VISUAL BASIC SCRIPT

PROGRAMOZÁSI ÚTMUTATÓ

Ez az összefoglaló a közismereti informatika tantárgy Algoritmusok és adatok fejezetéhez készült. Tartalmazza a tantárgy emelt szintű érettségijéhez szükséges tudnivalókat is. Nem tekinthető azonban a VBScript teljes körű leírásának. Ezzel kapcsolatban lásd: VBScript dokumentáció.
Az összefoglaló összeállításakor néhány kivételtől eltekintve a tanterv és az emelt szintű érettségi követelményeit tartottuk szem előtt. Nem tértünk ki a grafikus felület objektumainak kezelésére, mert ez nem része a tantervnek. A 10. osztályos tankönyv (Nemzeti Tankönyvkiadó, r.sz. 16172) *Szövegdobozok, parancsgombok és társaik* leckéje bemutatja az objektumok használatának alapjait. Az összefoglalóban esetenként megjegyzésként utalunk az objektumokkal kapcsolatos tudnivalókra.
Az összefoglaló általában nem tartalmaz konkrét példákat. Ezeket lásd [3]-ban. A közölt forráskód dőlt betűs részeit helyettesíteni kell a megfelelő tartalommal (például a *kifejezés* helyett írjuk be a megfelelő kifejezést). A szintaxis bemutatásánál a szögletes zárójelben lévő részek elhagyhatók.

A Visual Basic Script programozási nyelv

A Visual Basic Script jellemzői

A Microsoft Visual Basic Scripting Edition (rövidítve: VBScript) a Windows operációs rendszer vezérlésének[1], illetve a weblapok kliens- és szerveroldali programozásának egyik nyelve. Világos, könnyen érthető szintaxisa, gazdag eszköztára, a programfejlesztés egyszerű módja alkalmassá teszi az algoritmusok számítógépes megvalósításának bemutatására, a programozás alapelemeinek elsajátítására. A programok írásához és futtatásához csak egy szövegszerkesztőre (például Jegyzettömb), illetve egy interpreterre van szükség. A Windows operációs rendszer tartalmaz ilyen interpretert.
A szkripteket *hta* kiterjesztés esetén önálló ablakban, *htm* kiterjesztéssel pedig egy böngészőben (például Internet Explorer[2]) futtathatjuk. **A szkripteket tartalmazó (dinamikus) weblapok készítése nagymértékben motiválja a diákokat a tananyag elsajátításában.**
Bár a nyelvet elsősorban dinamikus weblapok programozására hozták létre, a forráskódban egyáltalán nem szükséges HTML-elemeket használni, így **a VBScript programok írásához alapvetően nincs szükség a HTML ismeretére**.
A VBScript tartalmazza mindazon eszközöket, melyek az emelt szintű informatika érettségi feladatainak megoldásához szükségesek.
A VBScript lehetővé teszi a grafikus felhasználói felület objektumainak kezelését (szövegdobozok, parancsgombok stb., lásd: [3]). Így egyszerűen és szemléletesen elsajátíthatjuk az objektumorientált programozási technikához szükséges tudnivalókat.
A VBScript megalapozza a Visual Basic programozási nyelv használatát. A VBScript ismeretében könnyen áttérhetünk a Visual Basic-re, melynek 2005-ös változata teljes mértékben objektumorientált, széles körben használható, modern programozási nyelv. A Visual Basic 2005 szerepel az emelt szintű informatika érettségin engedélyezett szoftverek listáján (2007). Ingyenes Express Edition fejlesztői környezete letölthető a Microsoft webhelyéről:

http://msdn.microsoft.com/vstudio/express/vb

A Visual Basic 6.0 fejlesztői környezete szintén szerepel az informatika érettségin és az informatika OKTV-n engedélyezett szoftverek listájában.
A VBScript ismeretek felhasználhatók az Office-alkalmazások programozásában is (makrók készítése). Az alkalmazások programozási nyelve, a Visual Basic for Applications (VBA) szintaxisa és utasításkészlete szinte teljes egészében megegyezik a VBScript-tel.

A VBScript interpreter

A Windows operációs rendszer VBScript interpretere a telepítéskor kerül a háttértárra:

\Windows\system32\vbscript.dll

A *hta* kiterjesztésű fájlokat az *mshta.exe* futtatja az interpreter segítségével. Az *exe* fájlt a *\Windows\system32* mappában találjuk meg. A *hta* fájlra történő dupla kattintás egyenértékű az

```
mshta elérésiút\fájlnév.hta
```

parancs kiadásával.
A *htm* kiterjesztésű fájlokat a böngésző futtatja (például *\Program Files\Internet Explorer\iexplore.exe*).

---

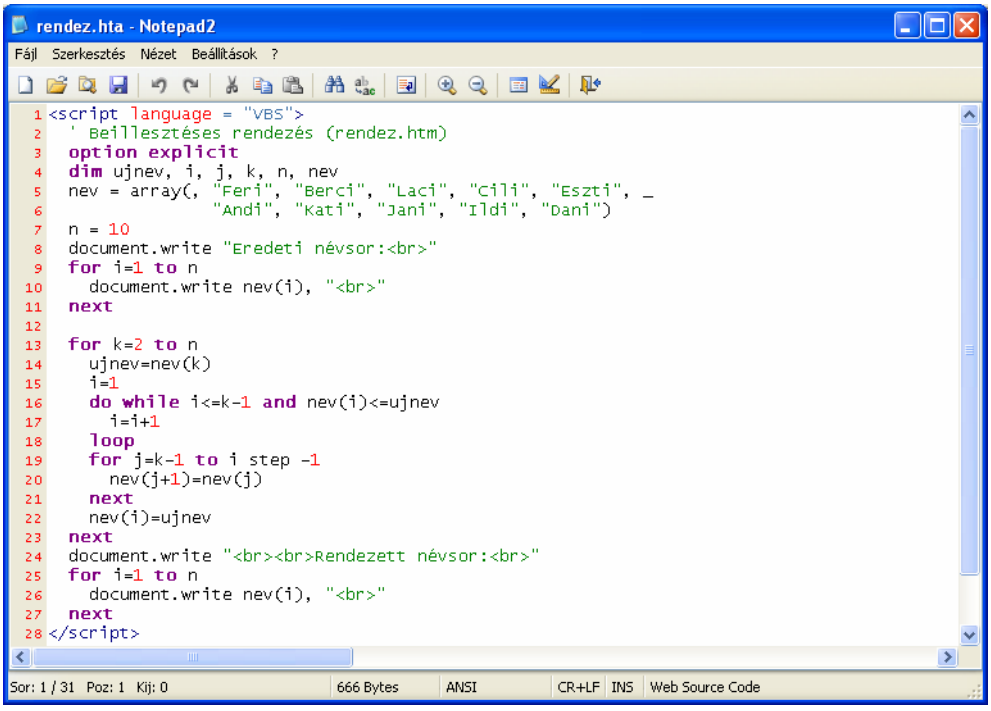
[1] A Windows Scripting Host szolgáltatás útján. Lásd például: Windows Script Host.
[2] Más böngészőkkel kapcsolatban lásd: A program futtatása.

A programfejlesztés menete

A forráskód elkészítése

A forráskódot egy egyszerű szövegszerkesztővel (például Jegyzettömb) írhatjuk meg. A fájlt *hta* (vagy *htm*) kiterjesztéssel mentsük el!
A Jegyzettömbben kapcsoljuk ki a hosszú sorok tördelését (*Formátum* menü). Ekkor bekapcsolhatjuk az állapotsor megjelenítését (*Nézet* menü). Az állapotsor a hibajavításnál hasznos, mert jelzi, hogy a kurzor hányadik sorban (és hányadik oszlopban) helyezkedik el.
Néhány Windows változatnál a kiterjesztések elrejtése esetén a Jegyzettömb automatikusan *txt* kiterjesztéssel látja el a fájlt. Ha például a mentésnél a *program.hta* nevet adjuk a fájlnak, akkor az azonosító *program.hta.txt* lesz, így a megnyitásnál nem indul el az interpreter (ugyanez vonatkozik a *htm* kiterjesztésre is). A kiterjesztések elrejtését az Intéző *Eszközök*/*Mappa* beállításai menüjében kapcsolhatjuk ki a *Nézet* fülnél a *Fájlok és mappák* csoportban (*Ismert fájltípusok kiterjesztésének elrejtése*).

A Windows Jegyzettömbje helyett a forráskódok elkészítéséhez és szerkesztéséhez javasoljuk az ingyenes **Notepad2** programot, melynek magyar nyelvű változata letölthető a flo's freeware webhelyről. A *Notepad2* telepítés nélkül futtatható. A program módosítható színkódokkal kiemeli a forráskód különböző szintaktikai elemeit, automatikus behúzásokat alakít ki és számos további eszközzel rendelkezik.
A *Notepad2* magyar nyelvű változatának szüksége van az *msvcr70.dll* fájlra, amit a környezeti változók beállítása helyett egyszerűen átmásolhatunk a *Notepad2*-t tartalmazó mappába. A *dll* fájlt megtaláljuk például a
C:*Program Files\Microsoft Office\OFFICE10\VS Runtime*
illetve a
C:*Program Files\Microsoft Office\OFFICE11\VS Runtime*
mappában, vagy letölthetjük a DLL-files.com webhelyéről.



Forráskódkészítés a Notepad2 programmal

Megjegyzés: a *txt* fájlok megnyitását az Intézőben a jobb egérkattintásra előtűnő helyi menü *Társítás* parancsával rendelhetjük hozzá a *Notepad2* programhoz.

A VBScript programok futtatása

hta fájlok futtatása

Ha a forráskódot *hta*[3] kiterjesztéssel mentjük el, akkor az Intézőből dupla kattintással indíthatjuk a végrehajtást. A program az operációs rendszerben megszokott ablakot nyit meg. Bármilyen HTML fájl átírható *hta* kiterjesztésűre.
A fájl megnyitásakor interpreter elvégzi a teljes kód szintaktikai ellenőrzését, illetve a függvényeket és eljárásokat kivéve végrehajtja a szkript utasításait (a függvényeket és eljárásokat csak a meghívásuk esetén).
A *hta* fájlok futtatásakor megnyíló ablak tulajdonságait (ikon, keret, gombok stb.) a [3] ismerteti (91. oldal). Ha megfelelnek számunkra az alapértelmezett értékek, akkor nincs szükség a beállításokra.
A *hta* fájlokról bővebben lásd: HTML-alkalmazások.

htm fájlok futtatása

A *htm* fájlokat böngészővel nyithatjuk meg. Az Internet Explorer az interpreter segítségével végrehajtja a szkriptek utasításait. A FireFox és a Mozilla (illetve a SeaMonkey) böngészőkhöz a

http://ietab.mozdev.org

webhelyről tölthetünk le a szkriptek futtatásához szükséges plugint (IE Tab). Bár a webhely nem említi, a plugin működik a SeaMonkey böngészőben is.

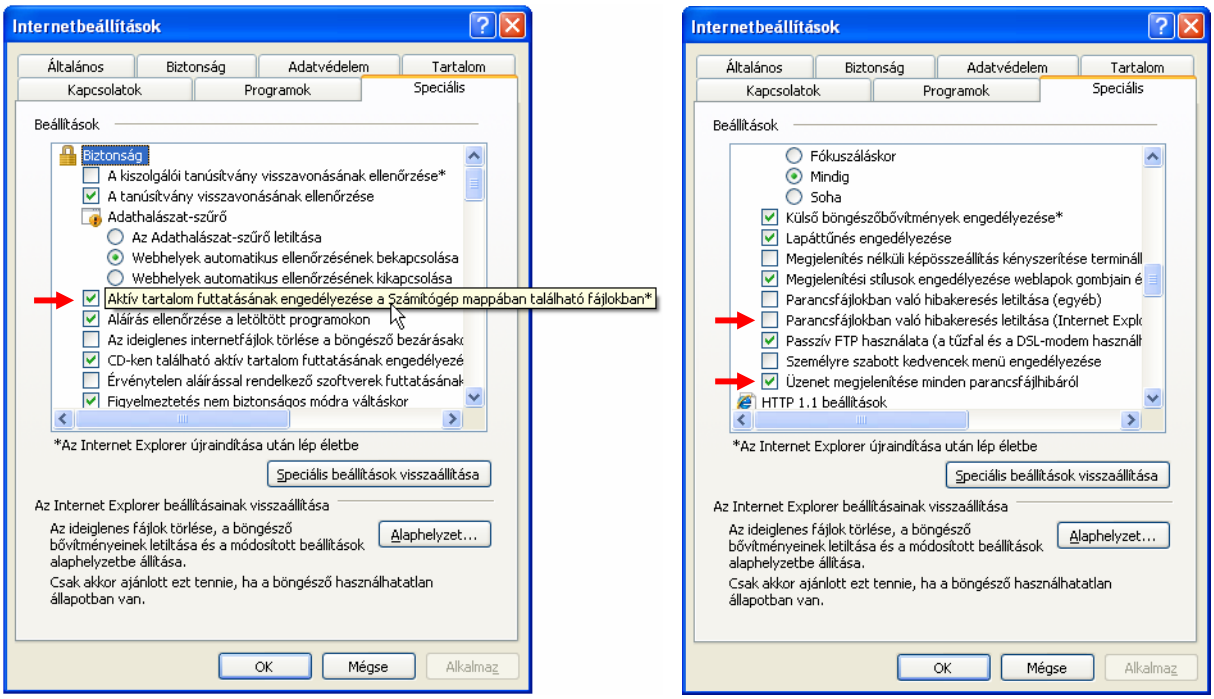
Az Internet Explorer beállítása

> A Microsoft XP SP2 biztonsági okokból letiltja a szkriptek végrehajtását. Ennek elkerüléséhez engedélyezzük az Internet Explorerben az aktív tartalom futtatását a Sajátgép (IE7: Számítógép) mappában található fájlokra (*Eszközök*/*Internetbeállítások*/*Speciális*/*Biztonság*).

A következő beállításokkal megkönnyítjük a hibakeresést:
Eszközök/*Internetbeállítások*/*Speciális*, *Böngészés* csoport

- Parancsfájlokban való hibakeresés letiltása kikapcsolva
- Üzenet megjelenítése minden parancsfájlhibáról bekapcsolva

Így minden hiba esetén hibaüzenetet kapunk, amely magyar nyelven megadja a hiba leírását és az előfordulás helyét.
Ha csak *hta* kiterjesztésű fájlokat használunk, akkor kikerülünk a böngésző ellenőrzése alól, így nincs szükség az Internet Explorer beállításaira.



Az Internet Explorer beállításai a szkriptek futtatásához

[3] HTML application: HTML-alkalmazás.

A vírusirtók és a szkriptek

Sok vírusirtó (például Norton AntiVirus) biztonsági okokból nem engedélyezi a szkriptek számára a fájlrendszer elérését. A futtatáskor megjelenő üzenetablakban engedélyezhetjük a szkript végrehajtását.

A program módosítása

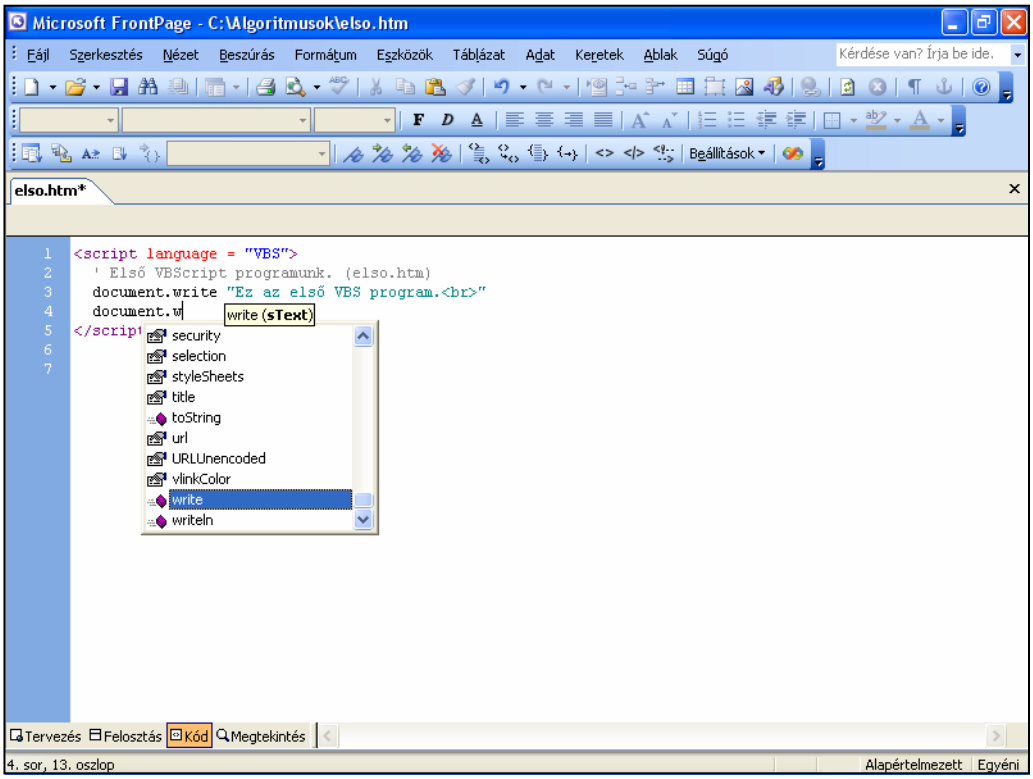
Egyszerű programok készítésénél hagyjuk megnyitva a Jegyzettömböt és a böngészőt is. A módosítás menete:

1. a Jegyzettömbben elvégezzük a forráskód módosítását,
2. mentjük a forráskódot,
3. átváltunk a programablakba, illetve a böngészőbe (például *Alt+Tab*),
4. frissítjük a fájlt (F5).

Egy már meglévő fájlt célszerű először megnyitni, majd megjeleníteni a forrást (a jobb egérgomb gyorsmenüjében: *Forrás megtekintése*).

A programfejlesztés eszközei

A VBScript programok készítése egy egyszerű szövegszerkesztőn és egy interpretert tartalmazó operációs rendszeren (illetve böngészőn) kívül más eszközt nem igényel. Összetett programok fejlesztéséhez azonban célszerű valamilyen integrált fejlesztői környezetet használni. Ehhez javasoljuk az MS Office **FrontPage 2003**-at, amely automatikus kódkiegészítéssel (*IntelliSense*), a kulcsszavaknál megjelenő magyarázatokkal és számos más hatékony eszközzel segíti a forráskód megírását.



VBS programfejlesztés FrontPage környezetben

Az MS Office tartalmazza a **Microsoft Script Editort**, amely szintén rendelkezik beépített hibakereső (debugger) eszközökkel. Ha a telepítésnél nem módosítottuk, akkor a Script Editort a

c:*Program Files\Microsoft Office\Office10* (MS Office XP)

illetve a

c:*Program Files\Microsoft Office\Office11* (MS Office 2003)

mappában találjuk (Office XP: *MSE7.exe*, Office 2003: *MSE7.exe*). Gyakori használata esetén célszerű parancsikonként kitenni az Asztalra.
A Script Editor használatát [3] ismerteti (373. oldal).
A Microsoft webhelyéről letölthető az ingyenes **Microsoft Windows Script Debugger**, amellyel töréspontokat helyezhetünk el a programban, futás közben lekérdezhetjük a változók értékeit, illetve lépésenként is végrehajthatjuk a kódot. A telepítőprogram megfelelő változatát (*Script Debugger Windows NT 4 and Later*) a Microsoft Download Center segítségével találjuk meg. A keresősorba írjuk be: Script Debugger.
A Debuggert részletesebben a [3] irodalom ismerteti.
A telepített fejlesztőeszközöket a böngészőből is megnyithatjuk (*Nézet/Parancsfájl-hibakereső/Megnyitás*).

VBScript dokumentáció

Magyar nyelven a [3] irodalom mutatja be a VBScript használatát. A könyv teljes egészében tárgyalja az emelt szintű érettségi követelményeinek megfelelő ismereteket. Ezen túlmenően azonban foglalkozik az objektumalapú programozással, a grafikus felhasználói felület kezelésével is.
A VBScript programozási nyelv teljes referenciáját a Microsoft Development Network (MSDN) webhelyén találjuk meg (*Web Development*).

HTML-alkalmazások (HTA)

A HTML-alkalmazások (HTA-k) olyan VBScript (vagy JavaScript) nyelven írt programok, amelyek a böngésző felügyelete nélkül, önálló ablakban futnak. Kikerülik a böngésző felhasználói felületét, de teljes egészében kihasználhatják a programozási nyelv és a dinamikus objektummodell (DOM) eszközeit, erőforrásait. Az ablak tartalmát HTML-elemek és stíluslapok (CSS) segítségével alakíthatjuk ki, a legegyszerűbb alkalmazások készítéséhez azonban nincs szükség a HTML-kód ismeretére.
A HTML-alkalmazások a böngésző ellenőrzése nélkül hozzáférnek a háttértárakhoz, használhatják az ActiveX vezérlőket, bezárhatják az ablakot stb. Nem kapunk figyelmeztető üzenetet, a böngésző nem kérdez rá az engedélyezésre.
Bármely HTML-fájl futtatható *hta* kiterjesztéssel.

Az ablak tulajdonságainak módosítása

A megjelenő ablak tulajdonságait a HTA:APPLICATION elem segítségével módosíthatjuk. Ha megfelelnek az alapértelmezett értékek, akkor nem szükséges beilleszteni a forráskódba az elemet. Az elemet a zárótag helyett a nyitótag végén elhelyezett /-jellel is lezárhatjuk:

```
<HTA:Application id = "név" tulajdonságok />
```

A tulajdonságokat sztringként kell megadni *név* = "*érték*" formában.

A tulajdonság neve	Jelentése	Alapértelmezett érték
applicationName	az ablak neve (objektumként történő hivatkozáshoz)	
border	a keret típusa (thick, dialog, thin, none)	thick
borderStyle	a keret stílusa (normal, raised, sunken, complex, static)	normal
caption	címsor engedélyezése (yes/no)	yes
contextMenu	a jobb egérgombra megjelenő menü engedélyezése (yes/no)	yes
icon	az alkalmazás ikonjának elérési útja	
innerBorder	belső keret megjelenítése (yes/no)	yes
maximizeButton	Teljes méret gomb megjelenítése (yes/no)	yes
minimizeButton	Kis méret gomb megjelenítése (yes/no)	yes
navigable	hivatkozás esetén az új dokumentum az alkalmazás ablakában jelenjen meg (yes: igen, no: új ablakban)	no
scroll	gördítősáv megjelenítése (yes, no, auto)	yes
scrollFlat	síkbeli megjelenésű gördítősáv (yes: igen, no: térhatású)	no
selection	kiválasztás engedélyezése az ablakban (yes/no)	yes
showInTaskBar	az alkalmazás megjelenítése a Tálcán (yes/no)	yes
singleInstance	az alkalmazás egyszerre csak egy példányban indítható (yes/no)	no
sysMenu	a rendszermenü engedélyezése a címsor ikonjára kattintáskor (yes/no)	yes
windowState	az ablak kezdeti mérete (normal, maximize, minimize)	normal

A HTA:Application elem tulajdonságai

A VBScript programok szerkezete

Önálló szkriptek készítése

A VBScript programok utasításait nem szükséges HTML-kódba illesztenünk. A forrásfájlban a `<script>` és `</script>` elemek között található utasítások a fájl megnyitásakor végrehajtásra kerülnek. A nyitó tagban meg kell adnunk az alkalmazott programozási nyelvet[4].

```
<script language = "VBS">
  utasítások
</script>
```

Így természetesen nem használhatjuk a grafikus felület objektumait, de erre az algoritmusok fejezet tárgyalásánál, illetve az érettségin nincs is szükség. A grafikus felület objektumaival kapcsolatban lásd a 10. osztályos informatika-tankönyv *Szövegdobozok, parancsgombok és társaik* fejezetét (119. oldal)!

A szkriptek beillesztése HTML-fájlokba

A SCRIPT-elemet HTML-fájlokba is beilleszthetjük. A szkript a HTML-kódban bárhol elhelyezkedhet. Egy fájlban több szkript szerepelhet. A nyelvet (*language*) csak az első szkriptnél kell megadni.
A szkriptet általában a HEAD-ben helyezzük el. Célszerű a HEAD-ben elhelyezni

- a globális konstansok és változók deklarációját,
- a globális változók értékadásához szükséges utasításokat,
- a dokumentum betöltésénél végrehajtásra kerülő utasításokat,
- a saját függvények és eljárások kódját.

A BODY tartalmazza

- az ablakban megjelenő objektumok kódját, szövegeket, képeket,
- azokat a szkripteket, amelyek a *document.write* segítségével a betöltésnél módosítják a HTML-kódot.

Megjegyzés
Ha felhasználjuk a HTML-kód elemeit, akkor ügyeljünk arra, hogy az utasítás végrehajtásakor már ismert legyen az objektum azonosítója. Az objektumot definiáló elem után kell elhelyezni a rá hivatkozó globális szkriptet! Mivel az eljárások és függvények utasításai csak híváskor kerülnek végrehajtásra, ezek megelőzhetik a HTML-objektumok definícióit. Így az eseménykezelő eljárásokat tartalmazó szkripteket tehetjük a HEAD-be is.

Kód csatolása a fájlhoz

A VBScript program utasításait külön szövegfájlban tárolhatjuk. A fájlban csak az utasítások szerepeljenek a SCRIPT-elem nyitó- és zárótagja nélkül! A szövegfájl kiterjesztése tetszőleges lehet. Általában a *vbs* vagy *inc* kiterjesztést használják.
Az utasításokat a *hta* vagy *htm* fájlban lévő SCRIPT-elem *src* (source) tulajdonságának megadásával csatolhatjuk a forráskódhoz:

```
<SCRIPT language = "VBS" src = "elérési út"></SCRIPT>
```

Ilyen SCRIPT-elemet több helyre is beilleszthetünk a forráskódba, de csak a nyitó- és zárótaggal ellátott szkriptek közé tehetjük (tehát a szkripteken kívül).
Készítsük el például a következő szövegfájlokat (az utasítások magyarázatát lásd később), és helyezzük el ezeket a *főprogram.hta* fájllal azonos mappában. A főprogramot dupla kattintással indíthatjuk.

értékadás.vbs fájl:

```
x=6 : y=7
```

függvény.vbs fájl:

```
function osszead(a,b)
  osszead=a+b
end function
```

kiírás.vbs fájl:

```
document.write x, " + ", y, " = ", osszead(x,y)
```

főprogram.hta fájl:

```
<script language="VBS" src="függvény.vbs"></script>
<script src="értékadás.vbs"></script>
<script src="kiírás.vbs"></script>
```

Az egyes szövegfájlokban deklarált változók globálisnak számítanak[5], és a szövegfájlokban definiált függvényeket, illetve eljárásokat is meghívhatjuk a *hta* fájl forráskódjában.

[4] Az alapértelmezett programozási nyelv a JavaScript.
[5] Kivéve természetesen a függvények, eljárások lokális változóit.

A VBScript elemei

A VBScript szintaxisa

Minden egyes utasítást külön sorba kell írni. Hosszú sorok esetén egy szóköz és egy aláhúzásjel után a következő sorban folytatható a kód:

```
utasítás _
  az utasítás folytatása
```

Szükség esetén egy sorba írhatunk több utasítást is, ekkor kettősponttal kell őket egymástól elválasztani:

```
utasítás1 : utasítás2
```

Ezt a lehetőséget azonban nem célszerű alkalmazni.
A VBScript a sztringek kivételével nem különbözteti meg a kisbetűket és a nagybetűket egymástól.

Megjegyzések a forráskódban

A megjegyzéseket aposztrófjel (') után írjuk. Az interpreter az aposztrófjeltől a sor végéig terjedő részt figyelmen kívül hagyja.

```
utasítás ' megjegyzés
```

Az aposztrófjel természetesen a sor elején is állhat.
Az aposztrófjel helyett használhatjuk a *Rem* kulcsszót (remark). Ez azonban utasításnak számít, tehát a sor elején (vagy kettőspont után) kell elhelyezkednie.

```
rem megjegyzés
utasítás : rem megjegyzés
```

Azonosítók a VBScript-ben

A program elemeit (változók, konstansok, függvények, eljárások) azonosítóval látjuk el. Két különböző elemnek nem adhatjuk ugyanazt az azonosítót.
Az azonosító az angol ábécé betűiből, számokból illetve aláhúzásjelből állhat. Betűvel kell kezdődnie, és legfeljebb 255 karaktert tartalmazhat. Nem használhatjuk azonosítóként a VBScript kulcsszavait.
Az interpreter az azonosítókban nem különbözteti meg egymástól a kisbetűket és a nagybetűket.

Számok

A számokat a matematikában szokásos formában adjuk meg. **Törtszámok esetén a forráskódban tizedespontot használunk, az input/outputnál azonban a területi beállításoknál megadott tizedesjel (például tizedesvessző) szerepel!**
Egész számoknál használhatjuk a tizenhatos számrendszert (például a színkódok esetén). A hexadeximális értéket a szám elé írt *&h* karakterekkel jelezzük, például: `&hffaa66`.
A valós számokat megadhatjuk hatványkitevős alakban is, például: 34.5E-6 = $34.5 \cdot 10^{-6}$.

Sztringek

A karaktersorozatokat (sztringeket) idézőjelek közé írjuk. A sztringen belül szükség esetén aposztrófjelet (') alkalmazunk. Ha ragaszkodunk az idézőjelhez, akkor duplán kell kiírnunk.

```
document.write "Idézőjel 'használata' az idézőjelen belül."
document.write "Idézőjel ""használata"" az idézőjelen belül."
```

Az üres sztringet két idézőjel jelzi (szóköz nélkül): `""`.
A VBScript nem ismeri külön a karakter típust, helyette egyetlen karakterből álló sztringet használunk, például: `"a"`.
A forráskódban szereplő sztringeket nem választhatjuk el két sorba!
A sztringek összehasonlítását a magyar ábécének megfelelő sorrend szerint az StrComp függvénnyel végezhetjük el.

Logikai változók

A logikai változók értéke *True* (igaz) vagy *False* (hamis) lehet. A VBScript az igaz logikai értéket -1-nek, a hamisat pedig 0-nak tekinti, így aritmetikai kifejezésekben és relációkban is szerepelhetnek. Például: *true* + 10 = 9, *false* > 0.5.

Dátumok

A dátum/idő típusú értékeket kettőskeresztek (#) közé kell tenni. A területi beállításoknál megadott évszámoknál elhagyható az első két számjegy (az évszázad). A részek közé a dátumoknál / jelet vagy kötőjelet (-), az időpontoknál kettőspontot (:) írunk. Az időpontoknál használható az *am* (délelőtt) vagy *pm* (délután) rövidítés. A dátum és az időpont közé szóköz kerül, de bármelyikük el is hagyható. A vezető nullákat egyik résznél sem kell kiírni.

```
#1996/05/11#          #1996-5-11#          #96-05-11#
#07:20pm#             #14:5:11#
```

Az év/hónap/nap sorrendet a területi beállítások szabályozzák.

A VBScript kulcsszavai

Abs
And
Array
Asc
Atn
Call
Case
CBool
CByte
CCur
CDate
CDbl
Chr
CInt
Class
Clear
CLng
Const
Cos
CreateObject
CSng
CStr
Date
DateAdd
DateDiff
DatePart
DateSerial
DateValue
Day
Description
Dictionary
Dim
Do
Each
Else
Empty
Eqv
Erase
Err
Error
Eval
Execute
Exit
Exp
Explicit
False
FileSystemObject
Filter
FirstIndex
Fix
For
FormatCurrency
FormatDateTime
FormatNumber
FormatPercent
Function
GetObject
GetRef
Global
Hex
HelpContext
HelpFile
Hour
If
IgnoreCase
Imp
Initialize
InputBox
InStr
InStrRev
Int
Is
IsArray
IsDate
IsEmpty
IsNull
IsNumeric
IsObject
Join
LBound
LCase
Left
Len
Length
LoadPicture
Log
Loop
LTrim
Match
Matches
Mid
Minute
Mod
Month
MonthName
MsgBox
Next
Not
Now
Nothing
Null
Number
Oct
On
Option
Or
Pattern
Private
PropertyGet
PropertyLet
PropertySet
Public
Raise
Randomize
ReDim
RegExp
Rem
Replace
RGB
Right
Rnd
Round
RTrim
ScriptEngine
ScriptEngineBuildVersion
ScriptEngineMajorVersion
ScriptEngineMinorVersion
Second
Select
Set
Sgn
Sin
Source
Space
Split
Sqr
StrComp
String
StrReverse
Sub
Tan
Terminate
Test
Time
Timer
TimeSerial
TimeValue
Then
Trim
True
TypeName
UBound
UCase
Value
VarType
Weekday
WeekdayName
Wend
While
With
Xor
Year

Változók

A változók deklarálása

A változókat nem kötelező deklarálni, és nem lehet megadni a típusukat. A változók deklarálása:

```
dim változónév1, változónév2, …
```

A *dim* utasítás tetszőleges számszor, a szkriptek tetszőleges helyén előfordulhat. A szkriptek egy fájlban összesen 127 globális változót tartalmazhatnak. (Ez a korlátozás természetesen nem vonatkozik a függvények és eljárások lokális változóira.)
A deklarálást a nehezen felderíthető hibák elkerülése miatt az *option explicit* utasítással kötelezővé tehetjük. Az utasítást a szkript elejére kell írni.

```
option explicit
utasítások
```

Megjegyzés
Az *option explicit* utasítás nem szerepelhet eseménykezelő szkriptben[6].

A változók típusa

A változók típusát nem lehet megadni a deklarációban. A típust az első értékadáskor az interpreter határozza meg. Ezt típuskonverziós függvényekkel módosíthatjuk.
Megjegyzés
A VBScript egy általános, *variant* típust rendel a változókhoz, amely különféle altípusok tárolására használható.

A változók alapértelmezett értéke

Ha egy kifejezésben olyan változót használunk fel, amely még nem kapott kezdőértéket, akkor az interpreter automatikusan a következő értéket rendeli hozzá:

numerikus kifejezésben:	0
sztringekkel végzett műveleteknél:	"" (üres sztring)
logikai műveletek esetén:	hamis (*false*, 0)

A deklaráció és az első értékadás között a változó értéke *Empty* (üres). Az alapértelmezett érték miatt az *Empty*-t csak az *IsEmpty* függvénnyel vizsgálhatjuk meg.

Az elemi típusok értékkészlete

Az alábbiakban megadjuk az általános *variant* típus altípusainak értékkészletét.

Altípus:	**Megnevezés:**	**Tartomány:**
boolean	logikai	true (-1) vagy false (0)
string	karakterlánc	legfeljebb ~kétmilliárd karakter
date	dátum	100. január 1. és 9999. december 31. között
time	idő	0:00:00-tól 23:59:59-ig
Egész típusú változók:		
byte	bájt	0-tól 255-ig
integer	egész	-32 768-tól +32 767-ig
long	hosszú egész	-2 147 483 648-tól +2 147 483 647-ig
currency	pénznem	-922 337 203 685 477,5808-tól -922 337 203 685 477,5807-ig
Valós típusú változók:		
single	egyszeres pontosságú valós	-3,402823E38-tól -1,401298E-45-ig a negatív és +1,401298E-45-től +3,402823E38-ig a pozitív számokra, illetve 0
double	dupla pontosságú valós	-1,79769313486232E308-tól -4,94065645841247E-324-ig a negatív és +4,94065645841247E-324-től +1,79769313486232E308-ig a pozitív számokra, illetve 0

[6] <script EVENT=*esemény* FOR=*objektum*>

A pénznem típusú változókat az egészekhez soroltuk, mert a VBScript a 10000-szeresüket tárolja egész számként. Így a számítás 4 tizedesjegy pontosságú.

Automatikus típuskonverzió

Az interpreter a kifejezések kiértékelésénél automatikusan elvégzi a szükséges típuskonverziót. Az eredmény típusát a kifejezésben szereplő operátorok típusa, illetve a kifejezés várt értéke dönti el (értékadó utasítás, reláció stb.).
Mivel sztringeket a + jellel is összefűzhetünk (lásd: sztringek összefűzése), a beolvasott értékekkel végzett műveletek nem várt eredményre vezethetnek. Például a következő utasítások esetén:

```
x = inputbox("x = ")
y = inputbox("y = ")
w = x + y
z = x * y
```

a *w*-be a két beolvasott érték összefűzése (konkatenációja), a *z*-be viszont a szorzata kerül!

A változók hatóköre

A függvények és eljárások definícióján kívül elhelyezkedő utasítások alkotják a globális szkriptet. A globális szkriptben deklarált változók (illetve konstansok) az egész dokumentumra nézve globálisak. Bármely szkriptben, bármely függvényben vagy eljárásban hivatkozhatunk rájuk (ha nem deklarálunk ugyanolyan nevű változót).
A függvényekben/eljárásokban deklarált változók az adott függvényre/eljárásra nézve lokálisak. Csak a függvényen/eljáráson belül hivatkozhatunk rájuk. A függvény/eljárás befejezésekor megszűnnek, értékük eltűnik.

Konstansok

Konstansok deklarálása

`const` *`név = kifejezés, név = kifejezés, …`*

A kifejezés nem tartalmazhat változókat, a felhasználó által definiált vagy beépített VBScript függvényeket.
A függvényekben/eljárásokban deklarált konstansok lokálisak az adott függvényre/eljárásra nézve.

Beépített konstansok

A beépített konstansok azonosítója vb-vel kezdődik. A beépített konstansokat deklarálás nélkül használhatjuk a kifejezésekben. Az alábbiakban csak a leggyakrabban előforduló értékeket adjuk meg.

MsgBox konstansok

Konstans	Érték	Gombok, ikonok
Parancsgombok		
vbOKOnly	0	OK
vbOKCancel	1	OK, Mégse
vbAbortRetryIgnore	2	Kilépés (Megszakítás, Leállítás), Ismét (Újra), Tovább (Kihagyás)
vbYesNoCancel	3	Igen, Nem, Mégse
vbYesNo	4	Igen, Nem
vbRetryCancel	5	Ismét (Újra), Mégse
Ikonok		
vbCritical	16	kritikus üzenet (piros kör fehér x-szel)
vbQuestion	32	kérdőjel
vbExclamation	48	felkiáltójel
vbInformation	64	információ (i betű)
Visszatérési érték		
vbOK	1	OK
vbCancel	2	Mégse (ESC)
vbAbort	3	Kilépés (Megszakítás, Leállítás)
vbRetry	4	Ismét (Újra)
vbIgnore	5	Tovább (Kihagyás)
vbYes	6	Igen
vbNo	7	Nem

Használatukat lásd: Kiírás az *MsgBox* függvénnyel.

Sztring konstansok

Konstans	Érték	Jelentés
vbNewLine	Chr(10)+Chr(13)	Új sor
vbTab	Chr(9)	Tabulátor

Színkonstansok

Konstans	Szín	RGB-kód (hex)
vbBlack	fekete	&h00
vbRed	vörös	&hFF
vbGreen	zöld	&hFF00
vbYellow	sárga	&hFFFF
vbBlue	kék	&hFFFFFF
vbMagenta	bíbor	&hFF00FF
vbCyan	türkizkék (cián)	&hFFFF00
vbWhite	fehér	&hFFFFFF

Kifejezések

Operátorok

Aritmetikai operátorok

Az aritmetikai operátorok a precedencia sorrendjében:

– ellentettképzés (legmagasabb precedenciájú, például: $-3^2 = +9$!)
^ hatványozás (az alap csak egész kitevő esetén lehet negatív)
* / szorzás, osztás
\ egész osztás hányadosa
Mod egész osztás maradéka
+ – összeadás, kivonás
& sztringek összefűzése.

Bár a sztringek összefűzése nem aritmetikai művelet, az automatikus típuskonverzió miatt szerepel a felsorolásban. Például: `"5" & 3 + 4 = 57`.

Logikai operátorok

A logikai operátorok a precedencia sorrendjében:

Not „nem” művelet
And „és” művelet
Or „vagy” művelet
Xor „kizáró vagy” művelet
Eqv ekvivalencia
Imp implikáció

A logikai műveletek két egész típusú numerikus érték között bitenként kerülnek végrehajtásra.
A VBScript nem ismeri a logikai kifejezések rövidre zárását. Egy kifejezést akkor is teljes egészében kiértékel, ha a részkifejezésből már lehet tudni az eredményt. Így például futási hibához vezet egy *n* maximális indexű tömb esetén a

```
do while i<=n and a(i)<>…
  …
  i=i+1
loop
```

ciklus. A helyes megoldást lásd: Az indexhatár túllépése fejezetben.

Relációs operátorok

A relációs operátorok azonos precedenciával rendelkeznek!

= egyenlő
<> nem egyenlő
< kisebb
> nagyobb
<= kisebb vagy egyenlő
>= nagyobb vagy egyenlő

A sztringek összehasonlítását a magyar ábécének megfelelő sorrend szerint az StrComp függvénnyel végezhetjük el. Bármely numerikus érték kisebb, mint egy sztring.

Megjegyzés
Két objektumváltozó azonosságát az *Is* operátorral ellenőrizhetjük (ugyanarra az objektumra mutatnak-e).

Sztringek összefűzése

Sztringeket az & operátor segítségével fűzünk össze. Az automatikus típuskonverzió miatt sztringeket és numerikus értékeket szintén összefűzhetünk. Például: `"5" & 3 = 53`.
Bár kompatibilitási okokból az összefűzésre a + jel is használható, kerüljük el az alkalmazását!

Az operátorok precedenciája

Az operátorokat a relációs operátorok kivételével a precedencia sorrendjében soroltuk fel. Az azonos precedenciájú műveleteket az interpreter balról jobbra végzi el. Ha egy kifejezés többféle operátort tartalmaz, akkor először az aritmetikai, aztán a relációs, majd a logikai operátorok kerülnek kiértékelésre. A sorrenden zárójelek segítségével változtathatunk.

Műveletek dátumokkal

A VBScript a dátum/idő típusú változóban az 1899. december 30. óta eltelt napokat tárolja (az előtte lévő dátumok értéke negatív). Az időpontot a nap törtrészében fejezi ki.
Két dátum különbsége a köztük eltelt napok számával egyenlő. Ha egy dátumhoz számot adunk hozzá, akkor annyi nappal megnöveljük (negatív szám esetén csökkentjük) az értékét. A különböző időintervallumokkal (például hét, hónap) történő módosítást lásd a DateAdd függvény ismertetésénél.

Utasítások

Az alábbi utasításokat a hivatkozásnak megfelelő helyen tárgyaljuk:
call, const, dim, erase, (dinamikus tömböknél: erase), function, on error, option explicit, redim, rem, set, sub.

Értékadó utasítás

Az értékadó utasítás jele az egyenlőségjel (=).

Beolvasás a billentyűzetről

Beolvasás az InputBox*-szal*

Billentyűzetről adatokat az *InputBox* függvény segítségével adhatunk meg a programnak. Az *InputBox* függvény, ezért csak kifejezésben (értékadó utasításban) szerepelhet.

```
inputbox(üzenet[, ablakcím][, kezdőérték][, xhely][, yhely])
```

üzenet:	az *InputBox* szövege. Soremelés a *vbNewLine* beépített konstanssal.
cím:	az *InputBox* ablak címe,
kezdőérték:	a szövegdobozban megjelenő sztring,
xhely:	vízszintes pozíció twipben[7],
yhely:	függőleges pozíció twipben.

Például:

```
nev = inputbox("Írja be a nevét!", "Beolvasás", "Ide írja a nevet!")
```

Az *InputBox* visszatérési értéke sztring. Szükség esetén a típuskonverziós függvényekkel numerikus értékké kell átalakítani. Például:

```
szam = cint(inputbox("Írjon be egy egész számot!", , 0))
```

Lásd még: automatikus típuskonverzió.
Az *InputBox* visszatérési értéke

"" (üres sztring)	ha a felhasználó törölte a kezdőértéket, és az *OK* gombra kattintott;
"" (üres sztring)	ha nem volt kezdőérték, és a felhasználó az *OK* gombra kattintott (de nem írt be semmit);
Empty	ha a felhasználó a *Mégse* vagy a *Bezárás* gombra kattintott.

Beolvasás szövegdobozokkal

A grafikus felhasználói felület objektumaival kényelmesebb adatbevitel érhető el (lásd a 10. osztályos informatika tankönyv *Szövegdobozok, parancsgombok és társaik* fejezetét a 119. oldalon). A szövegdoboz tartalmát az objektum *value* tulajdonságával érhetjük el. A beolvasás egy parancsgomb eseménykezelő eljárásának meghívásakor történik meg.

[7] 1 twip = 1/15 pixel, de függhet az operációs rendszer beállításaitól, például a betűmérettől.

Kiírás a képernyőre

Kiírás a document.write eljárással

```
document.write sztringkifejezés, sztringkifejezés, …
```

A sztring tartalmazhat HTML-kódot is. Soremelést a `<br>` elem beillesztésével hozunk létre. Például:

```
document.write "Első sor. <br>"
document.write "Második sor."
```

vagy:

```
document.write "Első sor. <br> Második sor."
```

Változó értékének megjelenítése (automatikus típuskonverzió!) például:

```
document.write "x = " & x & "<br>"
```

vagy egyszerűen:

```
document.write "x = ", x, "<br>"
```

Megjegyzés
A *document.write*-ot nem használhatjuk eseménykezelő eljárásokban, mert törli a dokumentum HTML-kódját!
A *document.write* a *document* objektum *write* metódusa.

Kiírás az MsgBox *függvénnyel*

```
msgbox(üzenet[, gombok][, cím])
```

üzenet: az *MsgBox*-ban megjelenő szöveg (soremelés: *vbNewLine*),
cím: az *MsgBox* ablak címe.

A *gombok* paraméter az ablakban megjelenő gombokat és ikonokat szabályozza. A hozzájuk tartozó konstansokat össze kell adni. Például:

```
valasz = msgbox("Tovább?", vbYesNoCancel+vbQuestion), vagy:
valasz = msgbox("Tovább?", 3+32)
```

A függvény visszatérési értékének megfelelő konstansokat lásd az MsgBox konstansoknál.
Ha nincs szükségünk a visszatérési értékre, akkor az *MsgBox*-ot eljárásként is meghívhatjuk:

```
msgbox "Kész.", vbokonly+vbinformation
```

Változó értékének kiírása az *MsgBox* függvénnyel (automatikus típuskonverzió!):

```
msgbox "x = " & x, vbokonly+vbinformation
```

Kiírás HTML-elemekkel

A kiíráshoz bármelyik, szöveg megjelenítésére alkalmas HTML-elemet felhasználhatjuk (lásd a 10. osztályos informatika tankönyv *Szövegdobozok, parancsgombok és társaik* fejezetét a 119. oldalon). Ilyen elem a bekezdés (P), a DIV, a SPAN stb. Az *innerText* tulajdonság csak formázatlan szöveg, az *innerHTML* tulajdonság tetszőleges HTML-elem beillesztésére alkalmas.

Feltételes elágazások

IF…THEN…ELSE

```
if feltétel then
  [utasítások]
[elseif feltétel
  [utasítások]]
  …
[else
  [utasítások]]
end if
```

Ha egy feltétel teljesül, akkor végrehajtja a hozzá tartozó utasításokat. Ekkor a hátralévő rész (*elseif*, *else*) nem kerül kiértékelésre. Az utasítás a Pascaltól eltérően valódi többágú szelekciót valósít meg.
Az *elseif* ág tetszőleges számszor ismétlődhet. Az *else* ág elhagyható.
Figyeljünk arra, hogy az *elseif*-et egybe kell írni, az *end if*-et viszont külön.
Az IF utasítások egymásba ágyazhatók.

SELECT CASE

```
select case tesztkifejezés
  [case kifejezéslista
    [utasítások]]
  …
  [case else
    [utasítások]]
end select
```

A tesztkifejezés értékétől függően hajtja végre az utasításokat. A kifejezéslista egymástól vesszővel elválasztott kifejezéseket (értékeket) tartalmazhat. Csak az elsőként megfelelő *case* ág utasításai kerülnek végrehajtásra.
A s*elect case* utasítások egymásba ágyazhatók.

Ciklusok

Számlálós ciklus

```
for számláló = kezdőérték to végérték [step lépésköz]
  [utasítások]
  [exit for]
  [utasítások]
next
```

Megismétli a ciklusmag utasításait. Ha nem adjuk meg a *lépésköz* értékét, akkor egyesével lépteti a ciklusszámlálót. A *lépésköz* lehet törtszám, illetve negatív érték is (visszafelé számlál). Az ismétlés feltétele:

lépésköz ≥ 0 ismétlés, ha *számláló* ≤ *végérték*
lépésköz < 0 ismétlés, ha *számláló* ≥ *végérték*

A *for…next* utasítások egymásba ágyazhatók, de egymástól különböző ciklusszámlálókat kell alkalmazni.

Ciklus iterátorral

```
for each elem in tömb
  [utasítások]
  [exit for]
  [utasítások]
next
```

A tömb összes elemére végrehajtja a ciklust. A ciklusban az *elem* változóval hivatkozhatunk a tömbelemekre.
Megjegyzés
A *for each* utasítást alkalmazhatjuk az objektumokból álló kollekciók elemeire is.

Feltételes ciklus

Elöltesztelő ciklus ismétlési feltétellel:

```
do while feltétel
  [utasítások]
  [exit do]
  [utasítások]
loop
```

Hátultesztelő ciklus ismétlési feltétellel:

```
do
  [utasítások]
  [exit do]
  [utasítások]
loop while feltétel
```

Elöltesztelő ciklus kilépési feltétellel:

```
do until feltétel
  [utasítások]
  [exit do]
  [utasítások]
loop
```

Hátultesztelő ciklus kilépési feltétellel:

```
do
  [utasítások]
  [exit do]
  [utasítások]
loop until feltétel
```

A *feltétel* teljesülésétől függően hajtja végre a ciklust. A *while* az ismétlési feltételt (amíg), az *until* pedig a kilépési feltételt (mígnem) jelzi.

Folyamatjelző az állapotsorban

Az interpreter csak a szkript végrehajtása után jeleníti meg az outputot az ablakban. A végrehajtás során az állapotsor tartalmát használhatjuk állapotjelzésre. Az állapotsor szövege a *window.status*[8] értékadásával adható meg.
A következő ciklus például kiírja a ciklusváltozó értékét az állapotsorba:

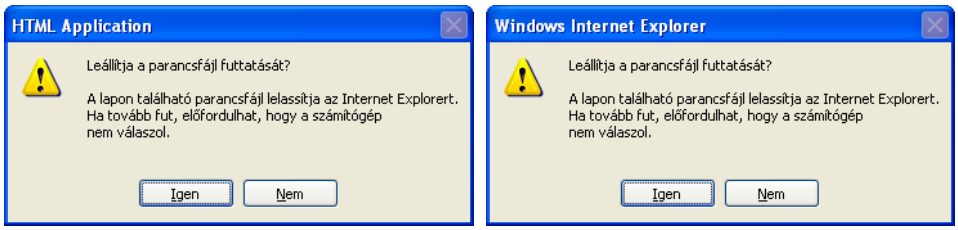
```
for i= 1 to 10000
  window.status=i
next
```

Az állapotsor használata jelentős mértékben lelassítja a program futását.

[8] A *window* objektum *status* tulajdonsága.

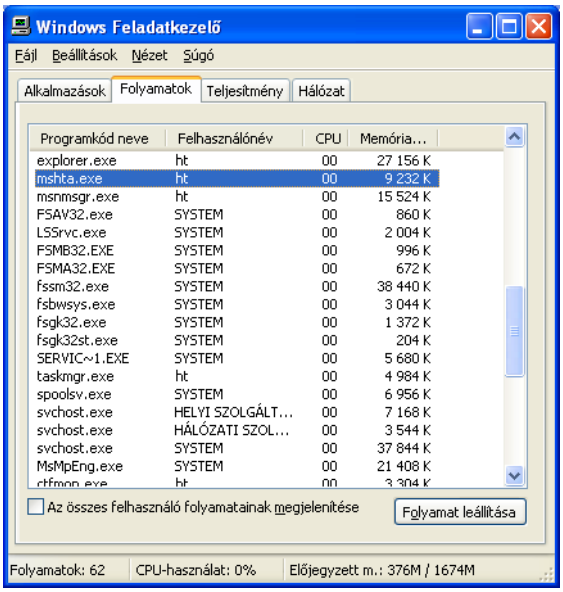
Hosszú ciklusok leállítása

Hosszú ciklusok esetén megtörténhet, hogy az interpreter figyelmeztet a végrehajtás elhúzódására. A program futtatása a *Nem* gombra kattintással folytatható.



Üzenet hosszú ideig futó ciklus esetén (hta, illetve htm)

Ha a program még az ablak megjelenítése előtt végtelen ciklusba kerül, akkor a *Feladatkezelőben* az *mshta* folyamatot kell leállítanunk:



Az mshta.exe a Feladatkezelőben

Egyéb utasítások

DEBUG.WRITE

`debug.write` *`sztring1, sztring2, …`*

Elküldi a debuggernek a sztringeket. Csak hibakereső üzemmódban van hatása.

DEBUG.WRITELN

`debug.writeln` *`sztring1, sztring2, …`*

Soremeléssel küldi el a debuggernek a sztringeket. Csak hibakereső üzemmódban van hatása.

EXECUTE

`execute` *`sztringkifejezés`*

Végrehajtja a sztringként megadott utasításokat. Több utasítást kettősponttal kell elválasztani egymástól. Az "x=y" sztringet az *execute* értékadó utasításként, az *eval* függvény pedig relációként értelmezi.

RANDOMIZE

`randomize`

Inicializálja a véletlenszám-generátort (rnd). Használata nélkül az *rnd* mindig ugyanazt a véletlenszám-sorozatot állítja elő.

STOP

`stop`

Felfüggeszti a program végrehajtását. Debugger program telepítése esetén átadja a vezérlést a hibakeresőnek, így hatása megegyezik egy töréspont elhelyezésével.

WITH

```
with objektumnév
  utasítások
end with
```

Az objektum tulajdonságaira és metódusaira az objektumnév elhagyásával hivatkozhatunk.
A *With* utasítások egymásba ágyazása esetén a belső blokkban ki kell írni a külső blokk által hivatkozott objektum nevét.

Függvények és eljárások

Eljárások definiálása és hívása

```
sub[9] név[(paraméter1, paraméter2, …)]
  [utasítások]
  [exit sub]
  [utasítások]
end sub
```

A paraméterek típusával kapcsolatban lásd: A függvények és eljárások paraméterei.
Az eljárásban deklarált változók lokálisak az eljárásra nézve.
Eljárást csak globális szkriptben definiálhatunk (másik eljárásban vagy függvényben nem).
Egy eljárást kétféleképpen hívhatunk meg:

- `eljárásnév paraméter1, paraméter2, …`
- `call eljárásnév(paraméter1, paraméter2, …)`

A paramétereket az első esetben nem szabad zárójelbe tenni, a második esetben kötelező zárójelbe tenni.

Függvények definiálása és hívása

```
function név[(paraméter1, paraméter2, …)]
  [utasítások]
  [név = kifejezés]
  [exit function]
  [utasítások]
  [név = kifejezés]
end function
```

A paraméterek típusával kapcsolatban lásd: A függvények és eljárások paraméterei.
A függvényben deklarált változók lokálisak a függvényre nézve.
Függvényt csak globális szkriptben definiálhatunk (másik függvényben vagy eljárásban nem).
Ha a függvénytörzsben nem szerepel a függvénynévnek értéket adó utasítás, akkor a visszatérési érték numerikus esetben 0, sztringek esetén pedig üres sztring ("").
A függvény hívása a függvény nevével és zárójelben az aktuális paraméterek felsorolásával történik.
A függvényeket az eljáráshívásnak megfelelő szintaxissal eljárásként is meghívhatjuk:

- `függvénynév paraméter1, paraméter2, …`
- `call függvénynév(paraméter1, paraméter2, …)`

Ekkor az interpreter figyelmen kívül hagyja a visszatérési értéket (lásd például az msgbox használatát).

A függvények és eljárások paraméterei

Érték szerinti paraméterátadás: `byval változónév`
Cím szerinti paraméterátadás: `byref változónév`
Az alapértelmezett mód a cím szerinti paraméterátadás, így a *byref* kulcsszó elhagyható. A *byval* kulcsszót minden érték szerint hívott változó elé ki kell tenni a paraméterlistában.
A függvénynek/eljárásnak úgy adunk át tömböt, hogy a tömb nevét zárójelek és indexek nélkül írjuk a paraméterlistába.
Az alprogramon belül a tömbelemek számát az UBound függvénnyel határozhatjuk meg.
Az interpreter a hatékonyság növelése miatt a kiértékelés előtt átrendezheti a kifejezéseket. Ezért nem várt eredménnyel járhat, ha egy kifejezésben meghívott függvény cím szerint átadott paraméterei között szerepelnek a kifejezésben lévő változók.

Rekurzív alprogramok

A függvényekben és eljárásokban használhatunk rekurziót.

[9] A szubrutin (subroutine) rövidítése.

Összetett adatszerkezetek

Tömbök

Tömbök deklarálása

`dim tömbnév(maxidex, maxindex, …)`

A tömb indexelése mindig 0-val kezdődik. A deklarációban a maximális index értékét kell megadni. Így egy egydimenziós tömb elemeinek a száma: *maxindex* + 1. A maximális index értékét az UBound függvénnyel kérdezhetjük le.
Maximálisan 60 dimenziós tömböket deklarálhatunk.
A tömbelemek tetszőleges (nem feltétlenül egyforma) típusú változót tartalmazhatnak.
A hivatkozásnál a tömb indexét kerek zárójelbe tesszük. Ha indexként törtszámot adunk meg, akkor az interpreter a legközelebbi egészre kerekíti.
A tömb összes elemére vonatkozó ciklust iterátorral egyszerűsíthetjük.

A tömbelemek törlése

`erase tömbnév`

Az *erase* utasítás numerikus értékek esetén 0-val, sztringek esetén üres sztringgel ("") tölti fel a tömböt.
Megjegyzés
Objektumhivatkozásokat tartalmazó tömb esetén a tömbelemek értéke *Nothing* lesz.

Tömböt tartalmazó változó

Egy változó tömböt is felvehet értékként. A tömb elemeire ugyanúgy hivatkozunk, mint a tömböknél. Például:

```
dim a(2), b
a(0)=0 : a(1)=1 : a(2)=2
b=a
msgbox b(0) & b(1) & b(2)
```

Értékadás több dimenziós tömböt tartalmazó változónak például:

```
dim a(2, 2), b
a(1, 1)=5
b=a
msgbox b(1, 1)
```

Ha egy tömbelem kap értékként tömböt, akkor a két tömb indexét külön zárójelpárok közé kell írni. Például:

```
dim a(2, 2), b(3)
a(0, 1)=5
b(2)=a
msgbox b(2)(0, 1)
```

Értékadás az Array függvénnyel

A tömböt tartalmazó változónak az *Array* függvénnyel adhatunk egyszerűen értéket.

`array(érték, érték, …)`

Például:

```
dim a
a=array(0, 1, 2)
msgbox a(0) & a(1) & a(2)
```

Tömbelem esetén például:

```
dim a(2)
a(1)=array(0, 1, 2)
msgbox a(1)(0) & a(1)(1) & a(1)(2)
```

Több dimenzió esetén például:

```
dim a
a=array(array(0, 1, 2), array(3, 4, 5))
msgbox a(1)(2)
```

Lásd még: IsArray függvény.

Dinamikus tömbök

A VBScript lehetővé teszi dinamikus tömbök használatát, melyek mérete (és dimenziószáma) a futtatás során módosítható. A dinamikus tömb törölhető is, így a helye felszabadul a memóriában.

Dinamikus tömb deklarálása

A deklarációban nem adjuk meg a maximális index értékét, csak üres zárójelpárt használunk:

```
dim tomb()
```

Dinamikus tömb használata

Mielőtt még hivatkozunk a tömbre (tömbelemekre), a *ReDim* utasítással meg kell adnunk a tömb méretét:

```
redim tomb(maxindex)
```

A *maxindex* tetszőleges kifejezés lehet, amely nem negatív számot eredményez. Törtszám esetén az interpreter kerekít.
A *ReDim* utasítással a dimenziószámot is megváltoztathatjuk, például:

```
dim tomb()
redim tomb(6, 7)
...
redim tomb(4)
```

Ha fel akarjuk szabadítani a tömb által elfoglalt memóriát, akkor –1-et adjunk meg paraméternek:

```
redim tomb(-1)
```

Ekkor a tömb csak egy újabb *ReDim* utasítás végrehajtása után használható.
A *ReDim* utasítás törli a tömb elemeit! Ha meg akarjuk őrizni az addigi értékeket, akkor adjuk meg az utasításban a *Preserve* kulcsszót:

```
redim preserve tomb(maxindex)
```

A tömbméret csökkentésekor természetesen mindenképpen veszítünk elemeket.
A *Preserve* használata esetén nem változtathatjuk meg a dimenziószámot, és egy többdimenziós tömbnél csak az utolsó dimenzió mérete módosítható, például:

```
redim tomb(4, 5, 6)
...
redim preserve(4, 5, 10)
```

A tömb teljes egészében az *Erase* utasítással törölhető a memóriából:

```
erase tomb
```

A tömb neve után nem írhatunk zárójeleket. Az *Erase* utasítással törölt tömbre többé nem adhatjuk ki a *ReDim* utasítást.

Objektumok

A Visual Basic Script objektumalapú programozási nyelv. Létre lehet hozni objektumokat, de nem ismeri az öröklést. A VBScript kódokban elérhetjük a weblapot alkotó objektumok tulajdonságait és metódusait is (Dokumentum Objektum Modell). Ez utóbbiak részletes ismertetését megtaláljuk a Microsoft Development Network webhelyén.
Az objektumok létrehozásának, kezelésének módját a [3] irodalomban olvashatjuk. Itt csak azokat az ismereteket foglaljuk össze, melyekre a továbbiakban szükségünk lesz.

Objektumosztályok definiálása

Az objektumosztály definícióját a globális szkriptben kell elhelyezni. A tulajdonságokat (attribútumokat, változókat) és metódusokat a *Class … End Class* utasítások közé zárjuk:

```
class osztálynév
  változók deklarálása
  metódusok definiálása
end class
```

Külön korlátozás nélkül az osztályok változói nyilvános elérésűek (*Public*).

Objektumok létrehozása és megszüntetése

Új objektumot általában a *New* operátorral hozunk létre, és a *Set* utasítással rendeljük hozzá egy változóhoz:

```
dim változónév
set változónév = new osztálynév
```

Az utasításokban szereplő változó úgynevezett objektum-hivatkozás (mutató), tehát az objektum memóriabeli helyére mutat. Így ugyanazon objektumhoz több hivatkozást is hozzárendelhetünk.
Az objektumot úgy szüntetjük meg, hogy a *Nothing* értéket rendeljük az objektum-hivatkozáshoz:

```
set változónév = nothing
```

Ha egy objektumra több változó hivatkozik, akkor az objektum csak akkor szűnik meg, ha már egyetlen hivatkozás sem mutat rá.

A *New* operátor helyett objektumokat létrehozó eljárások, metódusok is szerepelhetnek a *Set* utasításban (lásd később).

Hivatkozás az objektumok tulajdonságaira és metódusaira

A tulajdonságokra (attribútumokra) és metódusokra az objektumváltozó nevével minősítve hivatkozunk:

```
változónév.tulajdonságnév
változónév.metódusnév(paraméter, paraméter, …)
```

A hivatkozást a *With … End With* utasítással egyszerűsíthetjük:

```
with változónév
  .tulajdonságnév = …
  utasítások
end with
```

A *With*-ben megadott objektum-hivatkozást a *With* és az *End With* között nem kell kiírni a tulajdonságok és metódusok elé (a pontot viszont nem hagyhatjuk el).
A *With* utasításokat egymásba ágyazhatjuk, de mindig csak a legutoljára megnyitott (és le nem zárt) *With* változója hagyható el a minősítésből.

Kollekciók

Kollekcióknak nevezzük az objektumokat tartalmazó speciális tömböket. A kollekció maga is objektum, rendelkezik tulajdonságokkal és metódusokkal. A legtöbb kollekciónál megtaláljuk például a *Length* tulajdonságot, amely megadja a kollekció elemeinek (a benne lévő objektumoknak) a számát:

```
kollekciónév.length
```

A kollekció elemeire (a benne lévő objektumokra) azonosítójukkal vagy indexükkel hivatkozhatunk:

```
kollekciónév(index)
```

A kollekció elemeinek indexelése 0-tól *Length*–1-ig tart.
A kollekció elemeit kényelmesen feldolgozhatjuk a *For Each* ciklussal.
Kollekciók segítségével hatékonyan kezelhetjük a HTML-kód objektumait (lásd a [3] irodalomban).
A kollekciók tulajdonságait és metódusait a fájlkezelésnél ismertetjük.

Asszociatív tömbök

Az asszociatív tömbök elemeit név–adat párok alkotják, ahol a név minden párra egyedi érték a tömbön belül (kulcs). Két különböző elemnél nem lehet azonos a kulcs értéke. Egyébként a nevet és az adatot tetszőleges típusú változók, akár tömbök is alkothatják.

Megjegyzés
Az alábbiakban csak nagyon röviden tekintjük át az asszociatív tömbökre vonatkozó tudnivalókat. Az érdeklődők figyelmébe ajánljuk a [3] irodalmat, amely részletes példákon keresztül mutatja be a szótárobjektumok használatát.

Asszociatív tömb létrehozása

Asszociatív tömböt a *Dictionary* (szótár) objektummal hozhatunk létre. A tömbre, mint objektumra egy objektumváltozóval hivatkozhatunk.

```
set tömbnév = createobject("scripting.dictionary")
```

Asszociatív tömbök kezelése

Asszociatív tömböt az *Add* metódussal bővíthetünk. Paraméterként megadjuk a kulcs és az adat értékét:

```
tömbnév.add kulcs, adat
```

A tömbben az elemek a bevitel sorrendjében helyezkednek el.
Megadott kulcs létezését az *Exists* metódussal tudjuk ellenőrizni, melynek visszatérési értéke *true* vagy *false*:

```
tömbnév.exists(kulcs)
```

Az asszociatív tömb elemszámát a *Count* tulajdonság adja meg:

```
tömbnév.count
```

Mivel a tömb indexelése 0-val kezdődik, a maximális index eggyel kisebb, mint a *Count* értéke.
A *Remove* metódus törli a kulcsnak megfelelő elemet a tömbből:

```
tömbnév.remove(kulcs)
```

A *RemoveAll* metódus a tömb összes elemét törli:

```
tömbnév.removeall()
```

A tömb elemeire az *Item* függvénnyel hivatkozhatunk, melynek paramétere a keresett adat kulcsa:

```
tömbnév.item(kulcs)
```

A függvény értéke a megadott kulcshoz tartozó adat.
Az *Item* metódussal módosítani tudjuk a kulcshoz tartozó adatot:

```
tömbnév.item(kulcs)=kifejezés
```

Ha a *kulcs* nem található meg a tömbben, akkor az értékadó utasítás bal oldalán lévő kulcs a kifejezés értékével bekerül a tömbelemek közé! Ez a bővítés egyenértékű az *Add* metódussal.
Ha az *Item* egy kifejezésben szerepel, de kulcs paramétere még nincs benne a szótárban, akkor létrejön egy új elem az adott kulccsal és üres (*Empty*) értékkel.
Az *Item* kiírása elhagyható, így a tömb elemeit a kulcsokkal indexelhetjük:

```
tömbnév(kulcs)
```

A

```
tömbnév(kulcs) = érték
```

utasítás megváltoztatja a kulcshoz tartozó értéket, ha már létezik, és felveszi a kulcs–érték párt, ha még nem létezik.
A tömbelemek eléréséhez kényelmesen használható a *For Each* ciklus, amely sorra átadja a tömb kulcsait a ciklusváltozónak:

```
for each elem in tömbnév
  utasítások az elem változóval
next
```

A kulcs értékét a *Key* metódussal módosíthatjuk:

```
tömbnév.key(a kulcs régi értéke) = a kulcs új értéke
```

Halmazok

A VBScript közvetlenül nem rendelkezik a halmaz adattípussal. A halmaz azonban helyettesíthető olyan asszociatív tömbbel, amelynél a kulcsok jelentik a halmaz elemeit, a hozzájuk tartozó értékeket pedig üresen hagyjuk.
Példaként bemutatjuk a *h1* és *h2* halmaz metszetének kialakítását asszociatív tömbök segítségével:

```
for each kulcs in h1
  if h2.exists(kulcs) then
    temp=metszet.item(kulcs)
  end if
next
```

A példában kihasználtuk, hogy ha az *Item* metódus hívása egy kifejezésben szerepel, akkor létrejön a tömb eleme az adott kulccsal.

Rekordok

A rekordokat objektumokkal valósíthatjuk meg. Az objektumok teljes mértékben helyettesítik ezt az adatszerkezetet (sőt, sokkal több célra alkalmazhatók). Ha csak a rekord-adatszerkezetet akarjuk létrehozni, akkor leegyszerűsíthetjük az objektumok kezelését.
A rekordtípus definiálása osztálydefinícióval történik, amelyben felsoroljuk a rekord mezőit:

```
class rekordtípus
  dim mezőnév1, mezőnév2, ...
end class
```

A változókhoz hasonlóan a mezők típusát sem lehet megadni.
Egy rekord létrehozása a *New* operátorral történik (dinamikus memóriahasználat). A rekordot a *Set* utasítással rendeljük hozzá egy változóhoz:

```
dim rekordnév
...
set rekordnév = new rekordtípus
```

A *rekordnév* valójában egy mutató, amely a rekord címét tartalmazza a memóriában.
A mezők értékadásánál a rekord (objektum) nevét használjuk minősítőként:

```
rekordnév.mezőnév = kifejezés
```

A mezőhivatkozást a *With* utasítással rövidíthetjük:

```
with rekordnév
  .mezőnév1 = ...
  ...
end with
```

A rekordok mezői (az objektumok változói) tömböket (és objektumokat) is tartalmazhatnak.
A rekord automatikusan eltűnik a memóriából, mihelyt megszüntetjük a rá mutató hivatkozást (például egy másik rekordra állítjuk át). Explicit módon a rekordot úgy szüntetjük meg, hogy *Nothing*-et rendelünk a mutatóhoz:

```
set rekordnév = nothing
```

A rekordok használatát részletesen a [3] irodalom mutatja be.

Megjegyzés
Bár a mutatók használata nem igényli a Pascalban megszokott megkülönböztetést, mégis óvatosan kell bánnunk vele. Egy rekordokat tartalmazó tömb elemeinek cseréjénél, léptetésénél könnyen elveszíthetjük a rekordra mutató hivatkozást, ha nem a megfelelő sorrendben végezzük a hivatkozások módosítását. Részletesebben lásd a [3] irodalomban.

Hibakezelés

Kivételkezelés

A VBScript erősen korlátozott kivételkezelési lehetőséggel rendelkezik. Az *On Error Resume Next* utasítással kikapcsolhatjuk az interpreter hibakezelését. Az interpreter futási hiba jelzése nélkül abbahagyja a hibát okozó utasítás végrehajtását, és rátér a következő utasítás értelmezésére.
Az *On Error Resume Next* utasítást általában azon utasítás elé helyezzük, amely hibát okozhat. A hiba bekövetkezését az *Err.Number* változó nullától különböző értéke jelzi. A végrehajtás után az *On Error Goto 0* utasítással célszerű visszakapcsolni az interpreter hibakezelését. Például:

```
on error resume next
tort=szamlalo/nevezo
if err.number <>0 then
  msgbox "A nevező nulla!"
end if
on error goto 0
```

Az *On Error Resume Next* hatása akkor is megszűnik, ha kilépünk egy eljárásból vagy függvényből, illetve meghívunk egy újabb eljárást/függvényt.
Az *On Error Resume Next* utasítást leggyakrabban fájlkezeléskor vagy adatbeolvasáskor használjuk.
Megjegyzés
Az *On Error Goto 0* utasítás nem ugrik sehova, csak visszakapcsolja az interpreter hibakezelését.
Az *Err.Number* valójában az *Err*-objektum *Number* tulajdonsága. Az objektum további tulajdonságai:

Description: a hiba leírását tartalmazó sztring.
Source: a hibát okozó objektum. A szkriptek futtatásánál ez a VBScript interpretere.

Hibakezelés eseménykezeléssel

Egy hiba bekövetkezésekor létrejön az *onerror* esemény, melyhez eseménykezelő eljárást készíthetünk. Így a hibát az eljárásban kezelhetjük. Az *onerror* használatát a [3] irodalom ismerteti részletesen.

Fájlkezelés

A fájlrendszer kezelése

A VBScript a háttértárakat, mappákat, fájlokat objektumként kezeli[10]. Tulajdonságaikra a következőképpen hivatkozhatunk:

```
objektumnév.tulajdonságnév
```

A fájlrendszerobjektum

Az objektumok eléréséhez először létre kell hozni egy fájlrendszerobjektumot. Az objektumot a *Set* utasítással rendeljük hozzá egy változóhoz[11]:

```
set változónév = createobject("scripting.filesystemobject")
```

`Drive`	meghajtó, beleértve a különböző háttértárakat és a hálózati meghajtókat
`Drives`	a számítógéphez csatlakozó meghajtók kollekciója
`File`	fájl
`Files`	a mappa által tárolt fájlok kollekciója
`Folder`	mappa
`Folders`	a mappa almappáinak kollekciója
`TextStream`	szöveges fájl

A fájlrendszer objektumai

[10] Ehhez a *Script Runtime* osztálykönyvtár *File System Object* (fájlrendszerobjektum) modelljét használja.
[11] A változó az objektumra mutató hivatkozást tartalmaz.

A fájlrendszerobjektum szokásos elnevezése: FSO. A továbbiakban mi is így hivatkozunk rá.
A már nem használt objektumot célszerű a *Nothing* segítségével törölni:

```
set objektumnév = nothing
```

A fájlrendszer objektumain a metódusok segítségével hajtunk végre műveleteket. Az objektum metódusa a klasszikus nyelvek eljárásának, illetve függvényének felel meg, csak hívásakor eléje írjuk az objektum nevét:

```
objektumnév.metódusnév(paraméterek)
```

Az egyes meghajtókat a fájlrendszerobjektum *GetDrive* metódusával rendeljük hozzá az objektumra mutató változóhoz:

```
set változónév = fso.getdrive("név")
```

A név lehet egy meghajtó betűjele (kettősponttal vagy anélkül), illetve egy hálózati mappa elérési útja (*\\számítógépnév\megosztásnév*).

`DriveLetter`	a meghajtó betűjele kettőspont nélkül
`Drives`	a számítógéphez csatlakozó meghajtók kollekciója
`DriveType`	a meghajtó típusa
`FreeSpace`	a szabad terület mérete bájtban
`IsReady`	a háttértár készenléti állapotát jelző logikai érték
`Path`	a meghajtó betűjele kettősponttal (elérési út)
`RootFolder`	a gyökérkönyvtár elérési útja
`TotalSize`	a háttértár teljes kapacitása bájban
`VolumeName`	kötetcímke

A meghajtóobjektumok tulajdonságai és kollekciói

A kötetcímke írható/olvasható, a többi tulajdonság csak olvasható. A tulajdonságokat csak akkor érjük el, ha az objektum *IsReady* tulajdonsága *True* értékű. Egyébként hibaüzenetet kapunk.
A c: meghajtó tulajdonságainak lekérdezése például:

```
set fso = createobject("scripting.filesystemobject")
set merevlemez = fso.getdrive("c:")
if not merevlemez.isready then
  document.write "Állapot: nem olvasható."
else
  document.write "Kötetcímke: " & merevlemez.volumename & "<br>"
  document.write "Méret: " & merevlemez.totalsize & " bájt<br>"
  document.write "Szabad hely: " & merevlemez.freespace & " bájt<br>"
  document.write "Gyökérkönyvtár: " & merevlemez.rootfolder
end if
set merevlemez = nothing
set fso = nothing
```

A mappaobjektumok

A mappaobjektumok eléréséhez a mappát az FSO-objektum *GetFolder* metódusával hozzá kell rendelni egy változóhoz:

```
set változónév = fso.getfolder("elérési út")
```

`Files`	a mappa összes állományát tartalmazó kollekció
`IsRootFolder`	logikai érték, a gyökérkönyvtár esetén *True*, egyébként *False*
`Name`	a mappa neve (írható)
`ParentFolder`	a szülő mappaobjektum (!)
`Path`	a mappa teljes elérési útja
`Size`	a mappa mérete az almappákkal együtt
`SubFolders`	az almappákat tartalmazó kollekció

A mappaobjektumok és kollekcióik

A *Files* és *SubFolders* kollekciók elemszámát a *Count* tulajdonság értéke adja meg.

`CopyFile`	fájl másolása
`CopyFolder`	mappa másolása
`CreateFolder`	mappa létrehozása
`DeleteFile`	fájl törlése
`DeleteFolder`	mappa törlése
`FolderExists`	*True*, ha létezik a paraméterként megadott mappa
`GetFile`	fájl hozzárendelése egy változóhoz
`GetFolder`	mappa hozzárendelése egy változóhoz
`MoveFile`	fájl mozgatása
`MoveFolder`	mappa mozgatása

Az FSO-objektum mappákra és fájlokra vonatkozó metódusai

A *c:\windows* mappa tulajdonságainak kiírása például:

```
set fso = createobject("scripting.filesystemobject")
set mappa = fso.getfolder("c:\windows")
with mappa
  document.write "Név: " & .name & "<br>"
  document.write "Elérési út: " & .path & "<br>"
  document.write "Szülő mappa: " & .parentfolder & "<br>"
  document.write "Teljes méret (almappákkal): " & .size & "<br>"
  document.write "Almappák száma: " & .subfolders.count & "<br>"
  document.write "Fájlok száma: " & .files.count
end with
set fso = nothing
set mappa = nothing
```

A következő utasítások létrehozzák a *c:\Gyakorlás* mappát, és hozzárendelik a *Mappa* nevű változóhoz:

```
if fso.folderexists("c:\gyakorlás") then
  document.write "Már létezik ilyen mappa."
  set mappa = fso.getfolder("c:\gyakorlás")
else
  set mappa = fso.createfolder("c:\gyakorlás")
  document.write "Elkészült a mappa."
end if
```

A fájlobjektumok

A fájlobjektumok eléréséhez az állományt az FSO *GetFile* metódusával hozzá kell rendelni egy változóhoz:

```
set változónév = fso.getfile("elérési út")
```

`DateCreated`	a létrehozás ideje
`Name`	a fájl neve (írható)
`ParentFolder`	a fájl mappája
`Path`	a fájl teljes elérési útja
`Size`	a fájl mérete bájtokban
`Type`	a fájlt típusának megnevezése (megfelel az Intéző *Típus* oszlopában feltüntetett sztringnek)

A fájlobjektumok legfontosabb tulajdonságai

A *Windows* mappában lévő fájlok listázása.

```
set fso = createobject("scripting.filesystemobject")
set mappa = fso.getfolder("c:\windows")
document.write "A fájlok száma: " & mappa.files.count & "<br>"
for each fajl in mappa.files
  document.write fajl.name & " (" & fajl.size & " bájt, " & _
                 fajl.datecreated & ", " & fajl.type & ")<br>"
next
set mappa = nothing : set fso = nothing
```

Az aktuális mappa kijelölésével elkerülhetjük a teljes elérési út használatát. Az aktuális mappát a *Windows Scripting Host* szolgáltatás *Shell* objektumának *CurrentDirectory* metódusával kaphatjuk meg:

```
dim kornyezet, aktualismappa
set kornyezet = createobject("wscript.shell")
aktualismappa = kornyezet.currentdirectory
```

Az aktuális mappa értékét meg is változtathatjuk.

Szövegfájlok kezelése

A VBScript közvetlenül csak szövegfájlokat (szekvenciális fájlokat) tud létrehozni, írni, olvasni. Bináris fájlokat az *Asc* és a *Chr* függvények segítségével kezelhetünk.
A fájlt a használat előtt meg kell nyitni, a használat után pedig le kell zárni. A megnyitás történhet olvasásra, írásra, illetve hozzáfűzésre. Írás esetén a fájl előző tartalma elvész.
A fájl megnyitását az FSO-objektum *OpenTextFile*, lezárását pedig a *Close()* metódusával végezzük:

```
set változónév = fso.opentextfile(név, mód, létrehoz, kód)
```

Az egyes paraméterek jelentése:

név: a fájl elérési útja és neve;
mód: megnyitás olvasásra (1), írásra (2) vagy hozzáfűzésre (8);
létrehoz: *True*, ha nem létező név esetén hozza is létre a fájlt;
kód: Unicode esetén *True*, ANSI kód esetén *False*.

`AtEndOfLine`	*True*, ha elértük egy sor végét
`AtEndOfStream`	*True*, ha elértük a fájl végét
`ReadLine()`	beolvas egy sort
`ReadAll()`	beolvassa a fájl teljes tartalmát
`Read(n)`	beolvas *n* darab karaktert
`Skip(n)`	átlép (kihagy) *n* darab karaktert
`SkipLine()`	kihagy egy egész sort
`Write(sztring)`	kiírja a fájlba a sztring karaktereit
`WriteLine(sztring)`	kiírja a fájlba a sztringet és egy sorvége jelet

A szövegfájlobjektum legfontosabb tulajdonságai és metódusai

A következő programrészlet a *Gyakorlás* mappában létrehoz egy szövegfájlt, és beleír egy sort:

```
fajl = "c:\gyakorlás\szöveg.txt"
set fso = createobject("scripting.filesystemobject")
if fso.fileexists(fajl) then
  document.write "A létező fájl megnyitása hozzáfűzésre."
  set fajl = fso.opentextfile(fajl, 8, false)
else
  document.write "A fájl létrehozása és írása"
  set fajl = fso.opentextfile(fajl, 2, true)
end if
fajl.writeline("Ez lesz a fájl tartalma.")
fajl.close()
set fajl = nothing
set fso = nothing
```

A következő programrészlet beolvassa és kiírja az előbbi szövegfájlt:

```
ut = "c:\gyakorlás\szöveg.txt"
set fso = createobject("scripting.filesystemobject")
if not fso.fileexists(ut) then
  document.write "Nem létezik a fájl."
else
  set fajl = fso.opentextfile(ut, 1)
  do while not fajl.atendofstream
    document.write fajl.readline() & "<br>"
  loop
  fajl.close()
  set fajl = nothing
end if
set fso = nothing
```

Változók beolvasása

Ha az adatfájl egy sorában több numerikus változó szerepel, akkor a beolvasásnál a *Split* függvénnyel tudjuk az egyes értékeket szétválasztani.[12] A következő programrészlet beolvas egy fájlt, melynek minden sora három numerikus értéket tartalmaz[13]. A számokat egyetlen szóköz választja el egymástól.

```
dim adat(100,2)
...
n = 0
ut = "c:\gyakorlás\adatok.txt"
set fso = createobject("scripting.filesystemobject")
set fajl = fso.opentextfile(ut, 1)
do while not fajl.atendofstream
  sor = fajl.readline()
  n = n + 1
  temp = split(sor, " ")
  for i = 0 to 2
    adat(n,i)=csng(temp(i))
  next
loop
fajl.close()
set fajl = nothing
set fso = nothing
```

A *Split* használatát elkerülhetjük, ha a kiírásánál minden egyes értéket új sorba írnunk.

Hibakezelés a fájlműveleteknél

A fájlokra vonatkozó műveletek során az *On Error Resume Next* utasítással célszerű kikapcsolni az interpreter hibakezelését. Ha az *Err*-objektum *Number* tulajdonsága nulla, akkor hibátlanul végrehajtásra került a fájlra vonatkozó utasítás. Így elkerülhetjük a futási hibát. Az interpreter hibakezelését az *On Error Goto 0* utasítással kapcsoljuk vissza[14].
A következő példa futási hiba nélkül próbál megnyitni olvasásra egy fájlt a flopin:

```
const ut = "a:\adatok.txt"
set fso = createobject("scripting.filesystemobject")
on error resume next
if not fso.getdrive("a:").isready then
  document.write "Nem használható a meghajtó."
else
  set fajl = fso.opentextfile(ut, 1)
  if err.number <> 0 then
    document.write "Nem nyitható meg a fájl."
    set fajl = nothing
  else
    document.write "Megnyitottam a fájlt."
  end if
end if
on error goto 0
további utasítások
```

[12] A *Split* visszatérési értéke dinamikus tömb.
[13] Az *adat*-tömb 0 indexű elemét nem használjuk fel.
[14] Az *On Error Goto 0* nem ugrik sehova, egyszerűen csak visszakapcsolja a hibakezelést.

Programozási tételek megvalósítása VBScriptben

A programozási tételeket eljárásokkal valósítottuk meg. A keretprogram az adatokat készíti elő, és az eredményeket írja ki. A megjelenítésnél törekedtünk az egyszerűségre, nem alkalmaztuk a grafikus felhasználói felület objektumait.
A programozási tételek sorozatainak elemeit a *tomb*, *tomb1*, *tomb2* … tömbök tartalmazzák. A tömbök 0 indexű elemét nem használtuk fel.
Általában elkerültük az *ubound* függvény alkalmazását, a tömbök elemszámát paraméterként adtuk át az eljárásoknak. Tömböt tartalmazó változót, illetve dinamikus tömböt csak korlátozottan használtunk.
A programozási tételek algoritmusait [6] és [7] alapján kódoltuk. Ettől csak szükség esetén, például a logikai kifejezések rövidre zárásának hiánya miatt tértünk el.

A programokra mutató alábbi hivatkozások csak akkor működnek, ha jelen pdf dokumentum könyvtárában helyezkedik el a példafájlokat tartalmazó *prtételek* mappa.

A *PrTételek* mappában a keretprogramokat, *Sub* almappájában pedig az eljárásokat találjuk. Az eljárások kódját csatoltuk a keretprogramhoz (lásd: kód csatolása). Gyakran függvény helyett is eljárást készítettünk (például eldöntés).
Az eljárásokban nem használtuk fel a 0 indexű tömbelemet, tehát a

```
dim a(n)
```

esetén az *n* a tömbelemek számát jelenti.
A Visual Basic Scriptben az eljáráshívásnál a cím szerinti paraméterátadás az alapértelmezett mód. Ettől függetlenül az eljárások kimeneti változóinál kiírtuk a *ByRef* kulcsszót, hogy kiemeljük a paraméter megváltozását.

Az indexhatár túllépése

A programozási tételek szokásos megfogalmazása felhasználja a logikai műveletek rövidrezárását. A rövidzár nagymértékben leegyszerűsíti az algoritmust, bár didaktikai szempontból nem célszerű az alkalmazása. A VBScript nem ismeri a rövidzárat. A rövidzár hiányát és annak feloldását az eldöntés algoritmusán szemléltetjük.
A *tomb* nevű változó 1-től 10-ig tartalmazza a természetes számokat (a 0 indexű elemet nem használjuk fel):

```
tomb=array(, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Vizsgáljuk meg, hogy az elemek között van-e 11-gyel osztható szám! Az algoritmus szokásos kódolása:

```
n=10
i=1
do while i<=n and tomb(i) mod 11 <> 0
  i=i+1
loop
van=(i<=n)
```

Ez a megoldás VBScriptben futási hibához vezet, mert a ciklus utolsó ismétlésénél az *i* értéke már 11 így a *tomb(i)* hivatkozás kívül esik az indexhatáron.

1. módszer: A feltételek kettéválasztása

A futási hibát a feltételek kettéválasztásával kerülhetjük el, amihez bevezetünk egy logikai változót:

```
n=10
i=1
oszthato = false
do while i<=n and not oszthato
  oszthato = (tomb(i) mod 11 = 0)
  i=i+1
loop
van=oszthato ' Ha van, akkor az i-1 indexű elem rendelkezik az adott tulajdonsággal!
```

2. módszer: A ciklus korábbi befejezése

A futási hibát a ciklus korábbi befejezésével is elkerülhetjük. Ha a ciklust *i*<*n*-ig futtatjuk, akkor nem kerül sor az indexhatár túllépésére. Ekkor a kilépés után ismét el kell végezni a vizsgálatot:

```
n=10
i=1
do while i<n and tomb(i) mod 11 <> 0
  i=i+1
loop
van=(tomb(i) mod 11 = 0)
```

A megoldás hátránya, hogy két helyre kell beilleszteni a vizsgálatot, ami nem mindig célszerű. Rontja a hatékonyságot és a program áttekinthetőségét.

3. módszer: A tömb bővítése fiktív elemmel

Az eredeti algoritmust változatlan formában alkalmazhatjuk, ha a tömböt bővíteni tudjuk egy *n*+1-edik elemmel:

```
tomb=array(, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0)
...
n=10
i=1
do while i<=n and tomb(i) mod 11 <> 0
  i=i+1
loop
van=(i<=n)
```

Hangsúlyozzuk, hogy az utolsó elem értéke tetszőleges lehet, hiszen $i = 11$ esetén az „és” művelet bal oldala már hamis eredményre vezet, tehát nem fut le a ciklus.

1. Elemi programozási tételek

1.1. Sorozatszámítás
1.2. Eldöntés
1.3. Kiválasztás
1.4. Lineáris keresés
1.5. Megszámlálás
1.6. Maximumkiválasztás

2. Összetett programozási tételek

2.1. Másolás
2.2. Kiválogatás
2.3. Szétválogatás
2.4. Szétválogatás helyben
2.5. Metszet
2.6. Unió
2.7. Halmazfelsorolás készítése
2.8. Összefuttatás
2.9. Összefuttatás ütközővel (bármely numerikus érték kisebb, mint egy sztring, lásd: Relációs operátorok)

3. Rendezések

A rendezéseket bemutató programok a tömböt véletlenszámokkal töltik fel. A program végrehajtása az F5 billentyűvel (frissítés) ismételhető meg.

3.1. Egyszerű cserés rendezés
3.2. Minimumkiválasztásos rendezés
3.3. Buborékos rendezés
3.4. Beillesztéses rendezés
3.5. Szétosztó rendezés
3.6. Számlálva szétosztó rendezés
3.7. Számláló rendezés
3.8. Rendezés Shell-módszerrel
3.9. Gyorsrendezés (quicksort)

4. Keresések

4.1. Keresés rendezett sorozatban
4.2. Logaritmikus keresés
4.3. Visszalépéses keresés
  4.3. a) 8 vezér a sakktáblán (grafikus megjelenítéssel)
  4.3. b) A 8 vezér feladat összes megoldása
  4.3. b) Növekvő sorozat kiválasztása

A visszalépéses keresést célszerű egy olyan példán bemutatni, amelynél könnyű ellenőrizni a megoldhatóságot. A feladatban 6 számsorozat közül kell kiválasztani egymás után egy-egy elemet úgy, hogy a kiválogatott elemek növekvő sorozatot alkossanak.

A feladat látványosabb megoldását lásd: [3].

Beépített függvények

Matematikai függvények

Argumentumuk általában egy numerikus kifejezés.

Abs(*kifejezés*) abszolútérték.
Atn(*kifejezés*) arkusz tangens, radiánban.
Cos(*kifejezés*) a radiánban megadott szög koszinusza.
Exp(*kifejezés*) *e* alapú hatvány. Tetszőleges alapú hatványozáshoz használjuk a ^ operátort.
Hex(*kifejezés*) a kifejezés legfeljebb 8 jegyű hexadecimális értéke (sztring).
Int(kifejezés) a kifejezés egészrésze. Például: Int(-99,8) = -100.
Fix(kifejezés) elhagyja a kifejezés törtrészét. Például: Fix(-99,8) = -99.
Log(kifejezés) a kifejezés *e* alapú logaritmusa (ln *x*). $\log_a x = \log(x)/\log(a)$.
Rnd() véletlenszám a [0; 1) intervallumban.
Round(*kifejezés*[, *tizedes*])
a numerikus értéket a megadott számú tizedesre kerekíti. Ha a *tizedes* paraméter hiányzik, akkor egész számot kapunk. Az 5-öt a legközelebbi páros számra kerekíti.
Például: round(2,5) = 2, de round(3,5) = 4.
Sgn(*kifejezés*) a kifejezés előjele. –1: negatív; 0: nulla; +1: pozitív.
Sin(*kifejezés*) a radiánban megadott szög szinusza.
Sqr(*kifejezés*) a kifejezés négyzetgyöke.
Tan(*kifejezés*) a radiánban megadott szög tangense.

Sztringfüggvények

A sztringeket összehasonlító/kereső függvények *mód* paramétere megadja az összehasonlítás módját.
mód = 0: bináris összehasonlítás. Megkülönbözteti a kis- és nagybetűket.
mód = 1: szövegszerű összehasonlítás. Nem különbözteti meg a kis- és nagybetűket.
A paraméter alapértelmezett értéke: 0, ezt nem kötelező kiírni.

Asc(*sztringkifejezés*)
Az első betű ANSI kódja. Unicode esetén az *Asc* helyett használjuk az *AscW* függvényt!
Chr(*kifejezés*)
A kifejezésnek megfelelő ANSI kódú karakter. Unicode esetén használjuk a *ChrW* függvényt!
InStr(([*start*,]*sztring1*, *sztring2*[, *mód*])
Megkeresi a *sztring2* első előfordulásának pozícióját a *sztring1*-ben. Ha szerepel, akkor a *start*-nak megfelelő pozíciótól kezdi a keresést. A visszatérési érték 0, ha
- üres a *sztring1*,
- nem találta meg a *sztring2*-t,
- *start* > len(*sztring2*)

InStrRev(([*start*,]*sztring1*, *sztring2*[, *mód*])
Ugyanaz, mint az *InStr*, csak a *sztring1* végén kezdi a keresést.
Join(*tömb*[, *határoló*])
A sztringeket tartalmazó tömb elemeiből készít egyetlen sztringet. Az elemek közé beilleszti a határoló sztringet. Ha nem szerepel a határoló paraméter, akkor az elemek közé szóköz kerül. Ha a határoló paraméter üres sztring, akkor határoló sztring nélkül fűzi össze a tömb elemeit.
LCase(*sztringkifejezés*)
A sztring nagybetűit kisbetűkre váltja.
Left(*sztring*, *hossz*)
A sztring első hossz karakterét adja vissza.
Len(*sztringkifejezés* | *változónév*)
A sztring karaktereinek a számát adja vissza.
LTrim(*sztringkifejezés*)
Elhagyja a sztring elején álló szóközöket.
Mid(*sztring*, *start*[, *hossz*])
A *start* pozíciótól kezdve megadja a sztring hossz karakterből álló részét. Ha hiányzik a *hossz* paraméter, akkor a sztring végéig veszi a karaktereket. Ha a *start* nagyobb, mint a sztring hossza, akkor üres sztringgel tér vissza.
Replace(*sztringkifejezés*, *keres*, *helyettesít*[, *start*, *darab*[, *mód*]])
A sztringkifejezésben a *keres* sztringet helyettesíti a *helyettesít* sztringgel. Ha szerepel a *start* paraméter, akkor a keresést a *start* pozíciótól kezdi. Ekkor meg kell adnunk a helyettesítések *darab* számát is! Ebben az esetben az összes rész-sztring helyettesítését a *darab* = -1 jelzi.

Right(*sztring*, *hossz*)
A sztring utolsó, *hossz* darab karakterét adja vissza.
RTrim(*sztringkifejezés*)
Elhagyja a sztring végén lévő szóközöket.
Space(*szám*)
Megadott számú szóközt tartalmazó sztring.
Split(*sztring*[, *határoló*[, *darab*[, *mód*]]])
A sztringet a *határoló* sztringeknél feldarabolja rész-sztringekre, melyeket egy tömb elemeiként ad vissza. Ha a *határoló* paraméter hiányzik, akkor a szóközöknél vágja el a sztringet. A *darab* paraméterrel korlátozhatjuk a visszaadott sztringek számát (a tömb méretét).
StrComp(*sztring1*, *sztring2*[, *mód*])
A *mód*=1 értéke mellett a magyar ábécének megfelelő összehasonlítást végez (de nem különbözteti meg a kisbetűket és a nagybetűket). A visszatérési érték:

-1	sztring1 < sztring2
0	sztring1 = sztring2
+1	sztring1 > sztring2

String(*szám*, *karakter*)
Szám-szor megismétli az adott karaktert.
StrReverse(*sztring*)
Megfordítja a sztringet.
Trim(*sztringkifejezés*)
Elhagyja a sztring elején és végén lévő szóközöket.
UCase(*sztringkifejezés*)
A sztring kisbetűit nagybetűkké konvertálja.

Dátum/idő függvények

Date() megadja a rendszerdátumot.
DateAdd(*intervallum*, *érték*, *dátum*)
A *dátum* sztringhez hozzáadja az *intervallum* sztringnek megfelelő *érték*-et.
(Az érték lehet negatív is.)
Intervallum: "yyyy": év, "m": hónap, "d": nap, "w": a hét napja,
"h": óra,"n": perc, "s": másodperc stb.
Például: DateAdd("m", 1, "96/01/31") = 1996. február 29. (#1996/02/29#)
DateSerial(*év*, *hó*, *nap*)
A numerikus kifejezésekből (vagy konstansokból) dátum típusú értéket képez.
Például: DateSerial(1990-10, 8-2, 1-1)=1980. május 31. (#1980/05/31#)
DateValue(*sztringkifejezés*)
a dátumformátumnak megfelelő sztringkifejezésből dátum típusú értéket képez.
Day(*dátum*) a dátum napja. Például: Day(#1999/11/05#)=5
Hour(*idő*) az időpont órája.
Minute(*idő*) az időpont perce.
Month(*dátum*) a dátum hónapja.
MonthName(*sorszám*[, *rövidít*])
az adott sorszámú hónap neve. Ha a *rövidít* igaz, akkor rövidíti a hónapnevet.
A hónapneveket a területi beállításoknak megfelelő nyelven (tehát például magyarul) írja ki.
Now() megadja a rendszerdátumot és időt.
Second(*idő*) az időpont másodperce.
Time() megadja a rendszeridőt.
Timer() az éjfél óta eltelt idő másodpercben, századmásodperc pontossággal. Futási idő mérésére használhatjuk.
TimeSerial(*óra*, *perc*, *másodperc*)
a numerikus paraméterekből idő típusú értéket képez.
TimeValue(*sztringkifejezés*)
az időformátumnak megfelelő sztringkifejezésből idő típusú értéket képez.
Weekday(*dátum*[, *kezdőnap*])
a hét napjának sorszáma. *Kezdőnap* 1: vasárnap, 2: hétfő, …, 7: szombat.
WeekdayName(*sorszám*, *rövidítés*, *kezdőnap*)
a hét adott sorszámú napjának a megnevezése a területi beállítások szerinti nyelven (például magyarul). Ha a *rövidítés* igaz, akkor rövidíti a nevet. A *kezdőnap* megfelel a *Weekday* függvény paraméterének.
Year(*dátum*) a dátum éve.

Típuskonverziós függvények

CBool(*kifejezés*)[15]	logikai (*False*, ha a kifejezés értéke 0, egyébként pedig *True*)
CByte(*kifejezés*)	bájt
CCur(*kifejezés*)	pénznem
CDate(*kifejezés*)	dátum
CDbl(*kifejezés*)	dupla pontosságú valós
CInt(*kifejezés*)	egész
CLng(*kifejezés*)	hosszú egész
CSng(*kifejezés*)	egyszeres pontosságú valós
CStr(*kifejezés*)	sztring

Formátumfüggvények

FormatCurrency(*kifejezés*, *paraméterek*)

kiírja a *Területi beállításoknak* megfelelő pénznem megjelölését is. Paraméterei megegyeznek a *FormatNumber* paramétereivel.

FormatNumber(*kifejezés*, *tizedesjegyek száma*, *vezető nulla*, *negatív zárójel*, *ezresek elválasztása*)

kifejezés:	a formázásra kerülő aritmetikai kifejezés;
tizedesjegyek száma:	a kiírásra kerülő tizedesjegyek száma. Szükség esetén nullákkal egészíti ki. –1 esetén a *Területi beállításoknál* megadott értéket alkalmazza.
vezető nulla:	kiírja-e a nulla egészet az 1-nél kisebb számoknál;
negatív zárójel:	zárójelbe tegye-e a negatív számokat;
ezresek elválasztása:	csoportosítsa-e a *Területi beállításoknál* megadott módon az egészrész számjegyeit.

A három utolsó paraméter értéke:[16]

Érték	*Konstans*	*Jelentés*
–1	TristateTrue	igen (igaz),
0	TristateFalse	nem (hamis),
–2	TristateUseDefault	a *Területi beállításokat* veszi figyelembe.

Tesztfüggvények

A tesztfüggvények visszatérési értéke logikai igaz (*True*) vagy hamis (*False*).

IsArray(*változónév*)	igaz, ha a változó tömb. Azt vizsgáljuk meg vele, hogy egy változó tömb vagy tömböt tartalmaz-e.
IsDate(*kifejezés*)	igaz, ha a kifejezés dátummá/idővé konvertálható.
IsEmpty(*változónév*)	igaz, ha a változó még nem kapott értéket. Az automatikus típuskonverzió miatt a `szoveg = ""` akkor is igaz, ha a *szoveg* nevű változó még nem kapott értéket.
IsNumeric(*kifejezés*)	igaz, ha a kifejezés számmá konvertálható. Számok beolvasásának ellenőrzésére használjuk.

Egyéb függvények

Eval(*sztringkifejezés*)	Kiértékeli a sztringként megadott kifejezést.
RGB(*vörös*, *zöld*, *kék*)	Megadja a színt reprezentáló értéket. A paraméterek 0–255 közé eshetnek.
UBound(*tömbnév*[, *dimenzió*])	A tömb adott dimenziójának legnagyobb indexértéke.

Speciális karakterkódok

Néhány speciális karakter HTML-kódja:[17]

&alpha;	α		szóköz
β stb.	β stb.		(többször egymás után is szerepelhet)
&deg;	º (fok)	&plusmn;	±
&gt;	>	&radic;	√
&lt;	<	&times;	×

 sortörés-elem

[15] Convert to Boolean stb.
[16] Háromállapotú (tristate) konstansok.
[17] Karakter-entitások, beírhatók a HTML-kódba.

Irodalomjegyzék

[1] Hillier, Scot: Inside Microsoft Visual Basic, Scripting Edition (Microsoft Press, 1996)
[2] Holzner, Steven: Web Scripting with VBScript (M&T Books, A Subsidiary of Henry Holt and Company, 1996)
[3] Juhász Tibor – Kiss Zsolt: Tanuljunk programozni. Visual Basic Script, objektumok, web (ComputerBooks, Budapest, 2004)
[4] Kuzmina Jekatyerina – Dr. Tamás Péter – Tóth Bertalan: Programozzunk Visual Basic rendszerben! (ComputerBooks, Budapest, 2003)
[5] Microsoft Visual InterDev 6.0 Web Technologies Reference (Microsoft Press, 1998)
[6] Szlávi Péter – Zsakó László: Módszeres programozás: Programozási tételek (Mikrológia 19, 6. kiadás, ELTE Informatikai Kar, 2004)
[7] Szlávi Péter – Zsakó László: Módszeres programozás: Rekurzió (Mikrológia 4, 5. kiadás, ELTE Informatikai Kar, 2004)
[8] Szlávi Péter – Zsakó László – Temesvári Tibor: Módszeres programozás: A programkészítés technológiája (Mikrológia 21, 6. kiadás, ELTE Informatikai Kar, 2004)
[9] VBScript Language Reference (Microsoft Corporation, http://msdn.microsoft.com/downloads, 2001)
[10] VBScript User's Guide (MSDN Library, http://msdn.microsoft.com/library, 2006)
[11] Comprehensive JScript and VBScript Reference (Microsoft Corporation, http://www.microsoft.com/downloads, 2007)

Tartalomjegyzék