor lehet ésszerű, ha egy konkrét adatot többször szeretnénk használni a programozási folyamatban. Ilyenkor ugyanis csak egyszer, a deklaráció során kell megadni a konkrét *értéket* és utána a konstans *névére* hivatkozunk, így kisebb az elgépelésekből, utólagos módosításokból adódó hibák lehetősége. A konstansokat mindig a modullapok, rutinok *elején* kell összegyűjteni.

A konstans deklarálás szintaxisa VBA nyelven a következő:

```
[Public|Private] Const <konstansnév> [As <típusnév>] = <kifejezés>
```

A globális (Public) konstansok az alkalmazás minden kódlapjára érvényesek, a program során bárhol felhasználhatóak. Deklarálásuk azonban csak modullapon lehetséges, kódlapon nem. A Private kulcsszóval az aktuális kódlapra korlátozódik a konstans hatásköre.

Konkrét példák konstans deklarálásra:

- Egész típusú konstans deklarálása:

```
Const Maximum As Long = 32767
```

- Szöveg típusú konstans deklarálása:

```
Const szoveg = "Jó tanulást kívánok!"
```

- Dátum konstans deklarálása:

```
Const karacsony = #24/12/2012#
```

- Logikai konstans deklarálása:

```
Const valasz = False
```

5.2.2.2. Típusok

A programban *minden* adatnak rendelkeznie *kell* valamilyen *típussal*. A típus megválasztásakor a programozó arról dönt, hogy a fordítóprogram *mekkora tárhelyet* foglaljon az adat számára, valamint meghatározza, hogy szöveges, szám vagy logikai adatról van szó, mely befolyásolja az adattal végezhető műveletek körét. Például szöveges típusú adatnak deklarált változók esetén a fordító nem engedi meg, hogy az adattal osztási vagy szorzási műveletet végezzünk.

5.2.2.3. Változók

A változó *névének és típusának az összerendelését a változó deklarálásának* nevezzük.

A változó deklarálás szintaxisa VBA nyelven a következő:

```
[Public|Private] Dim változónév1 [As típusnév1] [, változónév2  
[As típusnév2]...]
```

Ha a kulcsszó után egynél több változónevet sorolunk fel, akkor ezeket vesszővel kell elválasztani egymástól. Típus nélküli változó nem létezik, így ha nem adunk meg típust, a fordító alapértelmezettként *Variant* típusúnak értelmezi.

A *Public* kulcsszó után deklarált változók *globálisak*, vagyis más modullapokról is elérhetőek. A *Private* kulcsszó után deklarált változók *modálisak*, vagyis csak az aktuális modullapon léteznek.

Példa változó deklarálásra

```
Dim alakult As String
```

```
Dim szam As Integer, alakultszam As Integer, alapszam As Single
```

Példák különböző típusú változók deklarálásra

```
Dim datum1 As Date, datum2 As Date
```

```
datum1 = #5/1/2012 11:20:05#
```

```
datum2 = #2012-04-12#
```

```
Dim szoveg As String
```

```
szoveg = "Visual Basic"
```

```
Dim szoveg2 As string*12
```

```
Szoveg2 = „próba      ”
```

```
Dim objektum As pelda
```

```
Set objektum = Text1
```

```
Objektum.Text = „objektumhoz készítünk referencia tartományt”
```

(tulajdonság)

```
Objektum.Move 10, 10
```

(metódushívás)

```
Dim a,b,c As Variant
```

(Mind a 3 változó kezdetben Variant típusú lesz. Az értékadásnál dől el, hogy milyen típusú változók lesznek.)

```
a = 2012
```

(integer típusú)

```
b = "1234"
```

(szöveg típusú)

```
c = a + b
```

(double típusú)

```
Dim tomb1(1 to 12) As Integer
```

```
Dim tomb2(1 To 10, 3 To 6) As Byte
```

```
Dim tomb(10) As Byte
```

(a tomb változó 11 elemű tömb, 0-tól 10-ig címezhető)

```
Option Base 1
```

```
Dim tomb(10)
```

(a tomb változó 10 elemű tömb, 1 től 10-ig címezhető)

```
Dim tomb3(3,3,2) As String
```

(a tomb3 változó egy 3 dimenziós (4x4x3 elemű) tömb.)

A típusokat két nagy csoportba sorolhatjuk: beépített típusok vagy saját, felhasználó által létrehozott típusok.

A beépített típusokat több csoportba sorolhatjuk:

EGYSZERŰ ADATTÍPUSOK			ÖSSZETETT ADATTÍPUSOK
SZÁM és DÁTUM TÍPUSOK	SZÖVEGES TÍPUSOK	LOGIKAI TÍPUS	
<ul style="list-style-type: none"> – Byte (<i>Byte</i>): 1 bájtban tárolja az adatokat. Olyan egész szám típusú adatokat deklarálunk Byte típusúnak, melyek értéke 0 és 255 közé esik. – Integer (<i>Int</i>): 2 bájtban tárolja az adatokat. Olyan egész szám típusú adatokat deklarálunk Integer típusúnak, melyek értéke -32768 és +32767 közé esik. – Long (<i>Lng</i>): 4 bájtban tárolja az adatokat. Olyan egész szám típusú adatokat deklarálunk Long típusúnak, melyek értéke -2147483648 és +2147483647 közé esik. – Single (<i>Sng</i>): Egyszeres pontosságú (4 bájtban tárol), lebegőpontos típusú valós számok esetén alkalmazzuk. Értékhatárai: ±3.402823 E38 – Double (<i>Db</i>): Kétszeres pontosságú (8 bájtban tárol), lebegőpontos típusú valós számok esetén alkalmazzuk. Értékhatárai: 1.797693134862315 E308 	<ul style="list-style-type: none"> – String (<i>Str</i>): A string típusú változókat változó hosszúságú szöveges adatok tárolására használjuk. A string változó hosszúságú, de maximum 2147000000 karakter hosszú karakterláncot képes tárolni. A String típusú változó tényleges hossza 10+ a karakterek száma. Ez az érték a Len(változónév) függvény segítségével kérdezhető le. – String*n: Abban különbözik a String típustól, hogy fix hosszúságú karakterláncok definiálására szolgál. Ez azt jelenti, hogy abban az esetben, ha a szó rövidebb, mint a megadott n karakter, akkor jobbról szóközökkel tölti fel. Ha azonban a beviendő karaktersorozat hosszabb, mint a megadott n paraméter, akkor az utána lévő karakterek elvesznek. A felesleges szóközöket a szövegünk elejéről, végéről illetve mindkét oldaláról a LTrim(szoveg); RTrim(szoveg) illetve a Trim(szoveg) függvényekkel távolíthatjuk el. 	<ul style="list-style-type: none"> – Boolean (<i>Bool</i>): 2 Bájtot foglal, így valódi 1 bájtos logikai típus VBA nyelvben nem létezik. A true (igaz) logikai értéknek -1, míg a false (hamis) értéknek 0 a számbeli megfelelője. Fordítva azt mondhatjuk, hogy minden 0-tól különböző érték True. Érdekesség, hogy minden összehasonlító kifejezésben először a logikai változó értékelődik ki, így az 5=True kifejezés, biztosan hamis értéket ad vissza, mivel 5≠-1 – el. 	<ul style="list-style-type: none"> – Object (<i>Obj</i>): Egy objektumhoz Set paranccsal rendelhetünk Object típusú változót, mely rendelkezik az objektum tulajdonságaival és metódusaival is. – Tömb: Azonos típusú konstansok vagy változók rendezett sorozata. Megkülönböztetünk statikus tömböket és dinamikus tömböket, illetve egy és többdimenziós tömböket. <u>Statikus tömb</u>: A tömb adott darabszámú azonos típusú elemből áll. Az elemeket sorszámmal (un. indexel) azonosítjuk. <i>Deklarálásuk</i>: a tömb nevét kerek zárójelek között követik az indexhatárok és As kulcsszóval adjuk meg az elemek adattípusát. <i>Dim tömbnév (alsó index To felső index) As elemek adattípusa</i> Több dimenziós tömb esetén annyi indexhatár párt kell megadni vesszővel elválasztva, ahány dimenziós a tömb.

EGYSZERŰ ADATTÍPUSOK			ÖSSZETETT ADATTÍPUSOK
SZÁM és DÁTUM TÍPUSOK	SZÖVEGES TÍPUSOK	LOGIKAI TÍPUS	
<ul style="list-style-type: none"> – Currency (Cur): 8 bájtos memóriagényű. Valutaértékek gyors és pontos számítására szolgál, vagyis fixpontos valós számokat tárol 11 egész- és 4 tizedesjegy pontosan. – Decimal: Hasonló a Currency típushoz, azzal a különbséggel, hogy a tárolás 64bit helyett 96 biten történik, 28 jegy pontossággal, de a tizedespont helye nem rögzített. – Date (Date): Ez az adattípus dátumokat és időpontokat tárol kétszeres pontosságú számként, ahol is az egészrész a napok számát, a törtrész a napon belüli időpontot jelenti. A Date típusú változó 0 értékét 1899.dec.30. 0:00:00 időpont adja. 	<p>A stringek karaktereire pedig a Mid(szoveg, ahányadik karaktertől, ahány karaktert) függvény segítségével hivatkozhatunk.</p> <p>Például:</p> <p>Szoveg="Visual Basic"</p> <p>Mid(szoveg, 2,1) = „D” → szöveg új értéke: VDsual Basic lesz.</p>		<p><u>Dinamikus tömb:</u></p> <p>Méretüket nem a fordítás, hanem a futás során kapják, mivel a tömböt indexhatárok nélkül deklaráljuk.</p> <p>A dinamikus tömb deklarációja a tömb nevét, üres zárójelpárt és az elemek típusát tartalmazza.</p> <p>Dim Dtomb () [As Integer]</p> <p>Az ilyen tömböket futás során a Redim utasítással hozzuk létre. Például: Redim [Preserve] Dtomb([1 To] 12) As Integer</p> <p>A Redim ismételt alkalmazásával a tömb bármikor újraméretezhető. A Preserve módosító megadásával a ki nem eső adatok megmaradnak. Az Option Base {0 1} paranccsal a tömbök induló értéke adható meg. Alapértelmezett: 0</p> <p>Variant típusú változókhoz az Array függvénnyel rendelhetők tetszőleges dimenziószámú és típusú adatok. Például:</p> <p>Dim Napok</p> <p>Napok=Array(„hétfő”, „kedd”, „szerda”, „csütörtök”, „péntek”, „szombat”, „vasárnap”)</p>
<ul style="list-style-type: none"> – Variant (Var): Lehetővé teszi, hogy ugyanazon változóban különböző típusú adatokat tároljunk. Ez első látásra kényelmesnek tűnhet, azonban az ilyen adattípusú változóknak köszönhetően lassul a programfutás, illetve nagyobb memóriaterületet foglal. Speciális értékek jelzik, hogy a Variant típusú változó még nem kapott értéket (Empty), vagy tartalma nem felel meg semmilyen típusnak sem (Null) vagy az érték sikertelen típus-átalakítás során jött létre (Error). Az ilyen típusú változó létrehozás utáni kezdőértéke Empty. 			

5.2.2.4. Saját, felhasználó által létrehozott típusok

A felhasználó által megalkotott adattípus olyan összetett adatszerkezet, amely tetszőleges számú, különböző típusú részből állhat. Az ilyen típusok deklarációját a **Type** és az **End Type** foglalt szavak *között* kell elhelyezni a modul deklarációs részében.

Általános szintaxisa:

```
Type <sajat_tipusom>  
    <Sajat_valtozonev1> As <típusnév>  
    [<Sajat_valtozonev2> As <típusnév>]  
    [...]
```

End Type

Az ilyen típusok belső elemeire úgy kell hivatkozni, hogy a típusnévhez ponttal kapcsoljuk a belső változónevet.

Konkrét példa saját felhasználói típus deklarálásra, és értékadásra:

```
Private Type Datumom
```

```
    Ev As Integer
```

```
    Honap As Byte
```

```
    Nap As Byte
```

End Type

```
Dim Szulnap1 As Datumom
```

```
Szulnap.Honap = 2
```

```
Szulnap.Nap = 15
```

```
Szulnap.Ev = 1915
```

```
With Szulnap
```

```
    Nap = 15
```

```
    Ev = 1915
```

```
    Honap = 2
```

End With

5.2.2.5. Típuskonverziók

A VB nyelv majdnem minden típust képes átalakítani valamilyen más adattípusra. Ehhez megfelelő átalakító-függvények állnak rendelkezésünkre. A legfontosabb ilyen függvényeket az alábbiakban ismertetem.

FÜGGVÉNY NEVE	AZ ÁTALAKÍTÁS EREDMÉNYE
<i>Karaktersorozat átalakítása számmá:</i>	
CByte (kifejezés) CInt (kifejezés) CLng (kifejezés)	Tetszőleges szöveges, vagy szám típus átalakítása Byte, Integer vagy Long típusúvá.
Val (szöveg)	A függvény megadja a szövegben tárolt egész vagy valós szám értékét. Ha az átalakítás nem sikerül, 0 értékkel tér vissza.
Asc (string) AscB (string) AscW (String)	A függvény a paraméterként megadott szöveg első karakterének kódját adja vissza. Az AscW függvény a Unicode táblázatot használja.
<i>Számok átalakítása karaktersorozattá:</i>	
Str (kifejezés)	A függvény a paraméterként megadott egész vagy valós számot karaktersorozattá alakítja.
Chr (kód) ChrB (kód) ChrW (kód)	Egy karakteres szöveg előállítása a karakterkód alapján. A ChrW függvény a Unicode táblázatot használja.
Hex (kifejezés)	A kifejezés átalakítása hexadecimális (16-os számrendszerbeli) karaktersorozattá.

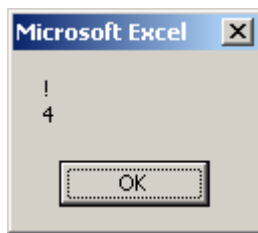
FÜGGVÉNY NEVE	AZ ÁTALAKÍTÁS EREDMÉNYE
<i>Valós számok átalakítása egész számmá:</i>	
CInt(valós szám)	A legközelebbi egész számmá való alakítása. Pl.: 3,6→4 lesz.
Round(szám)	Kerekítés a legközelebbi egész számra.
Fix(valós szám)	Egész számmá alakítja a valós számot úgy, hogy a törtrészt levágja. Pl.: 3,6→3
<i>Dátummal/idővel kapcsolatos átalakítások:</i>	
DateValue(kifejezés)	Tetszőleges kifejezés átalakítása dátummá.
Year(dátum)	Év lekérdezése a dátumból.
Month(dátum)	Hónap lekérdezése a dátumból.
Day(dátum)	Nap lekérdezése a dátumból.
TimeValue(kifejezés)	Tetszőleges kifejezés átalakítása idővé.
Hour(idő)	Óra lekérdezése az időből.
Minute(idő)	Perc lekérdezése az időből.
Second(idő)	Másodperc lekérdezése az időből.

Példa átalakító függvények használatára

```

Sub atalakitofuggv()
    Dim Egesz As Integer, Tizedes As Single
    Dim Szoveg As String, Kerekített As Integer
    Egesz = 33
    Tizedes = 3.6
    Szoveg = Chr(Egesz)
    Kerekített = CInt(Tizedes)
    MsgBox Szoveg & vbCrLf & Kerekített
End Sub

```



7. ábra Átváltó függvények eredménye

5.2.3. Törzs

A rutintörzs utasítások alkotják. Ezen utasítások alkotják tulajdonképpen magát a tényleges programot, mely az algoritmusban megfogalmazott célt megvalósítja.

A strukturált programok alapvetően 3 féle szerkezetből épülnek fel:

5.2.3.1. Szekvencia

Egyszerű utasítások *egymás utáni* végrehajtása. (Például bekérések, kiírások, értékadások...)

ÉRTÉKADÓ UTASÍTÁS

A leggyakrabban használt VB utasítás az értékadás. Ezen utasítás esetén a bal oldalon álló változó felveszi a jobb oldalon megadott kifejezés értékét.

Általános szintaxisa:

Változónév = kifejezés

Ha a bal oldali változó típusa nem azonos a kifejezés által előállított típussal, a fordító automatikusan a megfelelő típusúra váltja át.

Az operátorok típusától függően beszélünk aritmetikai, logikai, hasonlító vagy szöveges értékadásról.

Aritmetikai értékadás a hatványozás(^), szorzás, osztás, hányados(\) vagy maradék(MOD) meghatározás vagy az összeadás és a kivonás.

Példa aritmetikai értékadásra:

$Y = X + 7$

Logikai értékadás a tagadás vagy inverz meghatározás (Not), az ÉS művelet (And), a VAGY kapcsolat (Or), a kizáróVagy kapcsolat (XOR), az ekvivalencia (Eqv) és az implikáció (Imp).

Példa logikai értékadásra:

If (a>7) and (b<10) then

Szöveges művelet például a karakterláncok összefűzése (&).

Példa szöveges értékek összehasonlítására:

"alma" LIKE "a*a" → true értéket ad vissza (A csillag tetszőleges karakter helyettesítésére szolgáló karakter.)

Hasonlító operátorral előállított értékadó kifejezés például, ha objektumváltozókat hasonlítunk egymáshoz (Is), vagy vegyük csak az egyszerű, mindennapokban is használt relációs jelekkel meghatározható hasonlításokat.

Feladat: Az A1-es cellában jelenjen meg a Hello felirat, és a cella háttérszíne legyen piros.

```
Sub proba()  
    Cells(1,1).value = "Hello" vagy Cells(1,1) = "Hello"  
    Cells(1,1).Interior.Color = vbRed  
End Sub
```

Ha olyan makrót szeretnénk írni, mely nem használ vezérlőket és a munkafüzet minden munkalapján egyformán tud futni, akkor modulon kell megírunk.

A Value a cella leggyakrabban használt tulajdonsága, mely a cella értékét jelenti. Alapértelmezett tulajdonság, ezért ha nem írjuk ki, akkor ezt tudjuk megadni. A háttérszín megadásnál az angol színmegnevezések többsége használható, csak elé kell írni, hogy vb.

5.2.3.2. Szelekció (Elágazás)

Az elágazás azt jelenti, hogy a program futását meg kell szakítani egy adott ponton, és a program utasításainak végrehajtása egy feltétel teljesülésétől vagy nem teljesülésétől függ. A feltételt a számítógép értékeli ki. Ilyennel már találkozhattunk a HA függvény megismerésekor. Tehát, elágazásra akkor van szükségünk, ha egyes utasításcsoportokat csak bizonyos feltételek teljesülése esetén kell végrehajtani.

A VBA nyelvben kétféle elágazásképző utasítást használunk:

Az **If ... End If** elágazást akkor használjuk, ha 2-3 feltétel szerint akarjuk végrehajtatni.

```
If <feltétel> Then  
    [<Utasítás(ok)>]  
ElseIf <feltétel> Then  
    [<Utasítás(ok)>]  
Else  
    [<Utasítás(ok)>]  
End If
```


A **Select Case ... End Select** elágaztatást akkor használjuk, ha az elágaztatást kettőnél *több* feltétel szerint akarjuk végrehajtani.

Select Case <változónév>

 [**Case** <értéklista>
 [<Utasítás(ok)>]]

 [**Case** <értéklista>
 [<Utasítás(ok)>]]

 [**Case Else**
 [<Utasítások>]]

End Select

Pl, ha menüt készítünk, akkor a menüpontok választása és az egyes menüpontok megnyomása után milyen feladatot hajtson végre a program.

Példa az elágazások használatára:

Feladat, két szám közül eldönteni, melyik a nagyobb.

If a > b **Then**

 nagyobb = a

Else

 nagyobb = b

End If

Feladat: Adjunk a B1-es cellának egy számértéket, és a C1-es cellába írjuk ki, hogy a szám páros vagy sem.

Sub proba2()

Cells(1, 2) = 19

Ezt a számértéket folyamatosan átírhatjuk

If Cells(1, 2) Mod 2 = 0 **Then**

 Cells(1, 3) = "Páros"

Else Cells(1, 3) = "Páratlan"

End Sub

5.2.3.3. Iteráció (Ciklus)

Ciklusokra akkor van szükségünk, ha egy utasításcsoportot többször meg kell ismételnünk. Az ismétlődő utasításcsoportot *ciklusmagnak* nevezzük.

A ciklusoknak több fajtáját különböztetjük meg:

Számláló (For) ciklus

Akkor alkalmazzuk, ha pontosan tudjuk, hogy a ciklusmagnak hányszor kell megismétlődni.

For ciklus szintaxisa VBA nyelven:

For <ciklusváltozó> = <kezdőérték> **To** <végérték> [**Step** <lépésköz>]

[<Utasítás(ok)>] Ciklusmag

[**Exit For**]

[<Utasítás(ok)>]

Next [<ciklusváltozó>]

A ciklusváltozó egy egész szám típusú adat. A ciklus úgy működik, hogy a ciklus kezdetén a ciklusváltozó felveszi a kezdeti értéket és ha a ciklusváltozó nem nagyobb, mint a végérték, akkor a ciklusmag lefut. Amikor a program futása elérkezik a Next kulcsszóhoz, a VBA hozzáadja a ciklusváltozóhoz a lépésköz értékét (ha elhagyjuk, a lépésköz 1 lesz) és megismétli a ciklusváltozó vizsgálatát. A ciklusváltozó megfelelő értéke esetén lefut ismét a ciklusmag. A ciklus akkor ér véget, ha a ciklusváltozó értéke meghaladja a végértéket.

A számláló ciklus másik esete, amikor nem tudjuk előre az ismétlések darabszámát, csak azt tudjuk, hogy egy objektumhalmaz mely elemével kell a műveleteket elvégezni. Előnye, hogy lehetővé teszi, hogy akár tömbben tárolt adatokat indexek használata nélkül bejárjunk.

Ekkor a szintaktika a következő:

For Each <elem> **In** <objektumhalmaz>

[<Utasítás(ok)>] Ciklusmag

[**Exit For**]

[<Utasítás(ok)>]

Next [<elem>]

Példa számláló ciklus használatára:

Feladat: 1 és 10 kitevő közötti értéknél írassuk ki 2 hatványait.

```
Sub hatványozas()  
    Dim i As Integer  
    For i = 1 To 10  
        Cells(i, 5) = 2 ^ i  
    Next [i]  
End Sub
```

Feladat: Töltsük fel a D1:D10 tartományt az első 10 pozitív egész számmal.

```
Sub Kitoltes()
    For i = 1 To 10
        Cells(i, 4) = i
    Next
End Sub
```

Feladat: Töltsünk fel egy tömböt 10-től 100-ig tízesével, majd a tömb számait a For Each segítségével összegezzük, az eredményt az MSGBOX segítségével írjuk ki.

```
Sub Osszegez()
    Dim tomb(9), i, a
    For i = 0 To 9
        tomb(i) = 10 * (i + 1)
    Next
    osszeg = 0
    For Each a In tomb
        osszeg = osszeg + a
    Next
    MsgBox "A tomb nevű tömbben tárolt számok összege: " &
    osszeg
End Sub
```

Feltételes ciklusok

Nem tudjuk, hogy a ciklusmag hányszor fog megismétlődni, mivel a ciklusmagot egy feltétel teljesülése vagy nem teljesülése alapján ismétli meg. Tehát a feltételes ciklusokat aszerint különböztetjük meg, hogy a ciklusmagot a feltétel *teljesülése* vagy *nem teljesülése* esetén ismételjük meg, illetve hogy a feltétel a ciklusmag *előtt* vagy *után* helyezkedik el. A ciklusok végét a **Loop** utasítás jelzi.

Elöltesztelő ciklus: A feltétel kiértékelése a ciklusmag végrehajtásának megkezdése *előtt* történik, ami azt jelenti, hogy előfordulhat, hogy a ciklusmag *egyszer sem* fut le.

Elöltesztelő ciklusok szintaxisa VBA nyelven (a ciklus addig fut, amíg a feltétel teljesül, vagy nem teljesül.):

Do While <feltétel>	While <feltétel>	Do Until <feltétel>
[<Utasítás(ok)>]	[<Utasítás(ok)>]	[<Utasítás(ok)>]
[Exit Loop]	Wend	[Exit Loop]
[<Utasítás(ok)>]		[<Utasítás(ok)>]
Loop		Loop

A **Do While ... Loop** és a **While ... Wend** ciklusok csak abban különböznek, hogy a **While ... Wend** esetén a ciklusmagból nem lehet kilépni.

Hátultesztelő ciklus: A ciklusmag egyszer mindenképpen lefut, mivel a feltétel kiértékelése a ciklus végén történik meg.

Hátultesztelő ciklusok szintaxisa VBA nyelven:

Do	Do
[<Utasítás (ok)>]	[<Utasítás (ok)>]
[Exit Loop]	[Exit Loop]
[<Utasítás (ok)>]	[<Utasítás (ok)>]
Loop While <feltétel>	Loop Until feltétel

A feltételes ciklusoknál minden esetben kell lennie egy utasításnak, mely módosítja a kiértékelendő feltétel értékét, különben az ismétlés sohasem áll le, a gép *végtelen ciklusba* kerül.

Példa feltételes ciklusok használatára:

Feladat egy nevezetes programozási tétel, az összegzés tételének használata, vagyis 1 és 20 közötti számokat összegezzünk.

```
Sub osszegzes()  
    Dim osszeg As Integer  
    Dim szam As Integer  
    osszeg = 0  
    szam = 1  
    While szam < 21  
        osszeg = osszeg + szam  
        szam = szam + 1  
    Wend  
    Cells(1, 7) = osszeg  
End Sub
```

Feladat: Készítsünk makrót, mely az I oszlopot feltölti egész számokkal 1-től 20-ig.

Sub eloltesztelo1()	Sub eloltesztelo2()
Dim i as integer	Dim i as integer
i = 1	i = 1
Do While i <= 20	Do Until i > 21
Cells(i, 9) = i	Cells(i, 9) = i
i = i + 1	i = i + 1
Loop	Loop
End Sub	End Sub
Sub hatultesztelo1()	Sub hatultesztelo2()
Dim i as integer	Dim i As Integer
i = 1	i = 1
Do	Do
Cells(i, 9) = i	Cells(i, 9) = i
i = i + 1	i = i + 1
Loop Until i > 20	Loop While i <= 21
End Sub	End Sub

5.2.3.4. Eljárások, függvények

A rutinfejből, deklarációs részből és a rutintörzsből felépülő rutinok kétféleképpen lehetnek: eljárások vagy függvények. A rutin típusa azonban meghatározza a rutinfej szintaktikai szabályait.

Eljárás rutinfejének és keretének szintaxisa:

```

Sub <eljárásnév> [( <paraméterlista> )]
    [<Deklarációs rész>]
    [<Eljárás törzs, azaz utasítás(ok)>]
[Exit Sub]
    [<Eljárás törzs, azaz utasítás(ok)>]
End Sub

```

Függvény rutinfejének és keretének szintaxisa:

Function <függvénynév> [(<paraméterlista>)] [**As** <típusnév>]

[<Deklarációs rész>]

[<Eljárás törzs, azaz utasítás(ok)>]

[<Függvénynév> = <kifejezés>]

[**Exit Function**]

[<Eljárás törzs, azaz utasítás(ok)>]

[<Függvénynév> = <kifejezés>]

End Function

Mindkét szintaktikából kiderül, hogy a rutinok neve után nem kötelező kitenni a () zárójelpárt, csak akkor, ha legalább egy paraméterű a rutin. A paraméterlista elhagyható, de ha szerepel, akkor meg kell adni a paraméterek nevét (és típusát) egymástól vesszővel elválasztva.

Továbbá látható, hogy eljárás és függvény között lényegi különbség van: míg az eljárás csak végrehajt egy utasítássorozatot, addig a függvény kiszámít valamilyen értéket és ezt visszaadja a hívó eljárásnak vagy egy másik függvénynek. Így a függvény törzsében mindig szerepelnie kell egy értékadó utasításnak, vagyis a függvény nevét, mint változót egyenlővé tesszük a visszatérési értékkel.

Paraméterek:

A paraméterlista általában:

[**Optional**] [**ByRef** | **ByVal**] <paraméternév>[()] [**As** <típusnév>]
[= <alapérték>]

A paraméterek átadása lehet cím szerinti vagy érték szerinti.

Alapértelmezésben **cím szerinti paraméterátadásról** beszélünk. Ilyenkor a rutinfejben lévő formális paraméter *azonossá válik* a hívó rutin aktuális paraméterével. Ez azt jelenti, hogy híváskor a formális paraméter *rácímeződik* az aktuális paraméterre. Tehát, ha a műveletvégzések során megváltozik a formális paraméter értéke, akkor az aktuális paraméter értéke is változni fog. Az eljárás fejében deklarált paramétert azért nevezzük formális paraméternek, mivel ezek a paraméterek egészen addig csak formálisan léteznek, míg az eljárást konkrétan meg nem hívjuk az aktuális paraméterekkel.

A cím szerinti paraméterátadás onnan ismerhető fel, hogy **ByRef** kulcsszóval kezdődhet, hiszen ennek kitétele nem kötelező.

Érték szerinti paraméterátadás esetén a formális paraméter megkapja az aktuális paraméter aktuális *értékét*, de *nem hat vissza* arra. Onnan ismerhető fel, hogy **ByVal** kulcsszóval kezdődhet. Az érték paraméterek az eljárásba való belépéskor megszületnek.

```
Sub SzamotKiir (szam As Integer)
    Cells(1,1) = szam
End Sub
```

```
Sub Kiiratas_meghivasa
    SzamotKiir(12)
End Sub
```

VAGY

```
Sub szamkiir(szam As Integer)
    Cells(1, 1) = szam
End Sub
```

```
Sub kiiratas_meghivasa()
    Dim kiirando As Integer
    kiirando = InputBox("Mennyit írjak ki?")
    szamkiir (kiirando)
End Sub
```

Feladat: A lineáris egyenlet megoldását kiszámító függvény elkészítése

Function megold(a **As Single**, b **As Single**, c **As Single**) **As Single**

megold = (c - b)/a

End Function

A függvény hívása és az eredmény kétszeresének kiírása:

```
Sub fuggveny_meghivasa()
    Dim a As Single, b As Single, c As Single
```

a= 1

b= 2

c= 3

Cells(1,1) = (megold(a, b, c) * 2)

End Sub

Ugyanazon modulba írjuk meg. Először a függvény definíciót, aztán az eljárást, ami meghívja a függvényt.

A paraméterátadás módja a Visual Basicben referencia szerinti paraméterátadás, azaz a formális paraméterek automatikusan az aktuális paraméterek címkomponensét kapják meg.

Feladat: A lineáris egyenletet megoldó *eljárás* elkészítése.

```
Sub megold(ByVal a As Single, ByVal b As Single, ByVal c As Single, ByRef eredmény As Single)
```

Az első három paraméter érték szerinti, a negyedik viszont cím szerinti paraméterátadás

```
    eredmény = (c - b) / a
```

```
End Sub
```

Az eljárást a következő módon hívhatjuk meg:

```
Sub eljárás_hivas()
```

```
    Dim x As Single, y As Single, z As Single
```

```
    Dim megoldas As Single
```

```
    x = 1
```

```
    y = 2
```

```
    z = 3
```

```
    Call megold(x, y, z, megoldas) VAGY megold x, y, z, megoldas
```

```
    Cells(1, 1) = (megoldas * 2)
```

```
End Sub
```

5.2.3.5. Rutinhívó utasítások

Az eljáráshívás paraméter nélküli esetben csak az eljárás névével, paraméteres esetben az eljárás névével és zárójelben az aktuális paraméterek felsorolásával történik.

Az eljárás vagy rutin olyan névvel ellátott programrész, mely egy adott tevékenységsorozatot fog össze. Általában valamilyen részfeladatot hivatottak megoldani. A hívás azt jelenti, hogy egy rutint aktiválunk, lefuttatunk. Az eljárás meghívása után a meghívott programrész utasításai hajtodnak végre, aztán a vezérlés visszakérül a meghívást *követő* utasításra.

A rutinok hívásának általános szintaxisa:

```
Call <eljárásnév> (<paraméterlista>)
```

vagy

```
<eljárásnév> <paraméterlista>
```

Függvények esetén pedig:

```
Változónév = Call függvénynév (paraméterlista)
```

Függvények esetén, ha nem vagyunk kíváncsiak az eredményre:

```
függvénynév paraméterlista
```

Az *eljárásokat* tehát egyszerűen a nevükre történő hivatkozással vagy a **Call** foglalt szóval lehet meghívni. Ha csak a névére való hivatkozással hívjuk meg az eljárást, akkor a paramétereit csak fel kell sorolni, ilyenkor a zárójeleket azonban tilos kitenni. Továbbá tilos kitenni a zárójelpárokat abban az esetben is, ha az eljárásnak nincs paramétere. Ha az eljárást

a Call foglalt szó segítségével hívjuk meg, akkor viszont a paraméterlistát kötelezően zárójelbe kell tenni.

Ha függvényeket kívánunk aktiválni, akkor a függvény nevének egy értékadó utasítás jobb oldalán, vagy egy kifejezés részeként kell szerepelnie. A paramétereket zárójelek között kell felsorolni, de ha a függvénynek nincs paramétere, akkor az üres zárójeleket nem kell kitenni. Ha egy függvényt Call foglalt szóval hívjuk meg, akkor a függvény eljárásként fog viselkedni, vagyis visszatérési értéke elveszik.

Példa eljáráshívásra:

Feladat, hívjuk meg az MsgBox() függvényt eljárásként!

```
MsgBox "Jó napot kívánok!"
```

```
Call MsgBox("Jó napot kívánok!")
```

Példa egy függvény használatára:

Feladat: Számítsuk ki egy tömb elemeinek átlagértékét függvény segítségével! A függvény tetszőleges számú tömbelem esetén is működjön.

```
Function atlag(tomb()) As Double
    Dim osszeg As Double, i, elemsz As Integer
    osszeg = 0
    elemsz = 0
    For Each i in tomb
        osszeg = osszeg + i
        elemsz = elemsz + 1
    Next
    atlag = osszeg / elemsz
End Function
```

```
Sub atlag_kiiratas()
    Dim t(1 To 4)
    t(1) = 1.5
    t(2) = 2.5
    t(3) = 3.5
    t(4) = 4.6
    Cells(1, 9) = atlag(t)
End Sub
```

5.2.3.6. Beépített Excel függvények

Fontosabb szövegkezelő függvények:

`Left(<szöveg> As String, <hossz> As Integer) As String`

A `Left` függvény visszatérési értéke a megadott szöveg elejéből *hossz* hosszú karaktersorozat (ha a szám nagyobb, mint a szöveg hossza, akkor a teljes szöveget kapjuk vissza; ha a szám 0, akkor üres szöveget kapunk vissza). `Left("1234567890", 3)` eredménye: 123

Példa:

```
Sub string_proba()
```

```
    Dim szoveg, eredmeny As String
```

```
    Dim hossz As Integer
```

```
    szoveg = "Zugonicsné"
```

```
    hossz = 3
```

```
    eredmeny = Left(szoveg, hossz)
```

```
    Cells(1, 1) = eredmeny
```

```
End Sub
```

Az eljárás eredményként az A1-es cellában megjeleníti a „Zug” karaktersorozatot.

`Right(<szöveg> As String, <hossz> As Integer) As String`

A `Right` függvény visszatérési értéke a megadott szöveg végéből *hossz* hosszú karaktersorozat (ha a szám nagyobb, mint a szöveg hossza, a teljes szöveget kapjuk vissza; ha a szám 0, akkor üres szöveget kapunk vissza). Példa: `Right("abcdefghijkl", 4)` eredménye: ijkl

`Mid(<szöveg> As String, <start> As Integer, <hossz> As Integer) As String`

A `Mid` függvény visszatérési értéke a megadott szöveg *start* pozíciójától *hossz* hosszú karaktersorozat. Példa: `Mid("abcdefghijkl", 2, 7)` eredménye: bcdefgh

`Len(<szöveg> As String) As Integer`

A `Len` függvény a paraméterként kapott szöveg hosszát adja vissza. Példa: `Len("abc")` értéke: 3

`InStr(<hol> As String, <mit> As String) As Integer`

Az `InStr` függvény a *hol* szövegben keresi a *mit* szöveget, visszatérési értéke a *mit* szöveg első előfordulásának az első karakterpozíciója. A függvény visszatérési értéke 0, ha nem fordul elő a keresett sztring, vagyis nincs találat. Példa: `InStr("abcdabcdabcd", "bc")` értéke: 2

InStr(<start> As Integer, <hol> As String, <mit> As String) As Integer

Az InStr függvény három paraméteres változatának működése hasonló a két paraméteres változathoz, a különbség az, hogy a *mit* szöveget csak a *start* pozíciótól kezdi keresni a *hol* szövegben. Példa: InStr(5, "abcdabcdabcd", "bc") értéke: 6

Asc(<szöveg> As String) As Integer

Az Asc függvény a paraméterként kapott szöveg első karakterének az ASCII (Windows) kódtábla szerinti kódját adja vissza. Asc("B") értéke: 66

Chr(<kód> As Integer) As String

Az Chr függvény a paraméterként kapott ASCII (Windows) kódtáblában levő kódhoz tartozó karaktert adja vissza szöveggént. Chr(67) értéke: C

Konverziós függvények: lásd a deklarációs résznél.

Fájlkezelő függvények:

Könyvtárkezelés

A könyvtárak kezeléséhez a könyvtárak létrehozása, törlése, átnevezése és a könyvtár tartalmának lekérdezése tartozik.

MkDir(<könyvtárnév> As String)

Létrehozza a „könyvtárnév” nevű könyvtárat. A könyvtárnév tartalmazhat elérési utat is.

Rmdir(<könyvtárnév> As String)

Törli a „könyvtárnév” nevű üres könyvtárat. Ha nem üres, hibaüzenetet kapunk.

Name <régi_Név> As <új_Név>

A *régi_Név* és az *új_Név* egyaránt szöveg paraméterek, amelyek egy fájl vagy könyvtár nevét tartalmazzák, elérési úttal együtt. A függvény a régi_Név nevű fájlt vagy könyvtárat átnevezi új_Név névre. Ha az új_Név másik könyvtárra mutat, akkor a fájlt átmozgatja.

Dir(<könyvtárnév> As String[, <attribútum> As Integer]) As String

Egy könyvtárban található fájllista lekérdezésére szolgáló függvény. A könyvtárnév tartalmazhat „*” és „?” helyettesítő karaktereket. Az *attribútum* segítségével lehet speciális fájllistát lekérdezni az adott könyvtárban. Az *attribútum* lehet 0: normál fájlok; 1: Read Only, azaz csak olvasható fájlok; 2: Hidden, azaz rejtett fájlok; 4: System, azaz a rendszerhez tartozó fájlok; 8: Volume, azaz az adott partíció neve; 16: Directory, azaz alkönyvtárak, vagy kiterjesztés nélküli fájlok. Ha elhagyjuk, az attribútum nélküli fájlok (0) listáját kapjuk. Visszatérési értéke a beolvasott listából az első elem neve, vagy üres szöveg, ha a könyvtár nem tartalmazza a fájlt, vagy további alkönyvtárakat.

Ezt követően paraméter nélkül hívva a Dir függvényt a függvény visszaadja a korábban (első paraméterezésével) beolvasott fájl, vagy könyvtárlista következő elemét. Ha a lista kiürült, azaz minden fájlnevet vagy könyvtárnevet visszaadott, üres szöveg a visszatérési érték.

Fájlkezelés

```
FileCopy(<mit> As String, <hova> As String)
```

Fájlok másolására használt függvény, amely első paraméterként egy fájl nevét (és elérési útját), második paraméterként pedig egy másik fájlnevet (és elérési útját) kap, ahová másolnia kell. Ha az elérési út azonos, a fájlt *hova* néven duplikálja.

```
Kill(<mit> As String)
```

A *mit* fájl, vagy fájlok törlésére szolgáló függvény. A *mit* tartalmazhatja az elérési utat. Ha több fájlt szeretnénk törölni, használhatjuk a „*”, „?” helyettesítő karaktereket.

```
FileExists(<mi> As String) As Bool
```

A függvény a *mi* fájl létezését vizsgálja. Ha létezik, igaz értéket ad vissza, ha nem, hamisat.

Feladat: Írjunk most egy olyan eljárást, amely bekér a felhasználótól egy könyvtárnevet, megnyitja azt, és MessageBox-ban megjeleníti a könyvtárban található fájlok neveit, mindegyiket új sorban.

```
Sub fajlLista()
    Dim tmp As String
    Dim tartalom As String
    tartalom= ""

    tmp= Dir(InputBox("könyvtár név:", "könyvtár tartalmának listázása"), 0)

    Do While tmp <> ""
        tartalom= tartalom & vbCrLf vagy Chr(13) & tmp
        tmp= Dir
    Loop

    MsgBox tartalom

End Sub
```

Eljárásunkban két változót hozunk létre, a *tmp* nevűt, amit arra használunk, hogy egyesével tároljuk benne a Dir függvény által visszaadott fájlneveket, míg a tartalom változóba a tmp értékeit fűzzük össze "új sor" karakterrel elválasztva. Az "új sor" vagy "sortörés" karaktert nem jeleníthetjük meg szövegben, de használhatunk beépített változót helyette, vagy ismerve a sortörés ASCII kódját (13), felhasználjuk a Chr függvény visszatérési értékét, amely a 13 paraméterrel éppen egy sortörést tartalmazó sztring lesz. A While ciklus feltétele a tmp változó értékére vonatkozik, ugyanis ha minden fájlt visszaadott már a Dir függvény, vagy a könyvtár nem létezik, a visszatérési értéke üres szöveg lesz.

Néhány beépített változó:

vbCR: Carriage Return, azaz kocsi vissza (írógépes időkből származó) vezérlőkarakter

vbLF: Line Feed, azaz soremelés vezérlőkarakter

vbCrLf: Az előző két vezérlőkód együtt. A DOS és Windows rendszer ezt használja sortörésként.

vbTab: Tabulátor karakter. Az utána levő szöveg egy tabulátor pozícióval beljebb fog kezdődni

5.2.3.7. Üzenetküldés és válaszkérés VBA nyelven

A VBA széles lehetőséget biztosít interaktív programok elkészítésére, ahol is üzenetet küldünk a felhasználóknak és választ is várunk (pl. különböző nyomógombok segítségével), vagy billentyűzetről olvasunk be értékeket.

A felhasználókat az aktív munkalapon az **MsgBox** utasítással szólíthatjuk meg. Az, hogy eljárásként vagy függvényként hívjuk meg, attól függ, hogy csak üzenetet kívánunk küldeni, vagy választ is várunk a felhasználótól. Az utasítás hatására egy üzenetablak jelenik meg a képernyőn.

Általános szintaxisa:

Legegyszerűbb (üzenetküldésnek):



```
MsgBox <üzenet>
```

Üzenetküldés és válasz:

```
valasz = MsgBox (<üzenet> [, gomb konstans] [, <címsor>] [, <súgó  
állomány>, <súgó index>])
```

Az első paraméterként megadott üzenet a megjelenítendő üzenet szövegét tartalmazza. A *gomb* konstans paraméter értékétől függ, hogy milyen parancsgombok lesznek láthatóak az ablakban. Alapértelmezettként az OK gomb, tehát ha nem adunk meg más gombkonstanst, akkor csak ez a gomb lesz látható.

Néhány nevezetes gombkonstans:

- | | |
|----------------------|--|
| – vbOKOnly | – Csak az OK gomb látszik. |
| – vbOKCancel | – OK és Mégse gomb látszik. |
| – vbYesNoCancel | – Igen, Nem és a Mégse gomb látszik. |
| – vbQuestion | –  ikon is látszik. |
| – vbCritical | –  ikon is látszik. |
| – vbApplicationModal | – Alkalmazáshoz kötött: a makró futása felfüggesztődik mindaddig, míg a felhasználó nem válaszol. |

A címsor paraméter az üzenetablak fejléce lesz. Ha hiányzik, akkor a „Microsoft Excel” felirat fog látszódni.

Az utolsó paraméter a súgóállomány nevét, valamint a téma állománybeli azonosítóját (indexét) tartalmazza. Ez biztosítja, hogy ha a Súgó ikonra kattintunk az ablakban, akkor a súgóállomány megfelelő indexe nyíljon meg.

Szöveges adatokat bekérni az **InputBox** függvény segítségével lehet. Szintén üzenetablakot jelenít meg a képernyőn, de a rutin futása közben addig várakozik, amíg a felhasználó egy választógombra nem kattint.

Általános szintaxisa:

```
válasz = InputBox (<üzenet> [, <címsor>, [<alapérték>] [, X] [, Y] [, <súgó állomány>, <súgó index>])
```

Az üzenet, a címsor, a súgó állomány és súgó index paraméterek jelentése megegyezik az MsgBox-nál már említettekkel. A harmadik paraméter a beviteli mezőben megjelenő alapértéket tartalmazza (ami megjelenik a bekérő-mezőben, de szabadon átírható), az x és y paraméterrel pedig pozícionálni lehet az üzenetablak bal felső sarkát.

Feladat: Készítsünk makrót, amely InputBox-ban bekéri a felhasználó nevét, majd MsgBox-ban köszönti a felhasználót a saját nevén.

```
Sub koszonto()  
    Dim nev As String  
    Do  
        nev = InputBox("Hogy hívnak?", "Kérdés")  
    Loop Until nev <> Empty           → amíg a nev változó nem kap értéket.  
    MsgBox "Szia " & nev & " !"   
End Sub
```

5.3. Gyakorló feladat

A deklarációval, a makrók felépítésével és a legegyszerűbb utasításokkal kapcsolatos alapvető ismeretek elsajátítása után nézzünk egy nagyon egyszerű példát.

Feladat: Másodfokú, egyismeretlenes egyenlet megoldása során döntsük el a diszkrimináns alapján, hogy az egyenletnek hány gyöke van. Az elsőfokú tag együtthatóját, a szabad tagot és ezek előjelét véletlenszám generálással állítsuk elő.

```
Private Sub CommandButton1_Click()
    Randomize
    Dim a As Double, b As Double, c As Double
    Dim d As Double, X1 As Double, X2 As Double
    a = 1
    b = CInt(10 * Rnd()) * (1 - 2 * CInt(Rnd()))
    c = CInt(10 * Rnd()) * (1 - 2 * CInt(Rnd()))
    d = b ^ 2 - 4 * a * c
    If d < 0 Then
        MsgBox "Nincs valós gyök", , "Másodfokú egyenlet"
    ElseIf d > 0 Then
        X1 = (-b + Sqr(d)) / (2 * a)
        X2 = (-b - Sqr(d)) / 2 / a
        MsgBox "X1 = " & X1 & vbCrLf & "X2 = " & X2, , "Másodfokú egyenlet"
    Else
        X1 = -b / (2 * a)
        MsgBox "X = " & X1, , "Másodfokú egyenlet"
    End If
End Sub
```

Rutinfej

Deklarációs rész: változók deklarálása

Értékadó utasítások

Véletlenszám generálás

Diszkrimináns meghatározása

Elágaztatással gyökök számának meghatározása, képernyőre íratása egy üzenőablakban.

5.4. A programozott makrók indítása

Ebben a fejezetben a makrók programozásának alapjait már elsajátítottuk. Arra tehát már képesek lehetünk, hogy egyszerűbb makrókat elkészítsünk és ezeket a fejlesztői környezetben lefuttassuk, de ez nem jelenti azt, hogy egy átlagos Office felhasználó is el tudná indítani a makrókat. Most ezt nézzük meg.

Az általunk elkészített vagy felvett makrókat négyféleképpen indíthatjuk el:

1. **Billentyűkombinációval:** A *Ctrl+hozzárendelt billentyű* lenyomásával.
2. **Makró párbeszédablakon keresztül:** *Fejlesztőeszközök szalag/Makrók...* Ugyanezt az ablakot tudjuk megnyitni a *Nézet menü/Makrók/makró megjelenítése* paranccsal is. Az adott ablakban kiválasztjuk a futtatni kívánt makró nevét és az indítás gombra kattintva a makró folyamatosan lefut. A makrónkat ennek az ablaknak a segítségével azonban lefuttathatjuk lépésenként is, de mód nyílik itt a törlésre, szerkesztésre is.
3. **Ikonnal történő indítás:** Az ikonos indítás lényege, hogy egyik eszköztárunkba új ikont helyezünk el, melyhez hozzárendeljük a futtatandó makrónkat. Ennek a menete a következő:
 - A kívánt eszköztáron jobb gomb, majd válasszuk a *menüszalag testreszabása* almenüpontot. A megjelenő ablakban válasszuk ki a *Makrók* feliratot és vegyük fel a kívánt menüsorra vagy hozzunk létre neki egy új lapot.
 - Ha kész az új lap, készítsünk azon belül egy új csoportot. Alul az átnevezés gombra kattintva választhatunk neki egy nekünk tetsző ikont is.
4. **Parancsgommbal:** A gyakorlatban leginkább használt megoldás. A megoldás lényege, hogy a munkalapon egy parancsgombot (CommandButton) helyezünk el, amire kattintva indítható az általunk készített makró.

Ennek menete a következő:

- Jelenítsük meg először a lehetséges vezérlőeszközök készletét a *Fejlesztőeszközök/Beszúrás/ÚrlapVezérlőelemek* menü segítségével.
- Váltunk át *Tervező módba*, majd kattintsunk a parancsgomb ikonjára az eszközkészletben és húzzuk az egeret arra a helyre, ahol a parancsgombot el szeretnénk helyezni. Tartsuk lenyomva az egér bal gombját és egy átlós mozdulattal rajzoljuk ki a gombot.
- Ha felengedjük a bal gombot, megjelenik a parancsgomb. Ha előhívjuk a helyi menüjét (a vezérlőn jobb gombbal kattintunk) a gombnak (*Kód megjelenítése* menüpont), vagy duplán kattintunk a gombon, akkor a kód lapon egy üres eljárás születik:

```
Private Sub CommandButtonX_Click()  
...  
End Sub
```

- A makrót a fej és a záró rész között kell megírni, vagy ha készen van, akkor ide kell bemásolni. A Private kulcsszó pedig akár törölhető is.

5.5. Összetett példa a fejezetben tanultak elsajátítására

Feladat1: Összegezzünk két, a felhasználó által megadott számot (Inputbox), majd írassuk ki az eredményt MsgBox-al.

Megoldás:

```
Function osszegzes(ByVal a As Integer, ByVal b As Integer) As Integer
```

```
    osszegzes = a + b
```

```
End Function
```

```
Sub bekeres()
```

```
    Dim c, d, ossz As Integer
```

```
    c = InputBox("Kérem az első számot!")
```

```
    d = InputBox("Kérem a második számot!")
```

```
    ossz = osszegzes(c, d)
```

```
    MsgBox c & " + " & d & " = " & ossz, , "Két szám összege"
```

```
End Sub
```

6. Objektumok Visual Basic-ben (vezérlőelemek, formok használata)

6.1. Az objektumorientált programozás alapjai

Az eljárás-orientált programozás lényege, hogy algoritmusainkat egyszerű, viszonylag kevés műveletet végrehajtó, függvények/eljárások formájában valósítjuk meg. Az adatokat változókból tároljuk és a megfelelően paraméterezett alprogramok reprezentálják a modellezett adatok viselkedését. Az eljárás-orientált megközelítés azonban csupán kisebb feladatok megoldására alkalmazható, azonban nagyobb problémák kezelésére a rengeteg függvény/ eljárás és azok paraméterezése nehezen kezelhetővé/javíthatóvá/karbantarthatóvá teszi a kódot a felhasználók számára. Az objektum-orientált (OO) programozási paradigma ezen "rendezetlenségek" kezelésére jött létre. Az objektumorientált programozás során olyan *saját típusokat* hozhatunk létre, amelyek a korábbi rekordokkal ellentétben nem csak adattagokat, de függvényeket/eljárásokat is tartalmazhatnak. A típushoz kötött alprogramoknak köszönhetően (melyeket metódusoknak nevezünk), nem kell paraméterként átadni az adatokat reprezentáló változót ahhoz, hogy annak valamely viselkedésmintáját alkalmazzuk, így nem követhetünk el paraméterezésből származó hibákat. Azokat a *saját típusokat*, amelyek *metódusokat* is tartalmaznak, *osztályoknak* hívjuk, egy ilyen típusú változót pedig *objektumnak*.

A VBA nyelv bizonyos korlátozásokkal ugyan, de az objektumorientált programnyelvek csoportjába tartozik. A legfőbb korlátozás, hogy objektumokat nem hozhatunk szabadon létre, így csak Excel-objektumokat használhatunk, továbbá, míg az objektumorientált programnyelvekben az objektumokból újabb objektumok hozhatók létre, melyek öröklik szüleik tulajdonságait, addig a VBA nyelvben nincs erre lehetőség.

Az előző mondatokból kikövetkeztethető tehát, hogy a makróink alapvető eleme is az objektum lesz. Az **objektum** a valós világ egy jól körülhatárolható részének modellje. Az objektumok szerves részét alkotják az objektumot meghatározó tulajdonságaik és metódusaik. VBA nyelvi szempontból objektum lehet egy utasítás, egy eljárás, egy függvény, vagy maga a makró. A **tulajdonságok** az objektumok megjelenési formái. Egy adatbeviteli mező esetén tulajdonság például a betű mérete és színe. A tulajdonságokat **metódusokkal** lehet módosítani, lekérdezni. Metódus például egy objektum értékének törlése, másolása, újraszámolása.

Fontos megkülönböztetni az objektumok *osztályát* az *objektumegyedtől*. Az egyed ugyanis az osztály egy konkrét *előfordulási eleme*. Osztály például egy szabásminta, egyed például egy 42-es méretre kiszabott ruhadarab.

Ahogy a tankönyv elején is említettem már, a Visual Basic nyelv objektumokra épül. Az *objektumok* adatokat, illetve az adatokon műveletek végzésére használható programkódot tartalmaznak. Az *objektumpéldányok* létrehozásánál használt *adattípusokat osztályoknak* nevezzük. A legtöbb objektumorientált programnyelv támogatja, hogy osztályokat, azon belül objektumokat származtassunk egymásból, melyek *öröklik* elődeik tulajdonságait. A VBA

azonban ezt az egy lehetőséget nem teszi lehetővé, ezért nevezzük objektum-alapú nyelvnek csupán.

Visual Basic-ben az objektumok programozott kialakítására többféle eszköz (form modul, osztálymodul, vezérlőmodul) áll rendelkezésünkre. Ezen eszközök közül fogunk most megismerkedni a form modullal, mely a vezérlőelemeket tartalmazza.

6.2. Form module

6.2.1. Vezérlőelemek

Vezérlőknek nevezzük azokat az építőelemeket, melyeket űrlapokon (formokon) elhelyezve áll elő az alkalmazásunk felhasználói felülete.

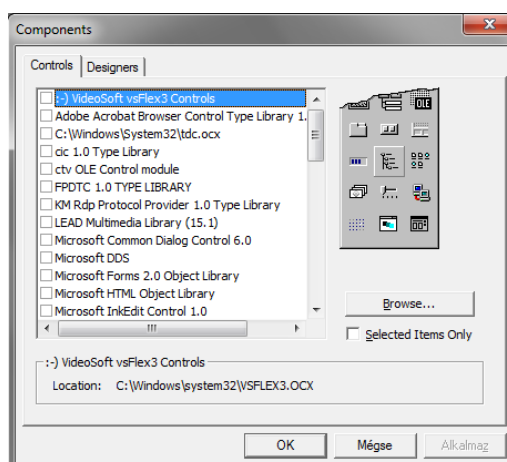
A vezérlőelemeket két nagy csoportba oszthatjuk: alapvezérlők és ActiveX-vezérlők.



8. Ábra Alapvezérlők eszközkészlete

Az alapvezérlők megkönnyítik a felhasználói adatok kezelését és nagyfokú interaktivitást biztosítanak.

Az ActiveX-komponensek olyan 32 bites vezérlőelemek, melyek a Windows alatt vannak regisztrálva, de kapcsolatba hozhatók a Visual Basic-el. Ezen kapcsolat úgy hozható létre, ha VB nézetben a kódlapon <Ctrl+T> billentyűkombinációt leütjük és a megjelenő párbeszédablakban a Control fülön kijelöljük azt a vezérlőt, amelyet be szeretnénk illeszteni az alkalmazásunkba.

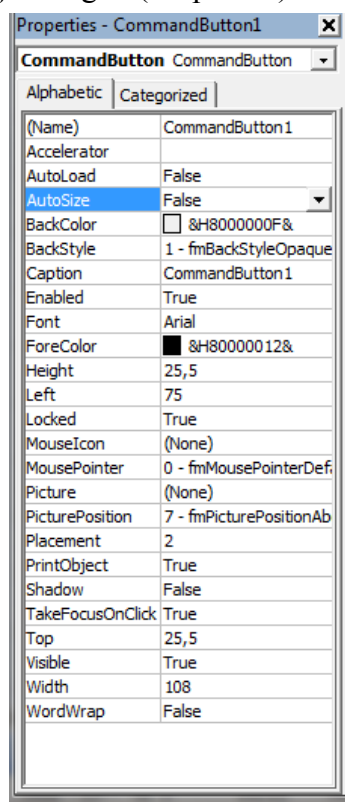


9. ábra ActiveX vezérlők megjelenítése

Jelen könyvben azonban csak az alapvezérlőkkel ismerkedünk meg részletesen. Azt gondolom, hogy azoknak, akik most kezdtek el a makrókkal ismerkedni, ez is kellően nagy kihívás.

Néhány szó a vezérlőkről általánosságban

A vezérlők alapvetően különböznek egymástól, mivel más-más céllal jöttek létre. Rendelkeznek azonban 28-50 tulajdonsággal (ezen tulajdonságokat a Properties – Tulajdonságok ablakban tekinthetjük meg), melyek között rengeteg azonosság fedezhető fel. A VBA nyelvben minden objektum létrejöttékor kap egy hivatkozási nevet (Name), amely név egy egyedi sorszámot is tartalmaz. Továbbá általában minden objektum rendelkezhet egy felirattal is (Caption). Ezek jelentősége, hogy amikor az objektumot azonosítani szeretnénk, akkor a hivatkozási nevét használjuk, abban az esetben viszont, ha az objektumot meg szeretnénk jeleníteni, akkor a feliratát látjuk. Mindkét név azonban tetszőlegesen változtatható a tulajdonságok (Properties) ablakban.



10. Ábra Vezérlőelemek tulajdonságai

A Name (név) Mindig betűvel kezdődik és maximum 40 karakter hosszú lehet. Amikor egy vezérlőt elhelyezünk a formunkon, a Visual Basic automatikusan ellátja egy névvel, amely a komponens nevét és sorszámát tartalmazza. (Pl.: Label1)

A vezérlőelemek elhelyezkedését azon téglalap bal felső sarkának koordinátaival (**Top, Left**), szélességével (**Width**) és magasságával (**Height**) állapítjuk meg, amely a vezérlőelemet képes magába foglalni.

A vezérlőelemek lehetnek láthatóak, vagy rejtettek. Ezt a tulajdonságukat a **Visible** tulajdonság értéke (True vagy False) dönti el.

A vezérlők azon tulajdonságát is mi döntjük el, hogy működőképesek legyenek vagy sem. Az **Enabled** tulajdonság engedélyezi vagy tiltja a vezérlő működését. A letiltott vezérlő látható, de színe szürke és felhasználói beavatkozásra nem reagál.

A **Font** tulajdonságérték szolgál arra, ha a vezérlőelemen megjelenő feliratok betűtípusát, méretét, stb. meg szeretnénk változtatni.

Ezen tulajdonságokra való hivatkozásoknál a kódban az objektum nevéhez ponttal kapcsoljuk hozzá a tulajdonság nevét. Pl.: `TextBox1.Font.Size = 12`

6.2.2. Vezérlőelemek elhelyezése a párbeszédlapon

1. Kattintsunk a megfelelő vezérlőelem ikonjára az eszközkészleten az egér bal gombjával.
2. Az egérkurzort mozgassuk az űrlap fölé olyan helyre, ahol a vezérlőt el szeretnénk helyezni. Keressük meg azt a pontot, mely a vezérlőelemet befoglaló téglalap bal felső sarka lesz.
3. Nyomjuk le az egér bal gombját az adott helyen és tartsuk lenyomva. Majd húzzuk el átlósan a befoglaló téglalap jobb alsó sarkáig. (Ha csak kattintunk, akkor egy alapértelmezett méretű vezérlőt tesz ki a program, de a méretét tetszőlegesen utólag is megváltoztathatjuk)
4. Engedjük fel az egér bal gombját.

Ha a vezérlőelem ikonjára kétszer kattintunk, akkor ugyanazt a vezérlőelemet többször is megrajzolhatjuk egymás után.

6.2.3. Vezérlőelemek részletes ismertetése

6.2.3.1. Beviteli mező (TextBox)



Kézi adatbevitelre, egyszerű szövegszerkesztési feladatok ellátására szolgáló vezérlőelem. Alapértelmezésben Szöveg (String) típusú és egysoros, így ha a beírandó szöveg ennél hosszabb, akkor a szöveg vége nem látszik. A mező azonban átalakítható többsorosra, abban az esetben, ha a **MultiLine** tulajdonságértéket *True*-ra állítjuk. A szöveg hosszának automatikus tördelését a szövegmező nagyságához a **WordWrap** tulajdonság *True* értékűre változtatásával oldhatjuk meg.

A szövegmező legfontosabb tulajdonságai:

- **Text:** a felhasználó által bevitt szöveget tartalmazza
- **MultiLine:** A szöveg beviteli mező többsorosra alakítható.
- **Locked:** *False* értéke esetén a szöveg szerkeszthető, *True* érték esetén csak olvasható.
- **MaxLength:** A beírt szöveg hossza korlátozható le. *0* érték esetén nem korlátozzuk, ha egyéb szám szerepel a mezőben, akkor ez a szám jelenti a bevihető karakterek maximális számát.
- **Alignment:** A MultiLine *True* értéke esetén van értelme, mivel ezzel a tulajdonsággal az állítható be, hogy a szöveg balra(1), jobbra(3) vagy középre(2) igazítva jelenjen meg.
- **ScrollBar:** Többsoros szövegmezőben engedélyezhetjük vízszintes illetve függőleges görgetősávok megjelenését. Lehetséges értékei: *0* – Nincs görgetősáv; *1* – csak vízszintes görgetősáv van; *2* – csak függőleges görgetősáv van; *3* – vízszintes és függőleges görgetősáv is van. Tervező módban a hatása nem látszik.
- **PasswordChar:** Ha ez a mező nem üres, akkor a vezérlőben megjelenő szöveg azzal a karakterrel jelenik meg, amit ebbe a mezőbe írunk pl. *** esetén *******-t látunk. Jelszó begépelésére alkalmas.

6.2.3.2. Címke, Felirat (Label)



Feliratok elhelyezésére szolgáló szöveges mező. Tartalmát csak tervező módban lehet begépelni.

A címkék legfontosabb tulajdonságai:

- **Caption:** A címke szövegét tartalmazza.
- **BackStyle:** Ha értéke 1 (Opaque), akkor a címke nem átlátszó, ha 0 (Transparent), akkor átlátszó.
- **BorderStyle:** Beállítható, hogy legyen-e kerete a címkének vagy sem. 0 – nincs keret; 1 – van keret.
- **Font:** a felirat betűtípusát, stílusát, méretét lehet beállítani.
- **Alignment:** A szöveg igazítása a címkén. 0 - balra; 1 – jobbra; 2 – középre
- **AutoSize:** True értéke esetén a címke a szöveg méretéhez igazodik.

6.2.3.3. Parancsgomb (CommandButton)



A parancsgomb olyan vezérlő, mely a nyomógombok működését szimulálja. A parancsgomb megnyomásakor végrehajtandó tevékenységeket a kattintás (Click) eseményét feldolgozó eseménykezelő eljárásban kell elhelyezni.

A parancsgombok legfontosabb tulajdonságai:

- **Name:** az objektum hivatkozási neve, azonosítója
- **Caption:** Az objektumon látható felirat
- **Font:** A felirat betűtípusa, stílusa és mérete
- **BackColor, ForeColor:** Háttérszín, betűszín
- **Top, Left, Height, Width:** a bal felső sarokpont koordinátái, az objektum magassága és szélessége
- **Visible:** az objektum rejtett (false) vagy látható
- **Enabled:** az objektum aktiválható, használható vagy sem
- **AutoSize:** Automatikus méretezés a nyomógombon lévő felirathoz

6.2.3.4. Listamező (ListBox)



A listavezérlők numerikus vagy szöveges típusú elemeket képesek tárolni és megjeleníteni. A tételek sorok állhatnak egy sorból vagy több sorból. A listába kézzel nem lehet adatot felvinni. A listamezők bármely eleme kijelölhető, törölhető, görgethető és új elemek felvitele is lehetséges.

A listavezérlő legfontosabb tulajdonságai:

- **BoundColumn:** Az aktív oszlop sorszáma. Ha értéke > 0 , akkor a Value tulajdonság az aktív oszlopból kapja értékét.

- **Value:** Ha a BoundColumn értéke 0, akkor a kijelölt sor sorszáma, különben egyoszlopos lista esetén a kijelölt tétel, többoszlopos lista esetén a BoundColumn tulajdonsággal meghatározott oszlop kijelölt sorában lévő tétel.
- **Text:** A szöveges oszlop kijelölt eleme
- **ListCount:** Lista sorainak darabszáma
- **ColumnCount:** A lista oszlopainak száma
- **ColumnWidths:** Többoszlopos lista esetén az oszlopok szélessége pontokban (twip)
- **MultiSelect:** beállítható, hogy egy vagy egynél több elem is kijelölhető legyen a listából.
- **ListRows:** A párbeszédpanelen a listamezőben egyidejűleg látható sorok maximális száma.

6.2.3.5. Kombinált Lista (ComboBox)



A beviteli mező és a listamező kombinálása. Egy szövegmező és egy lista összeépítésével keletkezett. Futási időben is adhatunk új tételt a listához az **AddItem** módszerrel vagy törölhetünk a **RemoveItem** módszerrel.

A lista működését a **Style** tulajdonsága szabja meg:

- **0:** Egyszerű kombinált listának is nevezik. A szövegmező ablak tartalma módosítható. Hátránya azonban, hogy rögzített listamérettel rendelkezik, és nem lehet becsukni.
- **1:** Legördülő kombinált listának nevezzük. A listaablak nem jelenik meg automatikusan, de megjeleníthető a szövegmező oldalán található lenyíló nyomógomb segítségével.
- **2:** Lenyíló listának nevezzük. A szövegmező tartalma csak listaelem választásával módosítható. Akkor alkalmazzuk, ha azt szeretnénk, hogy a felhasználó az általunk felkínált elemek közül tudjon csak választani.

6.2.3.6. Kijelölőnégyzet (CheckBox)



Kétállapotú vezérlőelem, melynek értékei: be – x1on; kikapcsolt – x1off. Kikapcsolt állapotban üres négyzetnek látszik, míg bekapcsolt állapotban egy pipa látható benne.

A kijelölőnégyzet fontosabb tulajdonságai:

- **Alignment:** A négyzet és a mellette lévő szöveg egymáshoz képesti elhelyezkedését határozza meg. Alapértelmezettként (0) a négyzet a szöveg bal oldalán helyezkedik el. Az értéket 1-re állítva, kettőjük helyzete megfordul.
- **Value:** 0 - a négyzet jelöletlen
 - 1 - a négyzet jelölt
 - 2 - a négyzet szürke, inaktív

6.2.3.7. Választógomb (OptionButton)



Kétállapotú vezérlő, melynek értékei: be – xlon; kikapcsolt – xloff. Kikapcsolt állapotban üres körnek látszik, míg bekapcsolt állapotban egy pont látható benne.

A választógomb tulajdonsága:

- **Alignment:** A gomb és a mellette lévő szöveg egymáshoz képesti elhelyezkedését határozza meg. Alapértelmezettként (0), vagyis a gomb a szövegtől balra helyezkedik el. Az értéket 1-re állítva, kettőjük helyzete megfordul.

6.2.3.8. Kép (Image)



Segítségével dekoratív célú képek, rajzok, ábrák jeleníthetők meg. A grafikus objektumokhoz úgy rendelhetünk makrókat, hogy az elemre kattintva elindul a makró. Minden vezérlőelemhez kapcsolható kép a *Picture* tulajdonságával. Ez a lehetőség teremti meg például azt a helyzetet, hogy egy parancsgomb valamilyen képként látszódjon. A képeknek az objektumokhoz viszonyított helyzetét 3 szempont befolyásolja. Ezek:

- **PictureSizeMode:** a kép az objektum alakjához illeszkedik.
- **PictureAlignment:** ha a kép kisebb, mint az objektum, akkor hol helyezkedjen el a kép az objektum felületén
- **PicturePosition:** a kép hol helyezkedjen el a felirathoz képest.

6.2.3.9. Görgetősáv (ScrollBar)



A görgetősávban egy téglalap alakú csúszka található, melyet az egér segítségével háromféleképpen mozgathatunk. Vagy a le- és felfelé mutató nyilakra kattintva soronként (kis lépéssel) tudjuk mozgatni az információkat. Ha a két nyíl közé kattintunk, akkor a csúszka nagy elmozdulását tapasztalhatjuk. A harmadik mozgatási lehetőség pedig, amikor magát a csúszkát húzogatjuk az egér bal gombjának lenyomva tartásával.

Automatikus orientációnál függőleges pozicionáláshoz a vezérlő legyen magasabb, mint szélesebb. Vízszintes pozicionáláshoz a vezérlő legyen szélesebb, mint magasabb.

A görgetősáv fontosabb tulajdonságai:

- **Min:** a görgetősáv kezdeti pozíciója.
- **Max:** a görgetősáv végpozíciójához tartozó érték.
- **SmallChange:** kis lépésnagyságnál (nyilakra kattintás) ennyivel változik a vezérlő értéke
- **LargeChange:** nagy lépésnagyságnál (csúszka mellé kattintásnál) ennyivel változik a vezérlő értéke
- **Value:** A csúszka aktuális pozícióját tárolja.
- **Orientation:** a tulajdonság beállítja, hogy a vezérlő vízszintes vagy függőleges elhelyezkedésű legyen. (-1: automatikus, 0: függőleges, 1: vízszintes)
- **Change** esemény: A csúszka pozíciójának megváltozását jelzi.

6.2.3.10. Léptetőgomb (SpinButton)



A léptetőnyilakra való kattintással a nyíl szerinti irányban a SmallChange tulajdonságnak megfelelően (alapértelmezésben 1) növeli vagy csökkenti a vezérlő értékét.

Automatikus orientációnál függőleges alakúhoz a vezérlő legyen magasabb, mint szélesebb. Vízszintes alakúhoz a vezérlő legyen szélesebb, mint magasabb.

A léptetőgomb fontosabb tulajdonságai:

- **Orientation:** a tulajdonság beállítja, hogy a vezérlő vízszintes vagy függőleges elhelyezkedésű legyen. (-1: automatikus, 0: függőleges, 1: vízszintes)
- **Min, Max:** a léptetett érték határainak beállítására szolgálnak.
- **Value:** A léptetőgomb aktuális értékét tárolja.
- **Change** esemény: A léptetőgomb értékének megváltozását jelzi.

6.2.4. Párbeszédlapok

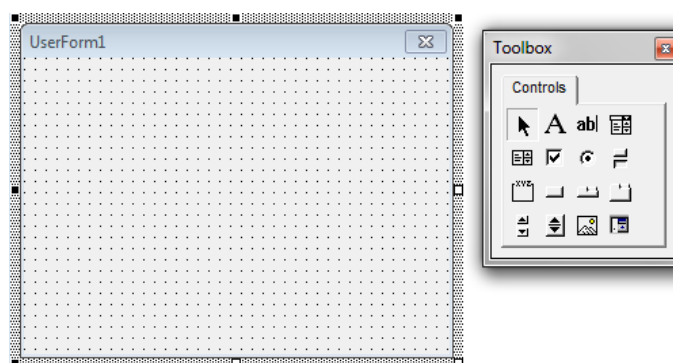
Ha Visual Basic-ben programozunk, akkor projekteket alkotunk. A projekt azon állományok halmaza, melyek szükségesek az alkalmazás létrehozásához. A futtatható projektek általában háromféle modult tartalmaznak:

- **Form modul**, mely az űrlappal együtt jön létre. Tartalmazza az űrlap és az űrlapon elhelyezett vezérlők tulajdonságait, a hozzájuk kapcsolódó eseménykezelő eljárásokat.
- **Osztálymodul**, ide olyan általános célú objektumok tartoznak, melyekhez nem tartozik grafikus felhasználói felület.
- **Modul**, a projektünk globális változóit, konstansait tartalmazza.

Ebben a fejezetben a form modullal és a hozzá kapcsolódó űrlappal ismerkedünk meg. A formra (űrlapra) különböző, elsősorban I/O célokat szolgáló objektumok példányait, vezérlőelemeket helyezünk el. A Form objektum kitüntetett szereppel rendelkezik az alkalmazások létrehozása és futtatása tekintetében. Megjelenítve ugyanis a formot - Show() - a felhasználói műveleteket fogadó párbeszédablak lesz belőle.

6.2.4.1. Párbeszédlap létrehozása

- Váltunk át VB nézetbe.
- Hozzunk létre egy új párbeszédlapot a *Insert/UserForm* menü segítségével. Ekkor megjelenik a vezérlőelemek eszköztára is.



11. ábra Form és eszköztára tervező nézetben

- Kapcsoljuk be a tervező módot a megfelelő ikonnal és ezután helyezzük el a vezérlőelemeket a formon a már megszokott módon (átlós mozdulattal az egér bal gombját lenyomva tartva).

6.2.4.2. Párbeszédlap tulajdonságai

A párbeszédlap is egy objektum, mely 34 féle tulajdonsággal rendelkezik.

A párbeszédlap fontosabb tulajdonságai:

- **Position:** a form ablak méreteit (**width**, **height**) és elhelyezkedését (**left**, **top**) adja meg.
- **Appearance:** Ebbe a kategóriába tartozó tulajdonságok az ablak megjelenési tulajdonságait szabályozzák.
 - **Caption:** mi legyen a fejléc szövege
 - **BackColor:** háttér színe
 - **ForeColor:** előtér színe
 - **FillColor:** kifestés színe
 - **BorderColor:** keret színe
- **Visible:** az ablak látható
- **Font:** A betűk méretét, stílusát meghatározó tulajdonság
- **Enabled:** A párbeszédlap engedélyezett-e, vagy sem.

6.2.4.3. A Form objektum metódusai

A metódusok az objektumokhoz tartozó nyilvános alprogramokat jelölnék. Egyaránt lehetnek eljárások vagy függvények.

A form metódusainak többsége az ablakban való megjelenítést, az ablakkal való műveleteket támogatja, ezek közül a legfontosabbak a következők:

- **Hide:** formablak elrejtése
- **Show:** formablak megjelenítése

6.2.4.4. Form megjelenítése, elrejtése

Egy párbeszédlap (hiába is hoztuk létre, szerkesztettük meg) csak akkor lesz látható a felhasználók számára, ha betöltjük a főtárba és megjelenítjük. A megjelenítésnél szem előtt kell tartani, hogy az alapértelmezett méretek szerint a form nem tölti ki a teljes képernyőt. Tehát vagy át kell méreteznünk majd, vagy ha csak kisebb felhasználói felületre van szükségünk, akkor célszerű előzőleg megnyitni egy üres munkalapot, melyen kikapcsoljuk a cellarácsokat és ez a fehér felület alkotja majd a háttérünket. Természetesen ha nem tetszik a fehér háttérszín, átszínezhető mind a form, mind a munkalap háttérszíne.

A párbeszédlap elindítása:

A háttérül szolgáló munkalapon elhelyezünk egy parancsgombot, melyre kattintva megjelenik a form, a form kódlapján pedig elhelyezünk egy kezdőrutint (*UserForm_Initialize*), melynek köszönhetően automatikusan elindul a párbeszédlap.

LOAD UTASÍTÁS, SHOW METÓDUS, HIDE METÓDUS

A Load utasítás az, mely a kiindulási adatokkal együtt betölti a főtárba a paraméterként meghatározott párbeszédlapot és automatikusan elindítja a *UserForm_Initialize()* nevű eljárást. A felhasználó számára azonban ekkor még nem látható. A felhasználó számára is látható formában való megjelenítést a *Show* metódussal érhetjük el.

Ahhoz, hogy ez a párbeszédlap eltűnjön a képernyőről, ahhoz vagy egy bezáró gombot kell elhelyeznünk a párbeszédlapon, vagy bizonyos utasítások bekövetkezése utánra be kell építeni az adott eljárásba az automatikus bezárást. A form képernyőről való eltüntetését a *Hide* metódussal oldhatjuk meg. Ennek szintaktikája: *párbeszédlap.Hide* lesz. Ha azonban végleg törölni szeretnénk a főtárból a párbeszédlapot, akkor használjuk az *Unload* utasítást. Ennek szintaktikája: *Unload párbeszédlap*. Ez azt jelenti, hogy többé nem is hivatkozhatunk a párbeszédlapra.

Példa: Jelenítsünk meg a felhasználó számára is látható módon, egy *proba* nevű formot a *háttér* nevű munkalapon, az eddig használt munkafüzetünkben.

Megoldás:

Hozzuk létre a form beszúrás parancssal a *proba* nevű formot! Majd készítsünk egy *hatter* nevű új munkalapot.

Gépeljük a ThisWorkbook – ba a következőket (ez az eljárás akkor fut le, amikor megnyitjuk a munkafüzetet):

```
Option Explicit
```

```
Sub Workbook_open()
```

```
    Workbooks(ActiveWorkbook.Name).Activate
```

```
    ActiveWorkbook.Sheets("hatter").Activate
```

```
    Load PROBA
```

```
    PROBA.Show
```

```
End Sub
```

Az Option Explicit

utasítással kötelezővé

tesszük a programunkban a
változók deklarálását.

Amennyiben ezt nem

tesszük meg, fordító

minden egyes nem

deklarált változó

észlelésekor hibaüzenetet

ad

6.2.4.5. Felhasználói beavatkozások a formon

Abban az esetben, ha a formon valamilyen adatbevitelre, vagy választásra alkalmas vezérlők vannak, gondoskodnunk kell arról, hogy a felhasználók csak bizonyos műveleteket hajthassanak végre. A felhasználói beavatkozásokat az *Enabled* tulajdonsággal tudjuk korlátozni.

Például, ha a formunkon található egy parancsgomb és egy kijelölőnégyzet és azt szeretnénk, hogy a felhasználó csak akkor használhassa a parancsgombot, ha már a kijelölőnégyzetet kijelölte, akkor a következő utasítást kell a rutinunkba beépíteni:

```
Parancsgomb.Enabled = kijelolonegyzet.Value = xlon
```

Érthetőbb, ha zárójelet teszünk:

```
Parancsgomb.Enabled = (kijelolonegyzet.Value = xlon)
```

Feladat:

Hozzunk létre egy formot, melyen helyezünk el egy összeadás nevű parancsgombot. A parancsgomb megnyomásakor a felhasználó adjon meg két egész számot, majd jelenjen meg a két szám összege.

Megoldás:

```
Private Sub gomb_proba_Click()  
    Dim a As Integer, b As Integer, s As Integer  
    Call Beolvas("a =", a)  
    Call Beolvas("b =", b)  
    s = osszead(a, b)  
    Kiir "a + b =", s  
End Sub  
  
Sub Beolvas(keres As String, ByRef valtozo As Integer)  
    valtozo = Val(InputBox(keres, "Adatbevitel"))  
End Sub  
  
Function osszead(t1 As Integer, t2 As Integer) As Integer  
    osszead = t1 + t2  
End Function  
  
Sub Kiir(info As String, ertek As Integer)  
    MsgBox info & CStr(ertek), , "Eredmeny"  
End Sub
```

6.3. Osztály modul (Class Module)

Osztály modult létrehozni a Beszúrás (Insert) menü Osztály modul (Class module) menüpontjával tudunk. Az osztály egy külön modul lesz a VBA makrószerkesztőjében. Az osztály nevét annak tulajdonságainál tudjuk beállítani, és ez lesz magának az újonnan létrejött típusnak is a neve. Osztályok létrehozása során az osztálymodulban globális változóként felsoroljuk az adattagokat, (Private/Public láthatósággal), illetve az osztály metódusait (függvényeit/eljárásait).

Az Osztály típusú *változót* az alábbi szintaktikával deklarálhatunk:

```
Dim <vmi> As New <Osztály>
```

A *New* szó használata arra utal, hogy *példányosításról* van szó. Az így létrejövő vmi *változó* egy új Osztály típusú objektum referenciáját tartalmazza. Amennyiben a *New* kulcsszót elhagytuk volna a deklaráció során, akkor egy üres referenciát tartalmazna a változó, azaz nem lennének használhatók az adattagjai. Ekkor a program egy későbbi szakaszában kell hozzá egy tényleges objektumot példányosítanunk, amelyet a *Set* utasítással tehetünk meg:

```
Dim vmi As Osztaly  
Set vmi = New Osztaly
```

A fenti két kódrészlettel ugyanazt érhetjük el.

Példa: Hozzunk létre egy *adozo* osztályt (Insert → Class module, majd a Properties ablakban a (Name) tulajdonságát írjuk át: *adozo*), mellyel egy adózó személyről a következő adatokat tudjuk tárolni: neve, anyja neve, adóazonosítója és 2010-ig visszamenőleg az adótartozásait.

Class module-ba írandó:

```
Public nev As String  
Public anyjaneve As String  
Public adoazonosito As String  
Public tartoz2012 As Integer  
Public tartoz2011 As Integer  
Public tartoz2010 As Integer  
Public ossztart As Integer
```

Állítsuk elő az *adozo* típusú objektum egy *konkrét előfordulását*! A Module-ba írandó kódrészlet (főprogram):

```
Sub Main()  
    Dim adoz As New adozo  
    adoz.nev = "Kiss Pál"  
    adoz.anyjaneve = "Molnár Piroska"  
    adoz.adoazonosito = "12345678"  
    adoz.tartoz2012 = 5
```

```

    adoz.tartoz2011 = 10
    adoz.tartoz2010 = 15
    MsgBox adoz.nev & " " & adoz.anyjaneve & " " &
    adoz.adoazonosito & " " & adoz.tartoz2012 & " " &
    adoz.tartoz2011 & " " & adoz.tartoz2010

```

End Sub

Az adozo osztálynak azonban jelenleg egyetlen metódusa sincs. Módosítsuk úgy a makrónkat, hogy hozzunk létre egy metódust, amely kiszámolja, hogy az adott adózónak mennyi az adótartozása összesen az elmúlt 3 évben:

Class module-ba írandó:

```

Public nev As String
Public anyjaneve As String
Public adoazonosito As String
Public tartoz2012 As Integer
Public tartoz2011 As Integer
Public tartoz2010 As Integer
Public ossztart As Integer
Sub tartozas_kiszamolas()
    If (tartoz2012 > 0 Or tartoz2011 > 0 Or tartoz2010 > 0) Then
        ossztart = tartoz2012 + tartoz2011 + tartoz2010
    End If
End Sub

```

Module-ba írandó kódrészlet (főprogram):

```

Sub Main()
    Dim adoz As New adozo
    adoz.nev = "Kiss Pál"
    adoz.anyjaneve = "Molnár Piroska"
    adoz.adoazonosito = "12345678"
    adoz.tartoz2012 = 5
    adoz.tartoz2011 = 10
    adoz.tartoz2010 = 15
    adoz.tartozas_kiszamolas
    MsgBox adoz.nev & " " & adoz.anyjaneve & " " &
    adoz.adoazonosito & " " & adoz.tartoz2012 & " " &
    adoz.tartoz2011 & " " & adoz.tartoz2010 & " " & adoz.ossztart
End Sub

```

A típushoz rendelt metódusok előnye akkor látszik igazán, ha létrehozunk még egy adoza típusú objektumot, és azt is feltöltjük adatokkal:

```
Sub Main()  
    Dim adoz As New adoza  
    Dim adoz2 As New adoza  
  
    adoz.nev = "Kiss Pál"  
    adoz.anyjaneve = "Molnár Piroska"  
    adoz.adoazonosito = "12345678"  
    adoz.tartoz2012 = 5  
    adoz.tartoz2011 = 10  
    adoz.tartoz2010 = 15  
    adoz.tartozas_kiszamolas  
    MsgBox adoz.nev & " " & adoz.anyjaneve & " " &  
    adoz.adoazonosito & " " & adoz.tartoz2012 & " " &  
    adoz.tartoz2011 & " " & adoz.tartoz2010 & " " & adoz.ossztart  
  
    adoz2.nev = "Nagy Judit"  
    adoz2.anyjaneve = "Horváth Éva"  
    adoz2.adoazonosito = "87654321"  
    adoz2.tartoz2012 = 0  
    adoz2.tartoz2011 = 30  
    adoz2.tartoz2010 = 2  
    adoz2.tartozas_kiszamolas  
    MsgBox adoz2.nev & " " & adoz2.anyjaneve & " " &  
    adoz2.adoazonosito & " " & adoz2.tartoz2012 & " " &  
    adoz2.tartoz2011 & " " & adoz2.tartoz2010 & " " &  
    adoz2.ossztart  
End Sub
```

A fenti kódban jól látható, hogy mindkét adózó esetén ugyanaz az eljárás számolja ki az összes adótartozást mindkét objektumban. Érdekesség azonban, hogy nincsen paramétere, hiszen az a típushoz rendelt, így a *tartozas_kiszamolas* metódus mindig *annak az objektumnak a mezőivel fog dolgozni, amelyhez kapcsolódóan meghívtuk*. Ennek köszönhetően az *ossztart* mezőt *el is rejthetjük* a felhasználók elől, oly módon, hogy az *ossztart* adattagnak *Private* láthatósági szintje legyen:

```
Public nev As String
Public anyjaneve As String
Public adoazonosito As String
Public tartoz2012 As Integer
Public tartoz2011 As Integer
Public tartoz2010 As Integer
Private ossztart As Integer
Sub tartozas_kiszamolas()
    If (tartoz2012 > 0 Or tartoz2011 > 0 Or tartoz2010 > 0) Then
        ossztart = tartoz2012 + tartoz2011 + tartoz2010
    End If
End Sub
```

Így azonban a főprogramban történő lekérdezés - MsgBox adoz.ossztart - nem fog működni, hiszen a privát láthatósági szintnek éppen az a lényege, hogy egyszerű minősítéssel nem férhetünk hozzá az adott adathoz (bezárás!).

A megoldás, hogy létrehozunk egy függvényt, amely visszatérési értéként visszaadja az ossztart mező értékét:

```
Public nev As String
Public anyjaneve As String
Public adoazonosito As String
Public tartoz2012 As Integer
Public tartoz2011 As Integer
Public tartoz2010 As Integer
Private ossztart As Integer
Sub tartozas_kiszamolas()
    If (tartoz2012 > 0 Or tartoz2011 > 0 Or tartoz2010 > 0) Then
        ossztart = tartoz2012 + tartoz2011 + tartoz2010
    End If
End Sub
Public Function gettartozas() As Integer
    gettartozas = ossztart
End Function
```

Az OO gyakorlatban az ilyen lekérdező függvények *get* szócskával kezdődnek. Ekkor a főprogram az alábbi módon módosul:


```
Sub Main()  
  
    Dim adoz As New adoza  
  
    Dim adoz2 As New adoza  
  
    adoz.nev = "Kiss Pál"  
  
    adoz.anyjaneve = "Molnár Piroska"  
  
    adoz.adoazonosito = "12345678"  
  
    adoz.tartoz2012 = 5  
  
    adoz.tartoz2011 = 10  
  
    adoz.tartoz2010 = 15  
  
    adoz.tartozas_kiszamolas  
  
    MsgBox adoz.nev & " " & adoz.anyjaneve & " " &  
    adoz.adoazonosito & " " & adoz.gettartozas  
  
    adoz2.nev = "Nagy Judit"  
  
    adoz2.anyjaneve = "Horváth Éva"  
  
    adoz2.adoazonosito = "87654321"  
  
    adoz2.tartoz2012 = 0  
  
    adoz2.tartoz2011 = 30  
  
    adoz2.tartoz2010 = 2  
  
    adoz2.tartozas_kiszamolas  
  
    MsgBox adoz2.nev & " " & adoz2.anyjaneve & " " &  
    adoz2.adoazonosito & " " & adoz2.gettartozas  
  
End Sub
```

Jelenleg a kódunk hátránya az, hogy a tartozas_kiszamolas() eljárást mindig meg kell hívunk, valahányszor módosítjuk a tartoz2010, tartoz2011, tartoz2012 mezőket. Az lenne a jó, ha ezt automatizálhatnánk. Erre megoldás lehet, ha a tartoz2010, tartoz2011, tartoz2012 adattagok értékét csak egy eljárásen keresztül engedjük módosítani, és ebben az eljárásban a módosításon kívül végrehajtjuk az ossztart mező beállítását is:

```
Public nev As String
Public anyjaneve As String
Public adoazonosito As String
Private tartoz2012 As Integer
Private tartoz2011 As Integer
Private tartoz2010 As Integer
Private ossztart As Integer
Sub tartozas_kiszamolas()
    If (tartoz2012 > 0 Or tartoz2011 > 0 Or tartoz2010 > 0) Then
        ossztart = tartoz2012 + tartoz2011 + tartoz2010
    End If
End Sub
Public Function gettartozas() As Integer
    gettartozas = ossztart
End Function
Public Sub settartoz2012(a As Integer)
    tartoz2012 = a
    tartozas_kiszamolas
End Sub
Public Sub settartoz2011(b As Integer)
    tartoz2011 = b
    tartozas_kiszamolas
End Sub
Public Sub settartoz2010(c As Integer)
    tartoz2010 = c
    tartozas_kiszamolas
End Sub
```

A beállítást végző metódusok neveit a gyakorlatban a *set* szócskával kezdjük.

Most már az új metódusok a tartoz2010, tartoz2011, tartoz2012 módosítását követően rögtön kiszámítják az össz adótartozás értéket is. Ezen mezők láthatóságát privátra állítottuk, ezáltal a főprogramban ezek az adattagok már nem érhetőek el, csak a megfelelő beállító metódusokon keresztül manipulálhatóak:

```

Sub Main()

    Dim adoz As New adozo
    Dim adoz2 As New adozo
    adoz.nev = "Kiss Pál"
    adoz.anyjaneve = "Molnár Piroska"
    adoz.adoazonosito = "12345678"

    adoz.settartoz2012 (10)
    adoz.settartoz2011 (10)
    adoz.settartoz2010 (15)

    MsgBox adoz.nev & " " & adoz.anyjaneve & " " &
    adoz.adoazonosito & " " & adoz.gettartozas

    adoz2.nev = "Nagy Judit"
    adoz2.anyjaneve = "Horváth Éva"
    adoz2.adoazonosito = "87654321"

    adoz2.settartoz2012 (0)
    adoz2.settartoz2011 (0)
    adoz2.settartoz2010 (2)

    MsgBox adoz2.nev & " " & adoz2.anyjaneve & " " &
    adoz2.adoazonosito & " " & adoz2.gettartozas

End Sub

```

Előfordulhat, hogy mégis szükségünk van a főprogramban a tartoz2010 vagy a tartoz2011 vagy a tartoz2012 adattagok értékére. Ekkor létre kell hoznunk megfelelő lekérdező metódusokat, melyeket privát láthatósági szint esetén *get/set* metóduspárokkal érhetünk el. A *get* metódus minden esetben függvény, a *set* pedig minden esetben egy eljárás.

```

Public Function gettartoz2012() As Integer

    gettartoz2012 = tartoz2012

End Function

```

A VBA lehetőséget biztosít a get/set függvények egyszerűsítésére, tulajdonságok, Property-k használatával. A tulajdonság egy olyan adattag, amelyet két metódussal valósítunk meg. Az egyik a beállítását végző eljárás (Let), a másik a lekérdezését végző Get függvény. A tulajdonságok részletes ismerete azonban meghaladja ezen tanfolyam kereteit, így ezekkel jelen könyvben részletesen nem foglalkozunk.

7. Állományok használata, eseményvezérlés

7.1. Állománykezelés

A fájlkezelés minden programozási nyelvben 3 lépésből tevődik össze:

1. fájl megnyitása
2. írás/olvasás a fájlba
3. fájl lezárása

Fájlok, munkafüzetek megnyitása:

A szöveges fájlok megnyitása írásra vagy olvasásra az alábbi szintaktikával lehetséges:

```
Open <fájlnév> For [Input|Output] As #<fájlazonosító>
```

A *fájlnév* egy szöveg, ami egy fájl *abszolút elérési útját* tartalmazza, a *fájlazonosító* egy egész szám (1-255), amelyhez még nem rendeltünk fájlt. A fájlokat tehát a fájlazonosítókön keresztül érhetjük el, de figyelniük kell arra, hogy ugyanazon azonosítóval ne nyissunk meg két fájlt egyszerre.

A munkafüzeteket használat előtt szintén mindig meg kell nyitni, használat után pedig le kell zárni. A munkafüzet megnyitására VBA nyelvben az *Open* metódus használatos.

Open szintaktikája munkafüzetek esetén a következő:

```
Workbooks.Open    Filename    :=   állománynév    [,   UpdateLinks,  
                                  ReadOnly, Format, Password, WriteResPassword,  
                                  IgnoreReadOnlyRecommended, Origin, Delimiter,  
                                  Editable, Notify, Converter]
```

A paraméterek közül csak az első kötelező, a többi elhagyható.

Állományok módosítása:

A szöveges fájlok feldolgozása többnyire sorról-sorra történik.

Az írásra megnyitott fájlokba a Print függvénnyel írhatunk:

```
Print #<fájlazonosító>, <szöveg>
```

amelynek első paramétere, hogy melyik fájlba szeretnénk írni, második paramétere pedig egy szöveg, amelyet az adott azonosítóval rendelkező fájlba szeretnénk írni. A fájlba a szöveg egy új sorként fog bekerülni, azaz a Print függvény automatikusan a szöveg végére ír egy "új sor" (CrLf) karaktert.

Fájlból olvasni a Line Input metódussal tudunk, amelynek paraméterezése:

```
Line Input #<fájlazonosító>, <szöveges_változó>
```

Az adott azonosítójú fájlból kiolvassuk egy sort, és ezt a sort beletesszük a második paraméterként kapott szöveg típusú változóba. A sor végéről az "új sor" karaktert (CrLf - sortörést) automatikusan levágja.

Állományok lezárása:

Fájlokat lezárni a Close függvénnyel tudunk, amelynek egyetlen paramétere a fájlazonosító:

```
Close #<fájlazonosító>
```

A szöveges fájlok feldolgozásánál gyakran használt függvény az EOF függvény, amely az "End-Of-File" kifejezés rövidítése, egyetlen paramétere egy fájlazonosító, visszatérési értéke logikai igaz, ha elértük a fájl végét, logikai hamis, ha nem értük még el a fájl végét. Ezt a függvényt használhatjuk ciklusok feltételében egy megnyitott fájl sorról sorra történő feldolgozásokor.

Ha már a munkafüzetre sincs szükségünk, akkor le kell azt is zárnunk. A lezárásra a Close metódust használjuk.

A Close szintaktikája a következő:

```
<Objektumnév>.Close [SaveChanges] [, [FileName] [, RouteWorkbook]]
```

A metódus minden paramétere elhagyható, ezért ezek ismertetését szintén nem teszem meg jelen könyvben.

Az objektumnév a lezárandó munkafüzetet vagy ablakot azonosítja, melyek szokásos formája a következő:

Workbooks(állománynév) illetve ActiveWorkbook vagy ActiveWindow.

7.2. Eseményvezérlés

Eseményvezérlésről akkor beszélünk, ha egy metódusnak, felhasználói eljárásnak akkor kell csak elindulnia automatikusan, *ha egy bizonyos esemény bekövetkezik*. Az eseményvezérelt eljárások neve nem választható meg szabadon. A névnek ugyanis utalnia kell az eseményre és az objektumra is, amelyhez kötődik. Ha tudni szeretnénk, hogy munkafüzetünkben milyen eseményvezérelt eljárások találhatóak, akkor váltsunk át VB nézetre és ott nyomjuk meg az F2 billentyűt, vagy nyissuk meg az OBJEKTUMTALLÓZÓ ablakot és ott jelöljük ki a VBAProject objektumot.

Az eseményvezérelt eljárásoknak több csoportját különböztetjük meg az esemény jellege szerint: munkafüzet események, munkalap események, párbeszédlapok eseményei, párbeszédlapok vezérlőelemeihez kötött események, eseményvezérelt metódusok.

A könyv és a hozzá kapcsolódó tanfolyam terjedelme miatt, csak a leggyakrabban használtakra térünk ki, a további metódusok megismerését az elszántabb olvasókra bízom.

7.2.1. Munkafüzet események

A munkafüzet események akkor következnek be, ha egy aktív munkafüzetben, vagy annak egy munkalapján következik be valamilyen *változás*. A munkafüzet események által vezérelt rutinokat a munkafüzet (*ThisWorkbook*) kódlapján kell elhelyezni.

- **Open**

A munkafüzet megnyitásakor következik be.

Szintaktika: `Sub Workbook_Open()`

- **NewSheet**

Ha új munkalapot szúrunk be az aktív munkafüzetbe, akkor fut le.

Szintaktika: `Sub Workbook_NewSheet(ByVal <lap> As Object)`

A lap paraméter az új munkalap nevét tartalmazza.

- **SheetActivate**

Bekövetkezik valamelyik munkalap aktiválásakor.

Szintaktika: `Sub Workbook_SheetActivate(ByVal <lap> As Object)`

A lap paraméter az aktivált munkalap nevét tartalmazza.

- **SheetDeactivate**

Bekövetkezik ha átváltunk egyik munkalapról egy másikra és az lesz az aktív.

Szintaktika: `Sub Workbook_SheetDeactivate(ByVal <lap> As Object)`

A lap paraméter az elhagyott munkalap nevét tartalmazza.

7.2.2. Munkalap események

A munkalap eseményeket az aktív munkalapon végzett kézi beavatkozások váltják ki, hatásukat csak a munkalap kódlapján lévő eljárások érzékelik. A munkalap eseményeket a munkalap kódlapján helyezzük el.

- **Activate**

Bekövetkezik egy létező munkalap aktiválásakor.

Szintaktika: `Sub Worksheet_Activate()`

- **Deactivate**

Bekövetkezik egy munkalap elhagyásakor.

Szintaktika: `Sub Worksheet_Deactivate()`

7.2.3. Párbeszédlap események

Mind a párbeszédlapokhoz, mind a rajtuk elhelyezkedő vezérlőelemekhez számos esemény kapcsolható. A párbeszédlap eseményeket a formon végzett változtatások, események váltják ki.

- **Load**

Az űrlap inicializálására használjuk.

- **Activate**

Minden olyan esetben jön létre, mikor a formhoz tartozó ablak aktívvá válik.

- **Deactivate**

Bekövetkezik, ha egy másik ablak lesz aktív.

7.2.4. Párbeszédlap vezérlőelemeihez kötött események

Általános szintaxisa ezen eljárások fejrészének a következő:

```
Sub vezeloelem_neve_esemeny ([parameterlista])
```

A négy legfontosabb esemény a következő:

- **Click**: bekövetkezik a vezérlőelemre kattintáskor.
- **Change**: bekövetkezik, ha megváltozik a vezérlőelem Value tulajdonsága.
- **Enter**: Bekövetkezik a vezérlőelem aktiválásakor, de még az aktívvá válás előtt.
- **Exit**: bekövetkezik, amikor elhagyjuk az aktív vezérlőelemet és még mielőtt az űrlap egy másik vezérlőeleme lenne aktív. Ez a rutin csak akkor aktivizálódik, ha a Cancel paraméter False értékű. Ennek köszönhetően a szintaxisa kicsit eltér a többitől.

```
Sub vezeloelem_neve_Exit(ByVal Cancel As MsForms.  
returnBoolean)
```

7.2.5. Eseményvezérelt metódusok

Az eseményvezérelt metódusokat egy rendszeresemény kelti életre. Közös jellemzőjük, hogy nevük mindig „on” taggal kezdődik. Két ilyen lényeges metódus van:

- Időzítés (**onTime**)

Beütemezi a hozzá kapcsolt eseménykezelő eljárást, melyet aztán a megadott időpontban lefuttat.

Szintaktika:

```
Application.onTime Earliest_Time := inditasi_ido,  
Procedure := "eljarasnev" [, Latest_Time :=  
varakozasi_ido] [, Schedule := boolpar]]
```

Az *inditasi_ido* tárgynapra vonatkozó abszolút idő. Az *eljarasnev* azon eljárás neve, melyet el kell indítani a megadott időpontban.

Ha az eseményként megjelölt időpontban az Excel foglalt, akkor az esemény várakozásra kerül. Legközelebb Latest_Time paraméterrel megadott időpontban próbálja újból lefuttatni, ha ekkor sem sikerül, akkor nem próbálkozik automatikusan tovább. A Schedule paraméterrel kell élesíteni vagy megszüntetni az időzítést.

Az időzítést tartalmazó munkafüzetet az indítókönyvtárban kell elhelyezni, az élesítő metódust pedig a Workbooks_Open indítóeljárásban.

– Billentyűzet figyelés

Az onKey metódus minden egyes alkalommal lefut, ha a felhasználó leüt egy billentyűt.

Szintaktikája:

```
Application.onKey    Key := "billentyűkód" [, Procedure :=  
"eljaras_nev"]
```

A billentyűkód a leütött billentyű azonosítója. Az eljárásnév paraméter nem kötelező, ha viszont szerepel, akkor a hivatkozott billentyű leütésekor elindul ez az eljárás.

8. Összetett példafeladat megoldása

1. Feladat:

Készítsünk egy bolthálózat tulajdonos részére egy kalkuláló és nyilvántartó programot a megfelelő felhasználóbarát kinézettel. Hozzunk létre ehhez egy formot, mely kitölti a teljes képernyőt. A form az Excel állomány megnyitásakor automatikusan nyíljon meg. Tartalmazzon két beviteli mezőt (mely az adott bolt napi bevételét és kiadását tartalmazná), egy lenyíló listát (ahol lehessen kiválasztani, hogy melyik kerületben lévő boltjának adatait szeretné rögzíteni), egy dátum mezőt, melyhez rendeljük hozzá a naptár ActiveX-komponenst. Ez úgy működjön, hogy amelyik napot kiválasztjuk a naptár panelon, automatikusan jelenjen meg a dátum mezőben. A napi nyereséget a gép számolja ki, abban az esetben, ha a számolás nyomógombra kattintunk. Az exportálás nyomógomb pedig a bevitt adatokat exportálja ki egy erre a célra létrehozott táblázatba. Tehát, nyissa meg a C:\nyilvantartas.xls nevű fájlt és a napi nyereséget listázza ebbe a fájlba oly módon, hogy csak a hely, dátum és a nyereség kerüljenek bele. Minden nap az előző napi bejegyzésekhez fűzze hozzá az újat. Az indítófájl (zarofeladat.xls) és benne a háttéradatokat tartalmazó munkalap neve is legyen: „zarofeladat”!

12. ábra Form kinézete

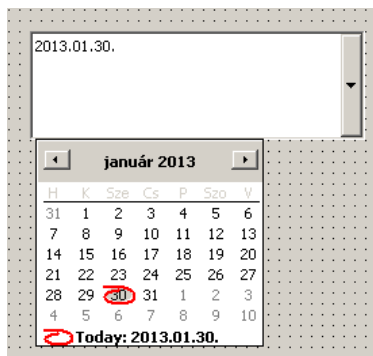
A MEGOLDÁS FŐ LÉPÉSEI:

1. Készítsük el a form felületét!

- Váltunk át VB nézetre.
- Válasszuk ki a megfelelő ikont és szúrjunk be egy formot. (*Insert UserForm*)
- Állítsuk be a form háttérszínét fehérre. → Tulajdonság panel *BackColor* tulajdonságánál.
- Az eszközkészlet segítségével rajzoljuk fel a formra a megfelelő számú beviteli mezőt, címkét, lenyílólistát, nyomógombot. → *átlós mozdulattal lenyomott bal egérgombbal*
- Helyezzük el a naptár panelt a formon az ActiveX vezérlő segítségével.

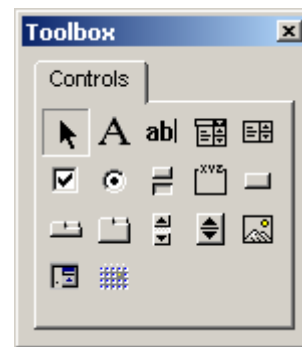
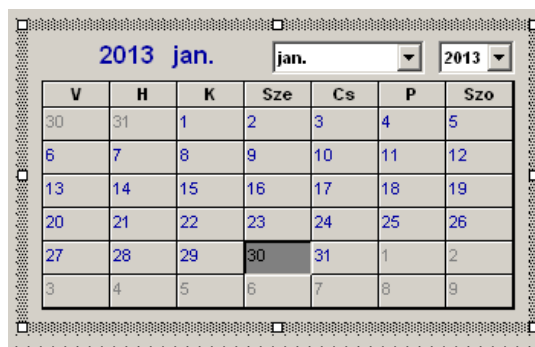
Két lehetőségünk van:

- Tools/Additional Controls → Microsoft Data And Time Picker Control 6.0 ☒ majd az ActiveX vezérlők közé nyissuk meg (Ctrl T) a C:\Windows\System32\mscomct2.ocx fájlt. Ekkor új komponensként megjelenik a Microsoft Windows CommonControls2 6.0 (SP6)



- Vagy A Start → Üzemeltetés → Hirdetett programok futtatása ablakból a *Segédprogram* kategória alatt megtalálható *mscal.ocx* (*Naptár vezérlőelem*) *ActiveX* vezérlő... programot futtassuk meg. Ez pár másodperc alatt lefut.

Visual Basic editorban a Userform szerkesztésekor a *Toolbox* ablak (lásd kép) *Controls* fülének egy üres részén kattintsunk jobb egérgombbal, válasszuk az *Additional Controls...* pontot, a megjelenő ablakban keressük meg a *Naptár vezérlőelem 11.0*-át, válasszuk ki, majd pipáljuk be, OK.



2. Gondoskodjunk arról, hogy a „zarofeladat.xls” megnyitásakor a formunk automatikusan betöltődjön.

- A ThisWorkbook nevű kódapon helyezzük el a következő kódot:

Option Explicit

```
Sub Workbook_open()

    Workbooks(ActiveWorkbook.Name).Activate

    ActiveWorkbook.Sheets(„zarofeladat”).Activate

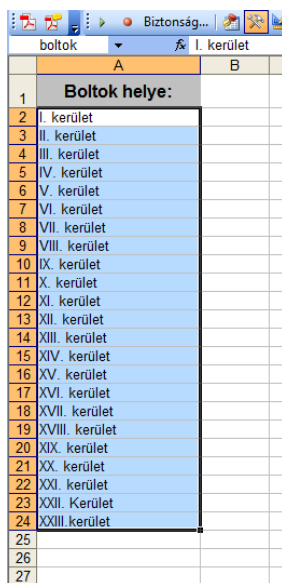
    Load UserForm1

    UserForm1.Show

End Sub
```

3. Állítsuk be, hogy a helyet lenyíló listából tudjuk választani.

- A lenyíló lista elemeit gépeljük be a „zarofeladat.xls” nevű fájl „zarofeladat” nevű munkalapjának A2 cellájától kezdve lefelé. Az A1 cellába gépeljük be, hogy „Boltok helye:”



	A	B
1	Boltok helye:	
2	I. kerület	
3	II. kerület	
4	III. kerület	
5	IV. kerület	
6	V. kerület	
7	VI. kerület	
8	VII. kerület	
9	VIII. kerület	
10	IX. kerület	
11	X. kerület	
12	XI. kerület	
13	XII. kerület	
14	XIII. kerület	
15	XIV. kerület	
16	XV. kerület	
17	XVI. kerület	
18	XVII. kerület	
19	XVIII. kerület	
20	XIX. kerület	
21	XX. kerület	
22	XXI. kerület	
23	XXII. kerület	
24	XXIII. kerület	
25		
26		
27		

14.ábra

- Az A2:A24 tartományba gépeljük be a 23 kerületet, majd jelöljük ki ezt a tartományt és végül a szerkesztőléc elején lévő név mezőbe gépeljük be, hogy *boltok*. Így ezt a tartományt ezután képesek leszünk egy névvel (boltok) azonosítani, amire a form szerkesztésekor lesz nagy szükségünk.
- Váltunk át VB nézetre és ott kattintsunk a ComboBox1 vezérlőelemre, ekkor az ő tulajdonságpanelje válik aktív. Itt keressük meg a RowSource tulajdonságot, és gépeljük be, hogy „boltok”. Ezáltal a lista felveszi a 23 kerületet, mint értéket.

4. Oldjuk meg, hogy a NAPI NYERESÉG mező vegye fel a (bevétel – kiadás) értéket, abban az esetben, ha a SZÁMOL gombra kattintunk.

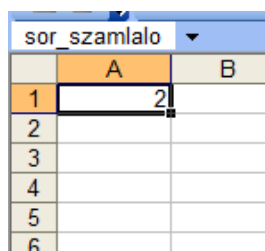
- Ahhoz, hogy a NAPI NYERESÉG mezőbe ne tudjon a felhasználó értéket bevinni, csupán megjelenítésre legyen alkalmas, végezzük el a következő beállítást: a tulajdonságpanelen az ENABLED tulajdonság értékét állítsuk False-ra.
- A form kódlapjára gépeljük be a következő eljárásokat:

```
Private Sub CommandButton1_Click()
    Call szamolas
End Sub

Private Sub szamolas()
    TextBox4.Value = TextBox2.Value - TextBox3.Value
End Sub
```

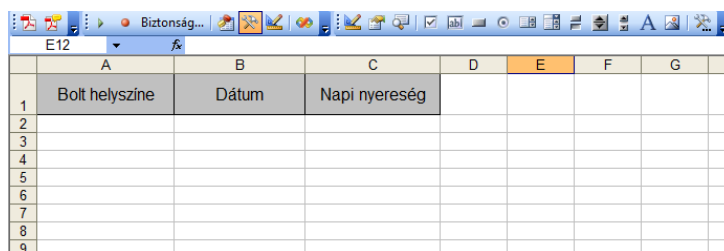
5. Exportálás gomb működése

- Hozzunk létre a C:\ meghajtón egy NYILVANTARTAS.XLS nevű munkafüzetet. A munkafüzet MUNKA1 nevű munkalapján készítsük el a táblázat fejlécét az első sorban. A MUNKA2 nevű munkalap A1-es cellájában pedig hozzunk létre egy segédváltozót, mely a sorok számlálására lesz alkalmas, ennek a cellának az új neve ezért legyen: **sor_számlalo**. Ezután zárjuk is be. → *Excel-ben: Fájl menü/új dokumentum, aztán mentés másként.*



	A	B
1	2	
2		
3		
4		
5		
6		

Munka2



	A	B	C	D	E	F	G
1	Bolt helyszíne	Dátum	Napi nyereség				
2							
3							
4							
5							
6							
7							
8							
9							

Munka1

- Aztán menjünk vissza VB nézetbe a form kódlapjára és gépeljük be a következő kódot:

```
Private Sub CommandButton2_Click()
    Dim sor As Integer
    Dim nev As String
    nev = ActiveWorkbook.Name
    Workbooks.Open ("C:\nyilvantartas.xls")
    Sheets("Munka1").Select
```

```
sor = Worksheets("Munka2").Range("sor_szamlalo").Value
Cells(sor, 1).Select
Cells(sor, 1) = ComboBox1.Value
Cells(sor, 2).Select
Cells(sor, 2) = TextBox1.Value
Cells(sor, 3).Select
Cells(sor, 3) = TextBox4.Value
sor = sor + 1
Worksheets("Munka2").Range("sor_szamlalo").Value = sor
Windows("nyilvantartas.xls").Close SaveChanges:=True
```

End Sub

2. Feladat:

Készítsünk makrót, amely az A1: D3 tartomány háttérszínét pirosra, betűszínét fehérre, betűtípusát 12-es Arial-ra állítja.

```
Sub gyakketto()
    Range("A1:D3").Select
    With Selection.Font
        .Name = "Arial"
        .Size = 12
        .Color = vbWhite
    End With
    With Selection.Interior
        .Color = vbRed
    End With
End Sub
```

3. Feladat:

Készítsünk makrót, mely az A1:J10 tartományt feltölti a 10×10-es szorzótáblával. A 7-el osztható számok háttérszíne legyen piros. Majd jelenjen meg egy MsgBox, melyben megjelenik, hogy hány darab 7-el osztható szám van.

```
Sub gyakorlo3()  
    Dim i, j As Integer  
    db = 0  
    For i = 1 To 10  
        For j = 1 To 10  
            Cells(i, j) = i * j  
            If Cells(i, j) Mod 7 = 0 Then Cells(i, j).Interior.Color  
            = vbRed  
            If Cells(i, j) Mod 7 = 0 Then db = db + 1  
        Next  
    Next  
    MsgBox ("Ennyi 7-tel osztható szám van: " & db)  
End Sub
```

4.Feladat:

Helyezzünk el a munka1 nevű munkalapon egy parancsgombot, amelyet ha megnyomunk, a képernyőn MsgBox-ban jelenjen meg a következő felirat: „Imádom a makró programozást!”

Excel nézetben helyezzünk el egy nyomógombot, majd rendeljünk hozzá egy új makrót. A forráskód modulba kerüljön, és a következőként nézzen ki:

```
Sub CommandButton1_Click()  
    MsgBox ("Imádom a makró programozást!")  
End Sub
```

5.Feladat:

Készítsen függvényt, amely a sugár megadása után kiszámítja a gömb térfogatát.

```
Function gomb_terfogat(ByVal r As Single) As Double  
    gomb_terfogat = (4 / 3) * 3.14 * (r ^ 3)  
End Function  
Sub bekeres_szamolas()  
    Dim sugar As Single  
    Dim terfogat As Double  
    sugar = InputBox("Kérem a gömb sugarát:")  
    terfogat = gomb_terfogat(sugar)  
    MsgBox ("A gömb térfogata: " & terfogat)  
End Sub
```

6.Feladat:

Készítsen makrót, mely a felhasználótól bekéri, hogy hány nevet szeretne rendezni, majd bekéri a neveket, végül ábécé sorrendbe rendezi (buborék rendezés) és kiírja a rendezett névsort.

```
Sub rendezes()
```

```
    Dim i, j, z As Integer
```

*A **dim <változó>()** utasítás dinamikus tömböt hoz létre*

```
    Dim t() As String
```

```
    Dim seged As String
```

```
    elemszam = InputBox("Hány nevet szeretne rendezni?")
```

```
    ReDim t(1 To elemszam) As String
```

*A **ReDim [Preserve] ([<alsóindex> To] <felsőindex>)** utasítás a dinamikus tömb dimenzióit megváltoztatja futás közben.*

```
    For i = 1 To elemszam
```

```
        t(i) = InputBox("Név" & i, ":")
```

```
    Next
```

```
    For j = 1 To elemszam - 1
```

```
        For z = j + 1 To elemszam
```

```
            If t(j) > t(z) Then
```

```
                seged = t(z)
```

```
                t(z) = t(j)
```

```
                t(j) = seged
```

```
            End If
```

```
        Next
```

```
    Next
```

```
    MsgBox ("A nevek rendezett sorrendben:")
```

```
    For i = 1 To elemszam
```

```
        MsgBox (t(i))
```

```
    Next
```

```
End Sub
```

Azoknak, akiknek a feladat meghozta a kedvét a kísérletezéshez, természetesen lehet még szépíteni különböző lehetőségekkel. Például: a fájlba történő adatexportálás ciklusokkal való kibővítése oly módon, hogy ne zárja, majd nyissa újra a fájlt, hanem a 23 kerület adatainak felvitele után, vagy a kilépés előtt kérdezzen csak rá a mentésre és csak akkor zárja be a fájlt. Továbbá szépíthető a feladat olyanokkal, ha a háttérben futó munkalap adatait elrejtjük, hogy

a felhasználó ne is lássa, hogy Excel munkalapról irányítjuk a formot. Aki pedig igazán elszánt lett a makró programozás tanulása iránt, az megpróbálkozhat azzal, hogy a nyilvantartas.xls fájlt csatolt állományként elküldje magának e-mail-ben egy gombnyomással. Ezek azonban már mind olyan szintű ismeretek, melyek az alapok elsajátítása után Önállóan is megtanulhatók, de ahhoz viszont nehezek, hogy egy ilyen terjedelmű könyvben kitérjünk rájuk.

További jó kísérletezést kívánok minden kedves olvasónak!