

SZOFTVEREK ÉRTÉKELÉSE ISKOLAI SZEMPONTOK SZERINT

Szlávi Péter

szlavip@elte.hu

1999-2009

TARTALOM

Általános megjegyzések.....	5
<i>Milyen szoftverekről lehet szó?</i>	<i>5</i>
<i>Az adott szoftver oktatásának céljai</i>	<i>5</i>
<i>Oktatás elvárásai a szoftverekkel szemben.....</i>	<i>5</i>
I. Programozási nyelvek.....	6
1. <i>A programozási nyelvek tanításának céljai</i>	<i>6</i>
1.1. Algoritmikus gondolkodás tanítása	6
1.2. Feladatmegoldás adott témakörben -- egy modell megértésének eszköze	6
1.3. Egy nyelvtípus megismerése	7
1.4. „Profi” programozó képzés	7
2. <i>Értékelési szempontok</i>	<i>8</i>
2.1. Nyelvi egyszerűség	8
2.2. Tipikusság	10
2.3. Használhatóság.....	11
2.4. Fejlesztői környezet léte.....	12
2.5. Szabványosság	13
2.6. Fejlesztői környezet finomabb részletei	14
2.7. Biztonságosság.....	15
2.8. Bonyolultság – egészében	16
3. <i>Az egyes nyelvek értékelése</i>	<i>17</i>
3.1. Free Pascal	18
3.2. PHP	19
3.3. Visual BASIC	20
3.4. Java Script.....	21
3.5. Turbo Prolog	22
3.6. Code::Blocks – C++.....	23
3.7. C#.....	24
3.8. Visual C++	25
3.9. Scratch.....	26
3.10. Comenius Logo	27
3.11. Delphi.....	28
3.12. Python	29
3.13. Java	30
3.14. Perl	31
II. Alkalmazói rendszerek.....	32
1. <i>Az Alkalmazói rendszerek tanításának céljai</i>	<i>32</i>
1.1. Amatőr (hétköznapi) alkalmazó	32
1.2. Rendszertípus megismerése	32
1.3. „Profi” alkalmazó.....	32
2. <i>Értékelési szempontok</i>	<i>32</i>
2.1. Egyszerűség	32
2.2. Vizualitás	34
2.3. Teljesség	34
2.4. Rugalmasság, „testre szabhatóság”	34
2.5. Megbízhatóság, biztonságosság	35
2.6. Kompatibilitás	36
3. <i>Az egyes alkalmazói rendszer-osztályok értékelése.....</i>	<i>37</i>
3.1. Táblázatkezelőkről (Quattro/Excel/Lotus 1-2-3/OpenOffice:Calc)	37
3.2. Szövegszerkesztőkről (Jegyzettömb/Norton Editor/WordPerfect/WinWord/T _E X/OpenOffice:Writer)	37
3.3. Adabázis-kezelőkről (DBase III/Clipper/Fox Pro/Access/Oracle/OpenOffice:Base)	37
3.4. Rajzoló programokról (Paint/CorelDraw!/PhotoShop/Gimp/OpenOffice:Draw)	38
<i>Táblázatkezelők</i>	<i>39</i>
<i>Szövegszerkesztők.....</i>	<i>40</i>
<i>Adatbázis-kezelők</i>	<i>41</i>
<i>Ábrászerkesztők.....</i>	<i>42</i>

ÁLTALÁNOS CÉLÚ SZOFTVEREK ÉRTÉKELÉSE

ISKOLAI SZEMPONTOK SZERINT

A dolog érdekessége számunkra az, hogy

1. abban a ritka pillanatban, amikor iskolai szoftverválasztáskor döntéshelyzetben kerülnénk, akkor nagy segítség lenne, ha lenne egy **szempontrendszer**, ami alapján összevethetnénk a választható szoftvereket; illetve
2. a **tematika** összeállításakor ezen szempontok irányíthatnák figyelmünket az adott szoftver „**gyenge pontjaira**”, erősségeire, amelyeket a normál hangsúlytól eltérően illesztenénk be az anyagba.

ÁLTALÁNOS MEGJEGYZÉSEK

Milyen szoftverekről lehet szó?

Az oktatást nyilván a (1) „**hétköznapi**”, ill. a (2) **szoftverfejlesztés** szoftverei érdeklik.

Az adott szoftver oktatásának céljai

- 1) Egy speciális alkalmazási, ill. fejlesztési **eszköz használata** (konkrét használat).
- 2) Az ilyen célú eszközök **általános bemutatása** (mire jó egy ilyen szoftver, mik az alapfogalmai, milyen sajátos filozófiával dolgozik: hogyan használható, nagy vonalakban).
- 3) Az adott szoftver **használatának mélyebb megtanítása, speciális céllal**.

Oktatás elvárásai a szoftverekkel szemben

Alkalmazásához **ne** legyen szükség

- 1) mély, **speciális ismeretekre** (vö. CAD-ismeret szükséges egy műszaki szki-ban, project manager – szervezési ismeretek egy közgazdasági szki-ban),
- 2) **speciális, drága környezetre** (gép, periféria, operációs rendszer, szoftver stb.; vö. winchester-/memóriaméret-igény .net, vagy Java futtatókörnyezet esetén; hálózati operációs rendszer [ki-szolgáló oldali] Perl¹ vagy MySQL esetén; a „drágaság” nemcsak az egyszeri beruházásra vonatkozik, hanem a fenntartásra –lízingre, folytonos rendszergazdai teendőkre– is)

Programozási nyelv esetén

- 1) a fejlesztői felület és a nyelv által „támogatott témaspektrum” **illeszkedjék** (vizualításban, apró-lékosságban) **korosztályhoz**;
- 2) előny a „**multifunkcionalitás**” – többféle környezetben (konzolos, GUI-s, web-es, mobil-világhoz), többféle programozási nyelvhez kapcsolódhasson.

¹ ... már, ami a nyelvválasztás értelmét illeti az OR „hálózati” jelzője elkerülhetetlen... bár nem kizárt az „egyedi” gépen történő használata sem.

Alkalmassint szolgálhasson **útmutató**ul más, „rokon” szoftverek felé. Pl.:

- 1) szövegszerkesztő $\Rightarrow$ kiadványszerkesztő / prezentáció- / honlapkészítés;
- 2) táblázatkezelő $\Rightarrow$ adatbázis-kezelő / statisztikai rendszerek (pl. SPSS);
- 3) strukturált programozás $\Rightarrow$ Logo (eljárás, elágazás, számlálás ciklus, rekurzió), Scratch („parketta” struktogramok [egymásba illeszthető]); típusmegvalósítás embrionális foka (Logo – típustalanság, adatstrukturálás: lista; C/Pascal – alapstruktúrák) $\Rightarrow$ típusmegvalósítás fejlettebb foka (Modula, OOP – Imagine, Scratch, C++, Python, Java, Ada, Eiffel).

I. PROGRAMOZÁSI NYELVEK

1. A programozási nyelvek tanításának céljai

1.1. Algoritmikus gondolkodás tanítása

Egy sikerélményhez vezető **eszköz** a gondolkodásfejlesztésben, annak a téveszmének elkerülésére, hogy „*mivel a programozás absztrakt tevékenység –s ilyen formán programozási nyelvtől független–, ezért a programozástanítás is nyelvfüggetlenül végezhető*”.

Az informatikaoktatás hajnalán jellemző volt a *programnyelv* oktatásának *túlhangsúlyozása*, egy „téveszme a négyzeten”, nevezetesen „informatika = programozás = programozási nyelv”.

E két szélsőség között kell a megfelelő utat megtalálni.

1.2. Feladatmegoldás adott témakörben – egy modell megértésének eszköze

Nem szorul magyarázatra –hitem szerint– az, hogy mekkora oktatási előnnyel jár, ha a tanuló egy rendszert (lehet az fizikai, kémiai, közgazdasági ...) maga is kipróbálhat, ha egy rendszerrel maga is kísérletezhet. A működés száraz leírásánál sokszorta többet jelent a „testközeli” próbálgatás, még akkor is, ha a valós rendszernek csak egy többé-kevésbé idealizált modelljét használhatja. Ilyen számítógépes eszközöket jelenthetnek a **szimulációs programok**.

A modell megismerése szempontjából a pusztán **programhasználat**nál is több a modell tervezése és megvalósítása, azaz a **szimulációs modellezés**. Ilyen modellek elkészítésénél természetesen nem állhatunk meg. A modellnek a kísérlet aktív résztvevőjévé, számítógépes eszközzé kell válnia. Kijelenthetjük, hogy a való világ rendszereinek megismeréséhez, működésük megértéséhez a **számítógépes szimuláció** „gondolkodásmódjának” megértetésén, és annak programozásán keresztül vezet az út. ²

² Ehhez persze „használatos módszertan” is kell, azaz a „túlméretes” program olyan feltördelése, amelyben csak egy-két darab megírása hárul a tanulóra. Nevezetesen egy keretprogramot készítünk, amelyben lévő „torzó” eljárásokat kell befejezni, vagy egy önálló modulba (Pascal: unit, Pascal/C++: include, C#: namespace) különítjük el a tanulóknak szánt eljárásokat még kifejtetlenül, amelyet csak hozzá kell fordítani a „nagy egészhez”.

Észre kell vennünk, hogy ez a hozzáállás, tehát a „feladatmegoldás algoritmikus alapon” rendelkezik egyfajta „**univerzalitással**”, azaz: a téma és az idealizáció fokától nagyban független. (Szemben például a hagyományosabb matematikai módszerekkel, amelyek jelentősen módosulnak, „durvulnak” az idealizáció csökkentésével. Kezdetben esetleg elegendő egy lineáris egyenlet, vagy egyenletrendszer, később ez magasabb fokúvá válik, amely megoldása már teljesen eltérő módszert igényel, végül akár differenciál egyenletrendszerek megoldására kényszerülünk. Lásd [SzimVsMat.pdf](#).)

1.3. Egy nyelvtípus megismerése

Egyrészt fontos cél a különféle **programozási nyelvosztályok modellezése**. Pl.:

- a funkcionális nyelvekhez $\Rightarrow$ Logo függvényes része (a „csiszolatlan computerium” korában a Forth);
- a logikai nyelvekhez $\Rightarrow$ Prolog;
- az automata-elvű nyelvekhez $\Rightarrow$ a Logo, a COMAL³, a Turbo Pascal (l. Graph3 unit), Scratch⁴ vagy a Python⁵ teknőcgrafikája;
- tárgyközpontúsághoz $\Rightarrow$ Imagine, Scratch, Python, Java, C#...

Másrészt be kell mutatni azt a **fejlesztői környezetet**, amelyen keresztül történik a nyelv megismerése. Pl.:

- Pascal „natív”, konzolos API-ja;
- Geany a kurrens fpc-re „hangolva” vagy Code::Blocks a C++ esetén;
- Visual Studio pl. a C# nyelvhez, Delphi

Harmadrészt hasznos lehet az **alkalmazói rendszerekhez** kapcsolódó nyelvek **modellezése**; mint például a Quattro/Excel vagy a WinWord makrónyelvéből kinőtt WordBASIC, majd Visual BASIC, de gondolhatunk a T_EX rejtélyesebb képességeit kiaknázó makrók nyelvezetére, esetleg a MAPLE-höz hasonló matematikai rendszerek programozási lehetőségeire ...

1.4. „Profi” programozó képzés

Bizonyos körülmények esetén szó lehet az átlag műveltséget meghaladó programozási ismeretek tanításáról is. Így például felvételi előkészítés, [OKJ](#)-vizsga címén.

³ A nyelvleírás: <http://en.wikipedia.org/wiki/COMAL>,

és egy jellegzetes példaprogram: <http://www.josvisser.nl/opencomal/whitepaper1.txt>

⁴ A Scratch teknőcgrafikájáról: <http://hirmagazin.sulinet.hu/hu/evilag/scratch-ii-teknocgrafika-es-vezeresi-szerkezetek>

⁵ A nyelvleírás: <http://www.python.hu/>.

2. Értékelési szempontok

Először azt a kérdést vizsgáljuk, hogy **maga a nyelv** milyen, s nem pedig valamely implementációja.

2.1. Nyelvi egyszerűség

Milyen könnyű megírni az első programot?

2.1.1. Értelmes alapszavak

Kulcsfeltétele a kezdőlépések megtehetőségének, hiszen a kezdők figyelmének elsődleges „célpontja”.

Az „ősi” BASIC-ben pl. 10½ **kulcsszó** van csupán: LET, IF, THEN, GOTO, FOR, NEXT, INPUT, PRINT, DIM, GOSUB, RETURN. Elborzasztó szintaktikai példák: Forth – „, !, @, ...; [APL](#) – [, #, ..., vagy [Perl](#) \$xxx, @xxx, &xxx, %xxx, @[, @_...⁶; C++ – cin>>, cout<<. (Lásd a mellékelt ábrákat!)

```
:EVEKTO
<BUILDS DUP ,
          1+ 1 DO 0 , LOOP DROP
DOES>
  DUP @
  ROT <
  IF ... HIBÁS INDEX ...
  ." HIBÁS INDEX:" DROP .
  ELSE SWAP 2* +
  ENDIF ;
```

Egy Forth programrészlet, amellyel megelőzte a korát (a vektor „objektuma”)

$$(\sim R_1 R_1 \cdot \times R) / R \leftarrow 1 \downarrow tR$$

Egy APL programrészlet, amely kiválogatja az összes prímet 1 és R között



Ezeknek némileg ellentmond (?) az a tapasztalat, hogy a C-t sokan éppen olyan vonásáért dicsérik, amely rövidíti a gépelést és amúgy világos is: { és }, ezért OK. Nem így a '++i' vagy 'i++'.

Elgondolkodtató az olvashatóság *túlhangsúlyozása* pl. a COBOL esetén. Aritmetikai műveleti „jelek”: ADD, SUBSTRUCT, MULTIPLY, DIVIDE...

Megállapítható, hogy olyan **szimbólumok** rövidítés célú alkalmazása lehet elfogadható, amelyek esetleg más (tudomány-) területen (pl. a matematikában) **már meghonosodtak**, közismertségnek örvendenek, és természetesen az ottanival azonos értelemben használható, így világos a kifejeznivalója, és rövid.

Az alapszavak **programba illeszkedéséről**: **kiemelendők-e**, pl. nagybetűkkel (ELAN, MODULA ...), vagy máshogyan (aposztróffal, mint anno az ALGOL 60-ban)? Általában (azaz nem csak az alapszavak esetében) a **kis- és nagybetűk** megkülönböztetésének kényszere is fontos (stílust meghatározó) *módszertani kérdés*. Illetőleg az **implicit** (értsd alapszó nélküli) döntések egy programban szintén zavarólag hathatnak. Pl. a Pascalban (C-ben) komoly veszélyforrás a hozzáférési jog nélkül szervezett paraméterátadás, ráadásul nehezen érthető a Const-tal történő paraméterátadással való összevetése. Érdekes túlzása a C#-nak az out kulcs-szó kiírásának híváskori kényszere. Hasonlóan igen sunyi hibákat okozhat bizonyos **alapszavak hiánya**, mint pl. a típusokat jelölők hiánya. Tipikus hiba: a hibaüzenetet sem okozó típuskeveredés az explicit **típusdekla-**

⁶ Emlékeztetőül: \$xxx – skalárváltozó; @xxx – tömbváltozó; %xxx – hash-táblázat...

* Így jelöljük –a későbbiek során is– a megvitatandó kérdéseket.

ráció nélküli nyelvekben, amelyekben dinamikusan és automatikusan definiálódik egy-egy adat típusa. (Ilyen pl. a Perl, a JavaScript.)

Még egy adalék: a Perl paraméterezési szokásaiban a kulcs-szavak „furecsa jelekké egyszerűsödését” tapasztalhatjuk, ami igen veszélyes lehet...

```
my @s = ('1','2','3');
my @b = ('A','B','C');
print " s="."@s";    #s értéke a hívás előtt
print " b="."@b\n";  #b értéke a hívás előtt
elj(\@s,\@b);        #eljárás-hívás két tömbcímmel
exit;
sub elj {
    my ($s,$b) = @_; #paraméterátvétel
    print " s="."@$s";    #s értéke belül
    print " b="."@$b\n";  #b értéke belül
}
```

Output:

```
s= 1 2 3 b= A B C
s= 1 2 3 b= A B C
```

2. Perl programrészlet
(Apró különbség a *cím*hivatkozás.)



Érdemes elgondolkodni azon, hogy *mikor?*, *mi?* a jobb: világos kifejezése annak, hol és mi az alapszó, vagy „ahogy tetszik” írni be őket. (*Programozási stílus*= *Kifejezőség*, *precízség* ⇔ *Programírási egyszerűség*.)

2.1.2. Egyszerű programszerkezet

Világosan **átlátható**, **memorizálható** programszerkezet.



Nagy kérdés, hogy a BASIC (PHP, Perl) egyszerűsége, „szabad” programozási lehetősége (változó **deklarációk** elhagyhatók, vagy szabadon, **ad hoc módon elhelyezhetők**) hasznos-e. Ide illik a C++ „*deklaráció per hecc*” szokása. Pl. `for (int i=0; i<n; ++i) {...}`. Hajlok rá, hogy hasznosnak minősítsem, mivel nyomatékosítja (sőt garantálja is!), hogy csak a ciklusban legyen használható!

A **szigorúság-kifinomultság** ára: lásd az Eiffel-t, az Adát vagy a korábbi nyelvek közül a PL/I-t, amelyek túlön-túl sok, látszólag –esetleg valóban– érthetetlen szabállyal terheltek.

Sarkalatos az oktathatóság szempontjából a **típusosság „mértéke”**. (Vö.: Pascal / C.) Jó indulattal egyfajta kifinomultságnak tekinthetjük a C-szerű nyelvek tömörségét eredményező *’értékadás a kifejezésben’* lehetőségét. Pl. az `’a=2+(b=5);’` utasítás valójában azonos a `’b=5; a=2+b;’` utasításkettőssel, mivel minden kifejezésnek, így az értékadásnak is van értéke, s persze érték-típusa. Vagy a szintaktikailag korrekt, szemantikusan értelmetlen: `’i=i++;’` kifejezés. Tehát ismét felvethetjük a *’tömörség vagy világosság’* kérdését.

2.1.3. Következetes programszerkezet, következetesség

Következetesség = kevés szabály (kevés kivétel) ⇒ kitalálhatóság.

A szintaxisban érthetetlen „**környezetfüggőségek**” tapasztalhatók pl. a Pascalban: a VAR és CONST *eltérő* alkalmazás a kétféle deklaratív részben, nevezetesen a lokális adatok és a formális

paraméterek megadásánál. Ide sorolható a WHILE és REPEAT ciklusok szokásos eltérő feltétel értelmezése okozta bizonytalanság.

Másik példa a C++ típusdefiniálása: `tip struct{...}` vagy `typedef struct{...} tip`, és `typedef etip tip[...]`, azaz a definiált típus helye hol itt, hol ott, addig pl. a

Pascalban egyöntetű: `Type tip=Record .... End, Type tip=Array [...] of ...`.

```
Var x: ...; y: ...;

de

Procedure ... (Var x:...; Var y:...);
```

Két Pascal programtöredék

Könnyen megjegyezhetők pl. az **elválasztó jelek**. Ellenpélda: a Pascal vessző, pontosvessző használata a paraméterezésben (aktuális, ill. formális paraméterek), vagy mikor kell vessző, mikor lehet, mikor nem szabad (lásd IF-THEN-ELSE).

Összetett szerkezetek eleje-vége jelzésének a kérdése. Pl. LCN Logo és a Python bekezdéses tagolás, kontra Logo WRITER „kikényszerített” hányaveti sorfogalma; vagy más ellenpélda: Pascalban BEGIN-END sokszor, de néha CASE-END, RECORD-END, REPEAT-UNTIL; C++ esetén a ciklusmag vagy az elágazás ágai körül elhagyható (egy-utasításos esetben) a `{}` zárójelpár... jó példa: ELAN-ban **IF-ENDIF**, **REP-ENDREP**... Maple-ben: if-fi, do-od, UNIX shell: if-fi, for do-done, case-esac (☺)...

Hierarchikus építkezés (a felülről lefelé tervezés kódtükröződése) –szintenként azonos gondolattal építhető program.

Másfajta következetesség jó példaként dicsérhetjük a Perl/C értékadás-szerű operátorainak családját: „+=”, „-=”, „*=”, „/=”, „**=”, „%=”, „&=”, „|=”, (Ez hasonlóan meg van számos C-szerű nyelvénél.) Mindazon által megvan ennek „kockázata” is: pl. az alábbi kifejezések szintaktikailag közeliek, míg szemantikusan roppant eltérőek: `i+=1` vs. `i=+1`.

2.2. Tipikusság

Milyen könnyű átvinni az első programozási nyelv tapasztalatait a későbbiekre?

2.2.1. Egyszerű kódolás, könnyen tanulhatóság (⇐ algoritmikus nyelv)

Nem tér el lényegesen az algoritmikus nyelvtől, annak csak „precizírozása”. A kódolás érdekében **ne kelljen** a programon **lényeges átalakításokat végezni**. Gondoljunk például a függvény érték-típusára tett korlátozások miatti eljárásra történő kényszerű áttérésre ANSI Pascal esetén (a Free Pascalban már megengedett a rekord-, tömbértékű függvény), vagy egy másik tipikus „konverziós” kényszerre: a többirányú elágazás ⇒ IF-ekké; s még egyre: a REPEAT-UNTIL ciklus kilépési feltétel negálására...

Jó példa a Perl „ad hoc struktúrakezelése” értékadásokban: `($Van, $Melyik) = &Kereses($N,@X)`, amely jól illeszkedik az algoritmikus nyelv, sőt a specifikációnál alkalmazott utasítás rövidítésekhez.

Veszély forrása a C-szerű nyelvek az algoritmikus nyelvben meghonosodottól eltérő operátor-szintaxisa: `:=` (értékkadás) helyett `=`; `=` helyett `==`. Különösen a korábban (2.1.2.) említett *értékkadás a kifejezésben* szintaktikai megfelelés mellett!

2.2.2. Jó modellje nyelvosztályának ($\Rightarrow$ más nyelv)

Következtesen, érzékletesen tartalmazza azon jellemzőket, amik **lényegesek az osztálya szempontjából**.⁷ A Logo „eklektikussága” ebből a szempontból nem ideális: nem mutatja meg világosan sem az *automata*, sem a *funkcionális* nyelvek jellemzőit.

Legyen jó alap a **tovább lépéshez**. Pl. a Pascal után az OOP vagy a 4GL folytatásra lehetőséget kínál a Delphi, ill. a Lazarus. A C-nek is számos alkalmas „folytatója” létezik.

2.3. Használhatóság

*Milyen könnyű elviselni a kialakult programozási szokásokat
programírás közben
és magát a programot használat közben?*

2.3.1. A fejlesztés közben – támogatja a haladó programozási stílust

Mivel az oktatásba alkalmazott nyelv a kialakuló **programozási stílust** is alaposan befolyásolja, ezért ez a szempont különösen fontos szerepet játszik a nyelvválasztásban.

A legfontosabb „stílusjegyek”:

- a) **felülről lefelé programtervezés**, mint gondolkozást meghatározó stratégia – finomítások, paraméterek (vö. Pascal, ELAN, Perl, C++);
- b) **algoritmikus absztrakció** – szekvencia, elágazások (l. C switch-beli break „elhagyhatósága”), ciklusok, eljárások / függvények / operátorok; vannak-e egyáltalán operátorok (az utóbbiak léte az oktatás későbbi fázisában, az adatabsztrakció fontossá válásakor lesz lényeges);
- c) **adatabsztrakció** – elemi típusok választéka, *felsorolási típus* (I/O-műveletek!), típuskonstrukciós eszközök (pl. rekord Pascalban és C-ben, vs. tömb Pascalban és C-ben), a *típus* korrekt megvalósíthatósága (vö. BASIC, Pascal, ELAN, objektumorientált nyelvek);
- d) **mentesség az „illegális” és veszélyes lehetőségektől** (pl. a Logo-ban nem lehet(ne) értékkadás; az algoritmikus nyelvekben GOTO, STOP, HALT, EXIT tilos, helyettük kifejezett *kivételkezelés* ajánlott; e lehetőség nélkül sajátos *kódolási szabályok* szükségeltetnek⁸);

Ciklusváltozók és a „lokalitás” kényszere:

```
Var i: Integer;
Procedure A;
Begin
  For i:= ... do ...
End;
Begin
  For i:= ... do
    Begin ... A; ... End;
  End.
```

Side effect:

```
Function fv (Var x:...): ...;
...
Begin
  ...
  b:=fv(a)+fv(a); { b? }
  b:=2*fV(a);      { b? }
  ...
End.
```

*Két probléma,
két Pascal programtöredék*

⁷ Ezt minden egyes nyelvosztályra érdemes lenne külön meggondolni.

⁸ Lásd Szlávi Péter – „A Programkészítés didaktikája.”, PhD értekezés 104-107. oldalain! [Az ide vonatkozó [részlet.](#)]

- e) **mentesség a nehezen észrevehető mellékhatásoktól** (pl. függvény paramétere megváltozik, lokális adatok láthatósági kérdései, vagy érték nélküli visszatérés egy függvényből);
 - f) **beszédes azonosítók** használhatósága (vö. ősi BASIC $\Leftrightarrow$ ELAN; Perl „változó-előkéi” [\$, @, %...]);
 - g) **modularitás** lehetősége pl. a típusmegvalósításhoz (pl. a Pascal include-ja, unitja, a Modula és a COMAL modulja, az Ada package-e, az ELAN packja, C# namespace-sze...)
- Megjegyzés: a Pascal hiányzó *generic* fogalma részben pótolható az ellenőrzött *include*-dal⁹, mindenesetre szóba hozható pl. az Ada, C++ irányába mutató hiányként.

2.3.2. A használat közben – rendelkezik minimális kódhatékonysággal, nyitottsággal

A **konkrét** gép/operációs rendszer **adottságaihoz** (minimális mértékig) **alkalmazkodjon** a nyitottsága által: (ko)processzor, memóriamodellek, grafikai lehetőségek, egérkezeléshez rutinok...¹⁰ Mindazon által ne függjön tőlük meghatározóan, kizárólagosan. Egyfajta nyitottságnak tekinthető egyes böngészőben futó nyelvek (pl. JavaScript) azon jellemzője, amely által képesek a HTML folyamatosan megújuló lehetőségeihez (pl. grafika a <canvas> elemre építve) alkalmazkodni.

Tartsa meg a „**téma-univerzalitását**”! Bosszantó, ha bizonyos problémák vizsgálatától nyelvi korlátok miatt kell eltekinteni. Például: kivárhatatlan lassúság (szimulációnál), a grafika teljes hiánya stb. A lassúság oka persze lehet egy, a nyelvhez kötődő végrehajtási stratégia (interpretáltság) is. Ez is számításba veendő! Az újkeletű témákhoz („webológia” – böngészős alkalmazások írhatósága; „telefonía” – mobil programozhatóság) való nyitottság nagy előnyt jelent!

Automatikus és „kézi” **hibafelderítő mechanizmusok** (indextúllépés, típusértés) legyenek beépíthetők, ill. kikapcsolhatók. (V.ö.: Turbo Pascal/Geany+Free Pascal, Code::Blocks környezetben beállítható fordítási opciók és feltételes fordítási lehetőségek.) Ez által használhatók a fejlesztői környezet programkészítést támogató funkciói a végrehajtás hatékonysága (gyorsaság, kódtömörség) érdekében.

2.4. Fejlesztői környezet léte

„Ó, azok a csodálatos, hősi évek...
egy program = egy félév”

Eddig a nyelvről *általában* esett szó (a 2.3.2. kivételével, melyben már felbukkantak a környezet adta lehetőségek is). Ettől kezdve a **megvalósítás** mikéntje a legfontosabb kérdés, mellesleg tovább finomítjuk a nyelvi „jószág” szempontjait.

2.4.1. Integrált nyelvi környezet

Ellenpéldák a korai professzionális nyelvek: programszerkesztés + fordítás + futtatás + programjavítás papíron + ugyanezek újra. (Bár bizonyos nyelvi környezetek ma¹¹ is ilyenek. Lásd UNIX-os Adát; vagy a szkript nyelveket, a böngészőben futó kódok nyelvi környezeteit általában.) Ezen

⁹ A 2.6.4-es fordítóban már létezik osztállyal kapcsolatban a *generic* fogalom!

¹⁰ Persze, ha ezek nincsenek a lényeggel ellentmondásban.

¹¹ Az ezredforduló táján.

némileg segítenek az „univerzális” (szintaxist ismerő) **programszerkesztők**. (Pl. BSEdit, Notepad++, Geany...)

2.4.2. Párbeszédesség a programfuttatásban

Interaktív futtatási lehetőségek: nyomkövethetőség, ..., adatok menet közbeni lekérdezhetősége, módosíthatósága. Ez a programmal való „együttlélegzés” záloga; s ilyenformán a programozási kedvet, a program iránti „empátiát”, a fejlesztés hatékonyságnövekedését szolgálja.

A nyomkövetés **módszertani jelentőséggel** is rendelkezik a programozástanítás kezdeti szakaszában: lehetővé teszi a „a számítógép hogy csinálja?” kérdés mélyebb megértését. Másrészt viszont elkényelmesíthet: csábít a gondolkodás nélküli hibakeresésre.

2.4.3. Súlyok

Az oktathatóság szempontjából különösen nagy jelentőséggel bír a súlyok léte, milyensége. Nagy nehézséget jelent, ha a súly kimerül egy internetcímre való utalásban.

Helyzetérzékeny segítő információk megjeleníthetők legyenek. Nagyra becsülendő az a keretrendszer, amely rendelkezik automatikus kulcsszó-magyarázó buborékokkal.

2.5. Szabványosság

Olyan-e a nyelv, mint amilyennek képzelem?

2.5.1. Dokumentáltság

A mai „shareware-world”-ben különösen fontos kérdés: van-e a kósza híreken, és a kísérletezésen túl más is, ami alapján **megismerhető a nyelv?** (*Szabványok, dokumentáció... ⇔ OEM¹²?*)

2005. év aktualitása volt az ELTE IK-n: a FreePascal–Lazarus bevezetése; ennek tapasztalatai:

- komoly kockázatai vannak egy *kiforratlan szoftver* (pl. fordítóprogram, fejlesztői környezet...) bevezetésének;
- még abban az esetben is, amikor egy jól ismert „ős” (jelen esetben a Pascal, a Delphi) leszármazottjaként jön világra az új szoftver, kidolgozatlan *súly nélkül* bevezetni az oktatásba;
- nagy *többlet terhet* jelent a bevezető tanárnak (mindent előre, akkurátusan ki kell próbálni, és ki kell dolgozni kerülő utakat a fellelt hibákhoz, hiányosságokhoz, nem is beszélve a diákság számára készítendő segédanyagokról, keretprogramokról). Hogy csak egyetlen példát említsünk, a TP-FP áttérés nehézségeit: 1-ről 2-ablakúság, a grafikus képernyő (=ablak) felbontásának (operációs rendszertől függő) megnövekedése.

2.5.2. Elterjedtség

Egyedi –bár csudajó– nyelv –korlátozott, azaz *egy* iskolabeli– oktatásba való bevezetése **nem praktikus**. Ennek köszönhető pl. a COMAL elhalása. Két nem oktatási példa: ALGOL nem

¹² Az OEM (Original Equipment Manufacturer) program lényege, hogy a Microsoft a hardvergyártókon és forgalmazókon (rendszerépítőkön) keresztül, meghatározott hardver eszközökkel együtt a számítógépre előtelepítve forgalmazza termékeit.
(Lásd <http://www.microsoft.com/hun/jogtisztasag/31.msp> és <http://www.microsoft.com/hun/oktatas/oem.msp>.)

terjedt el igazán a programozási gyakorlatban, pedig ő az őse a Pascalnak, Adának is, míg a FORTRAN / BASIC igen, pedig...! Az ok kézenfekvő: olyan alapvető rész maradt kidolgozatlan a nyelvben, mint az I/O. Persze a cél egy *publikációs nyelv* definiálása volt. Hasonló sorsra jutott az APL (=„A Programming Language”) is, amely speciális jeleit bebillentyűzni sem lehet akármilyen környezetben (különleges billentyűzetet igényel), jól lehet, erejét sokan dicsérik (lásd az IBM 360 számítógép leírása).

Az elterjedtség klasszikus rossz példája: a BASIC. A személyi számítógép korszak hajnalán ahány fajta gép, annyi BASIC-**nyelvjárás** született, a számos szabványosítási kísérletek ellenére.

2.6. Fejlesztői környezet finomabb részletei

A nyelv birtokba vételének ára... „Anyám, nem ilyen lovat akartam” I.

2.6.1. Beépített programszerkesztő milyensége („alázatosság” / „erőszakosság”)

Jó-e, ill. mennyire jó, ha a programszerkesztő kifejezetten a *nyelv* vagy egy (*módszertani, oktatási* ...) *cél* „logikáját” követi, **erőlteti**? Nem biztos. Lásd LCN Logo, ELAN. Miért mondható sikeresnek a „Turbo-szerkesztő”? (Elterjedt, a legfontosabb szövegszerkesztési és néhány speciális programszerkesztési funkciót megvalósít.) De az sem célszerű, ha a programszerkesztő bántóan igénytelen, amilyen pl. a LogoWriter-é. Számos *fejlesztői környezet* (sőt fordítótól független *programszerkesztő*) képes „szép” programstruktúrák *automatikus* létrehozására, a bebillentyűzés támogatásra, ill. kulcs-szavak (sőt esetleg más lexikális elemek) *kiemelésére*. Alkalmassint a lexikális elem első néhány jele alapján felkínálja az odailleszthető lexikális elemeket; elárulja pl. egy eljárás paramétereinek típusát stb.



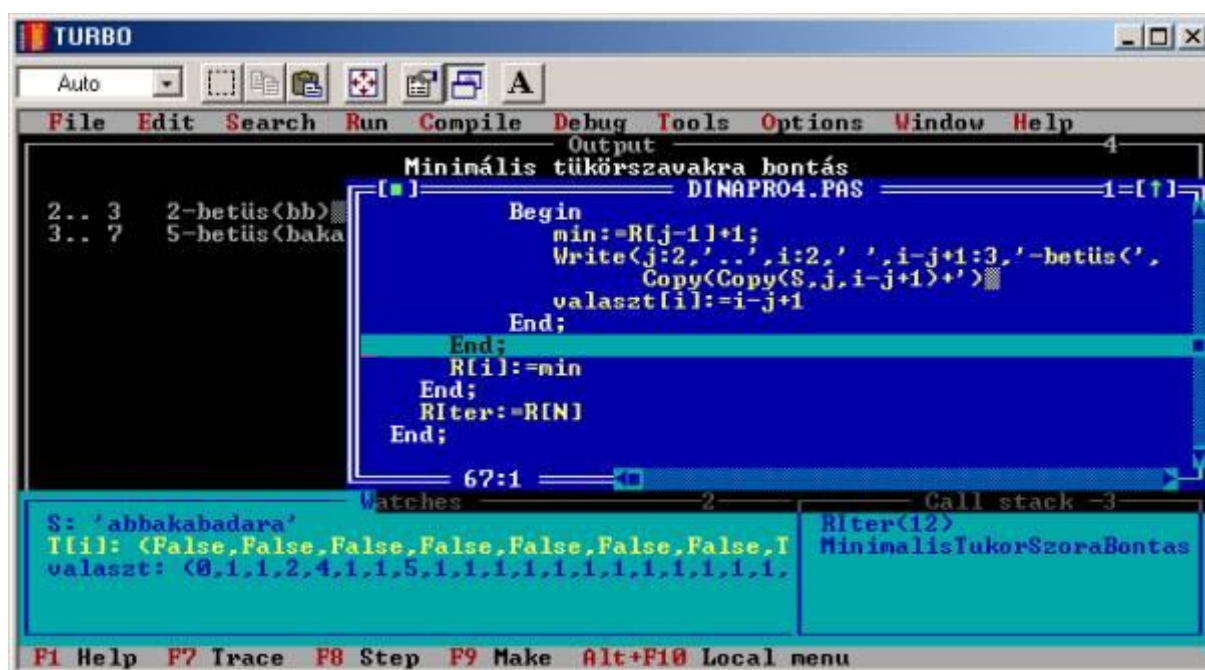
Mi tehát a baj az LCN Logo „konzekvens” programszerkesztőjével? Szokatlan a funkcionalitást tükröző hierarchikus programstruktúra, pláne *ahogy* kényszeríti a bevitelt¹³... És az ELAN programszerkesztőjével? Fókusz-fogalom nehezen áttekinthetővé teszi a program egészét.

A milyenség más szempontból is érdekes: illeszkedik-e ahhoz a paradigmához, amelyet programozás szempontból megvalósítani igyekeznek. Pl. a Turbo Pascal az objektum fogalmat teljesen alkalmatlan módon támogatja: minden kapcsolódó fogalmat (pl. metódus, property, „objektum-hierarchia”...) kulcsszó-szinten kell ismerni, semmi segítséget nem nyújt az ismerkedő számára. Ablakközéppontú programozáshoz elengedhetetlen a GUI-s, eszközpalletás, ... fejlesztői környezet. (4GL)

2.6.2. Beépített futtató-/nyomkövető-rendszer

Több-ablakos megjelenítésű nyomkövetési rendszer hasznos (a párhuzamos, sok oldalról történő szemléltetés miatt): algoritmusok (tételek), paraméterezés, lokális-globális adatok lényegének megértetésénél, rekurziónál stb.

¹³ Lásd <http://people.inf.elte.hu/szlavi/InfoOkt/LCNLogoPl.htm>



2.6.3. Fordítási lehetőségek

Biztonságos vagy hatékony kód generálása közötti **választást** „lokálisan” (egy-egy kód darabra fókuszálva) fölkinálja-e (opciók)? Ilyen lehetőségeket kínál fel pl. a Turbo és a FreePascal API (de meggondolandó a Geany-n keresztüli „lehetőség”), a Code::Blocks, a Visual Studio fejlesztői környezet stb.

Többplatformúság előnyös lehet, hiszen támogatja az esetleges operációs rendszerváltoztatást. De figyelem! A végrehajtásnak **szemantikusan ekvivalensnek** kell lennie. Sajnos messze nem igaz ez –ma még– a böngészőkben futó programokra (pl. a JavaScriptre sem).

Moduláris programozási lehetőségek oktatási szempontból is praktikusak: hisz elrejthetők a preparált célmodulok, amikkel a tanulás hatékonyabbá, élvezhetőbbé tehető (grafikai „primitívek”, fájl-, menü-kezelési stb. modulok, vagy gondoljunk a backtrack tanításánál egy sakkfigura-rajzoló „háttérmodul” mit jelenthet ¹⁴). Egy 4GL-környezet szinte kínálja ezt a modulokra épülő, „keret-programos” oktatási stratégiát, amelyben az alkalmazói felületet kezelő modult készen adhatjuk a tanulóknak, s nekik „csak” a lényegét jelentő modul elkészítése a feladat.

2.7. Biztonságosság

A nyelv megtartásának ára... „Anyám, nem (is) ilyen lovat akartam” 2.

Azt teszi-e a lefordított, interpretált program, ami várható tőle, nincsenek –egy nem vájt fülű számára– **megmagyarázhatatlan „effektusok”**. (Geanyben vagy Code::Blocks-ban fordított C++-os példa: nem ugyanúgy működik exe-vé téve, mint a fejlesztői környezetben [megálláskor automatikusan vissza a szerkesztő ablakba, de gondok jelentkeznek nagy memóriát igénylő programok

¹⁴ Lásd <http://people.inf.elte.hu/szlavi/InfoOkt/ModularisPascalPI.htm>

esetén, ill. a grafikus programok nyomkövetésekor is]. További sok példát sorolhatnánk a Lazarus ma még béta változatai kapcsán.)

A **kezdőértékek** kérdése: deklarációban (a változó létrejöttékor) kap-e a változó valamilyen *definiált* kezdőértéket vagy sem (*véletlen állapot anomáliái*). Az interpretált nyelvek esetében gyakorta a változók egy jóldefiniált „undef” kezdőértékkel jönnek létre, ami lehetővé teszi legalább a futás közbeni észlelést és esetleges hibajelzést. (Sőt a Perl által generált kód is ilyen!)

A fejlesztői környezet „szigora” összefügg a nyelvvel magával (pl. típusosság), de a filozófiával is (fordít vagy értelmez), sőt a konkrét megvalósítás minőségével is. Pl. egy publikus JavaScript mintaprogramban találtam teljeséggel fölösleges és szintaktikusan hibás sort is, amellyel (és nélkül is) helyesen fut a program.

Végrehajtási stabilitást is a biztonságossághoz sorolom. Ez alatt azt értem, hogy a választható végrehajtó környezet milyenségétől függetlenül *azonosan fut-e* a készített program. Ebből a szempontból kritikusak a szkript nyelvek. (Az előbb említett JS-es példaprogram másként működik Explorerben, mint Firefox böngészőben!) Sajnos nemcsak a szkript nyelvek szolgálnak „jó” példával. Ui. a FreePascal ($\leq v1.0.10$) végtelen ciklusba esik textfájlolvasás közben, ha az utolsó sor végén nincs sorvégjel, míg a Turbo Pascal helyesen lefut. Hasonló a probléma a FreePascal 2.6.4 Graph unitbeli hiba miatt: egy korábban még jól futó, grafikát is használó program a 2.6.4-es verzióban fordítva szemantikusan rosszá válik. De hasonló jelenségek adódtak a gcc fordító egyes verzióváltásaikor is.

2.8. Bonyolultság – egészében

*Van királyi út? avagy
Megmászható-e profi hegymászó felszerelés nélkül is?*

Záró szempontként egy utolsó: a vizsgált nyelv egészében mennyire tűnik bonyolultnak a kezdők/haladók nyelveként?

Következetesség $\Leftrightarrow$ *egyszerűség*. (Lásd [2.1.1-2.](#))



Van-e egyszerű út is a megközelítéséhez egy még nem szabványosodott lehetőség (OOP, grafika, hang), vagy túl nagy az első sikerélményhez megteendő lépés? (Lásd Windows-os, OOP-s programírása Pascalban egykor $\Leftrightarrow$ Delphiben ma.)

3. Az egyes nyelvek értékelése

Az alábbi táblázatokat ki kell tölteni –csoportosan– úgy, hogy az egyes szempontokra 0-5 pont adható (0=nincs, vagy rossz; 5=kiváló), továbbá, a szempontok fontossági sorrendjét is 0-5 számokkal jelöljük ki (0=nem fontos, 5= nagyon fontos).

P1.: „Értelmes alapszavak” 5/4 (jelentése: Kiváló/Nagyon fontos)

3.1. FreePascal

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.2. PHP

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.3. Visual BASIC

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Súgó (,,hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.4. JavaScript

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.5. Turbo Prolog

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.6. Code::Blocks – C++

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédesség a programfuttatásban										
2.4.3. Súgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.7. C#

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédeesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.8. Visual C++

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédeesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.9. Scratch

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.10. Comenius Logo

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sógók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.11. Delphi

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.12. Python

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.13. Java

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.14. Perl

Mennyire „szimpatikus” globálisan?										
Mennyire ismerem?										
1.1. Értelmes alapszavak										
1.2. Egyszerű programszerkezet										
1.3. Következetesség										
2.1. Egyszerű kódolás, könnyen tanulhatóság										
2.2. Jó modellje nyelvosztályának										
2.3.1. Támogatja a haladó programozási stílust										
2.3.2. Rendelkezik min. kódhatékonysággal, nyitottsággal										
2.4.1. Integrált nyelvi környezet léte										
2.4.2. Párbeszédekesség a programfuttatásban										
2.4.3. Sűgók („hol/miben/milyen” léte)										
2.5.1. Dokumentált										
2.5.2. Elterjedt										
2.6.1. Beépített programszerkesztő („alázatos / erőszakos”)										
2.6.2. Beépített futtató-/nyomkövető-rendszer										
2.6.3. Fordítási lehetőségek										
2.7. Biztonságosság										
Saját szempont:										
Saját szempont:										
Saját szempont:										
Saját szempont:										

II. ALKALMAZÓI RENDSZEREK

1. Az Alkalmazói rendszerek tanításának céljai

1.1. Amatőr (hétköznapi) alkalmazó

Mindenki fogja használni, tehát ilyenek ismerete az **általános műveltség** része. Az alapfogalmak és az „alapfogások” megismertetése a cél.

1.2. Rendszertípus megismerése

Valószínűleg nem pont a tanult rendszert fogja később alkalmazni, ezért fontos, hogy magát a „kategóriát” is megismerje: funkciók, típusfeladatok és megoldási lépések...

1.3. „Profi” alkalmazó

Munkakörök egy része (titkárnő, adminisztratív munkakörök stb.) **profibb alkalmazói ismereteket** igényel, ezért az ilyen helyre orientálódóknak fontos ebben tökéletesedni. (Lásd [ECDL](#), [OKJ](#)-s vizsga.)

2. Értékelési szempontok

2.1. Egyszerűség

2.1.1. Menük

Hierarchikus **menürendszer**, fontos az arányos és logikus csoportosítás. Rossz példaként említhető a WinWord 2.0 esetén a 'Fattyú/Özveg sorok' a 'Nyomtató' opcióba helyezése, a WinWord 6.0 esetén a 'Fájl' menübeli 'Oldalbeállítás...' -ba. Később került a mai 'Formátum/Bekezdés/Szövegbeosztás' menüsorral kiválasztható paraméterablakba. Nagyon kétséges az Office 2007 „felhasználó kezéhez simuló”, szokatlan és –rutinos és beavatatlan használók számára egyaránt– konfúzus menürendszer.

2.1.2. Ikonok

Ikonikus eszköz-sor (ami „analfabétáknak” különleges előnyt jelent ☺), jó, ha van, de a túlzásba vitele ellentétes hatású. Itt is ügyelni kell az ikonok kifejező (és megfelelő méretű) voltára. Jó példa: WinWord 97 „**összeválogathatóság**” szolgáltatása (Testreszabás, 1. módszertani szempontból lényeges más vonatkozását is: [2.4.1.](#)). Az ikonok **méretnövelhetősége** gyengénlátók iskolájában való alkalmazhatóság feltétele.

2.1.3. Sógó

Van-e egyáltalán beépített sógó, ha van, használhatók-e szövegezésüket és terjedelmüket tekintve. Ha a **szakszavakat** előzetes magyarázat nélkül használja, a kezdőknek nem segítség. Ugyancsak

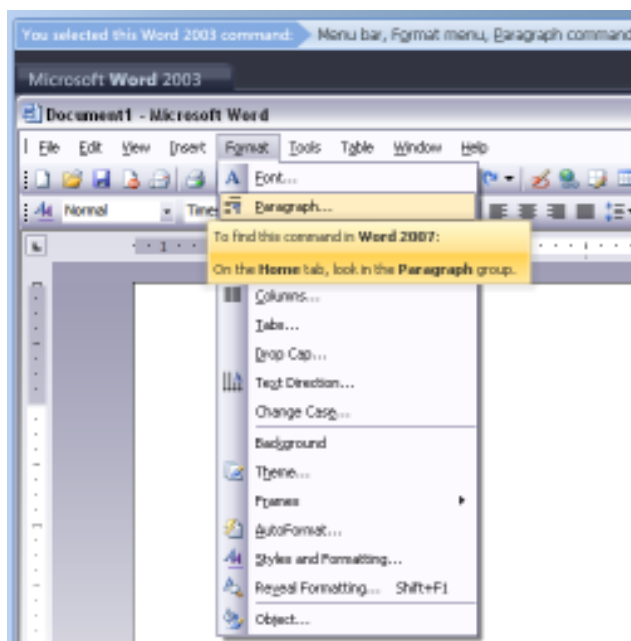
hasznavehetetlen a rosszul, **magyartalanul** („félmagyarul”) megfogalmazott súgó. Probléma, ha csak **idegen nyelven** hozzáférhető. (A többnyelvűségnek van kihasználható oktatási jelentősége!) Szerencsés, ha jellegzetes **példákat** is tartalmaz. Jó példa az Excel-beli függvények némelyikéhez tartozó súgórész.

Helyzetérzékeny módon legyenek a **segítő információk** megjeleníthetők. Hasznos az automatikus ikon-magyarázó buborék: a „hint”.

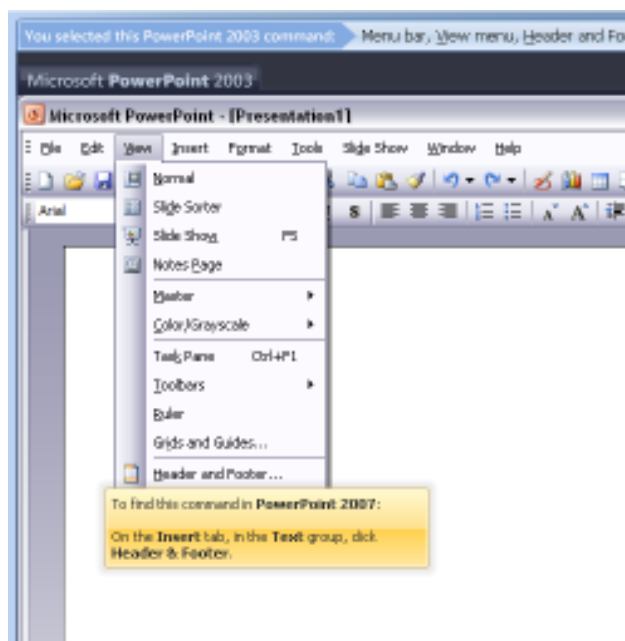
2.1.4. Uniformitás

„**Hasonlóság**” más szoftver-felületekhez. **Uniformitás** – pl. billentyű konvenciók (F1 = Help ...), hasonló elhelyezkedésű és jelentésű menük stb. (Lásd Win98 és utódai – Explorer GUI „általánossá válása” pl. az Intéző esetén.) A szakma által elfogadott, **egységes terminológia**-, fogalomhasználat. Különösen megneveziténé az egyedi szókincs egy más, hasonló célú szoftverre való áttérést. (Gondoljunk olyan alkalmazásokra, amelyek a hétköznapi szókincsen túl, speciális fogalmakat, elnevezéseket is használnak, ilyenek pl. a grafikus vagy a prezentációkészítő szoftverek.)

Az uniformitás nem tisztelete sok bosszúság forrása lett pl. az Office 2007 GUI megreformálásakor. Ezen még „interaktív segítség” (a <http://www.rufusz.hu/hirek/interaktiv-segitseg-az-office-2003-rol-2007-re-valo-attereshez> címen elérhető webes, online súgó) sem tud hatékonyan segíteni, amely „megmutatja” az egykori menüpont új feltalálási helyét.



A webes súgó a Word „Paragraph...” menü új elérését magyarázza.



A webes súgó a PowerPoint „View/Header...” menü új elérését magyarázza.

2.1.5. Egyebek

Egér és klaviatúra funkcionalitás tekintetében **egyenértékű** használhatósága.

„**Előreláthatóság**”, **következetesség**: kitalálható egy-egy funkció léte, működése (és szintaxisa) az addig megismert rokon funkciók alapján. Pl. az Excel 2003-ban DARABTELI függvényének paraméterezése eltér attól, amit várnánk az elemibb SZUM, ÁTLAG, MIN, DARAB stb. alapján. Érthetetlen módon rossz (bár nem szintaktikusan hibás!) a `'=DARABTELI($B$2:$E$2;"=A1")'`

függvényalkalmazás, míg helyes a '=DARABTELI(\$B\$2:\$E\$2;A1)' és az általánosításra is alkalmas következő alak: '=DARABTELI(\$B\$2:\$E\$2;"="&A1)'.

2.2. Vizualitás

A **WYSIWIG**-ség alapkövetelmény az általános használatban, de különösen az oktatásban. Egykori rossz példa: az egykori $\text{T}_{\text{E}}\text{X}$ dokumentumszerkesztő igen **átgondolt lehetőségei** (matematikus akkurátusság tükröződik benne) és **igénytelen** „maj'meglátod'a'nyomtatáSután” ígértű **külső**. (Már vannak WYSIWYG jellegű szerkesztők is.)

Vizuális lehetőségek mind teljesebb beépítése. Pl. egy **szövegbe grafika** beilleszthetősége, egy **táblázatba** változatos, **kifejező hisztogramok** generálhatósága, egy **prezentációba** a vetítés menetének **animációs** lehetőségekkel segítése, dinamizálása.

2.3. Teljesség

Jól „**modellezze**” azt az alkalmazói programcsaládot, amelynek tagja. Ez vonatkozik a szoftver „**filozófiájára**”, a **funkciókra** és **fogalmakra** egyaránt. L. [3. fejezetet](#).

2.4. Rugalmasság, „testre szabhatóság”

Különösen fontos, hogy az **oktatás** adott fázisában a pillanatnyi célhoz, célközönséghez „**idomítani**” lehessen. A **hétköznapi alkalmazásban** sem nélkülözhető a tágabb (szoftver-, probléma- és igény-) környezet figyelembe vétele, az ahhoz való igazíthatóság képessége.

2.4.1. Menük és súgók

A menük, a súgók **magyar(osítható)sága** az oktatásban elemi elvárás.



A **menük szűkíthetősége**, **átcsoportosíthatósága** előnyös lehet különösen a tanulás kezdeti stádiumába. (Lásd Quattro Pro, Word 97 újítása: „Testreszabás” – eszközsorok, menük, más beállítások [pl. nagy gombok].) Figyelem, ennek is lehetnek veszélyei! Előbb-utóbb **át kell térni a „szabványos” környezetre**.

2.4.2. Más szoftverek

Más szoftverekkel is legyen „beszélő viszonyban”: adatcsere, adatkonverzió. **Adatcsere** –pl. lásd Windows-os alkalmazások többségét: Write/Jegyzetömb-Excel-Word-Paint...–, amelyben a **vágólapon** keresztül történhet az objektumok beillesztése, csatolása **Adatkonverzió** alatt értem a különféle fajtájú, „rendszerbe illő” dokumentumok **exportálhatóságát-importálhatóságát**. Pl. egy szövegszerkesztő tudjon beolvasni és kiírni a saját formátumán túl **rtf, pdf, html, xml** alakban is, vagy egy táblázatkezelő tudjon **csv, tab**-bal elválasztott „szövegmezőkből” álló táblázat, **xml**-ben definált struktúra beolvasására, kiírására; vagy egy adatbázis-kezelő számára elfogadható input legyen bármely **standard relációs adatbázis**, de akár **xml**-struktúra is...

2.4.3. Makrók, programozás

„Taníthatóság”, azaz **makrózás** v. –magasabb szinten– „**cél-programozás**” lehetősége. De bármilyen programozási képesség előnyös lehet (pl. adatbázis-kezelők esetében **SQL**-irány, vagy más

esetben a **Visual Basic**). A programozhatóság különösen érdekes, ha valamely, önmagában is érdekes nyelv irányába mutat: pl. ORACLE-beli eljárások Java-ban is megfogalmazhatók.

Megjegyzések:



1. **motiváció** – a saját munka egyszerűsítése,
2. éppenséggel előnyösen használható **algoritmizálás bevezetésére**, gyakorlására.

2.4.4. „Intellisence”

Lényege: **automatikus javítás** (lásd helyesírás-javítás, rövidítések kifejtése), vagy a „**gondolatok kitalálása**”. (Pl. Excel – a szöveg kezdetéből megpróbálja kitalálni a folytatást, Word – dátum elejéből a mait.)

Azonban ne felejtsük el ennek **árnyoldalait** sem! Hogy a leggyakoribb bosszantó automatizmust említsem: egy számozott felsorolás száma után kérdés nélküli, szándék ellenére történő nagykezdőbetűre átváltás. Ennek a jó esetben egyszerűsítő szolgáltatásnak akkor van **használhatóságot növelő** szerepe, ha a használnak is van módja a **rendszer intelligenciáját bővíteni** (pl. Excel egyéni listáinak definiálhatósága), vagy –horribile dictu!– **kikapcsolni**.

2.5. Megbízhatóság, biztonságosság

2.5.1. Visszavonás

Visszavonhatóság –esetleg– **több lépcsős lehetősége** (UNDO). Jó példák az Office 97-beliek (meg a „leszármazottak”). Egyszerűbb szövegszerkesztők esetén (pl. Jegyzetömb) csak visszavonás és a visszavonás visszavonására van mód.

2.5.2. Mentés

Automatikus és „**több-változatos**”, beállítható sűrűségű mentés (BACKUP). Ehhez persze jó, ha az *operációs rendszer* is *támogatást* nyújt pl. a fájlnev-szerkezettel („vezeték-” és „keresztnev”, azaz kiterjesztés, sőt verziószám). (Lásd VMS verziószám, ill. a UNIX puritán, kiterjesztés nélküli fájlnevek világát is!)

2.5.3. Megerősítés-kérés

Olyan esetben, amikor végérvényesen rongálódhat, **megsemmisülhet adat, rá kell kérdezni**, hogy „valóban így akarja-e”. A biztonságnövelés célját már nem szolgálja a „túl gyakori kérdezősködés” (sokkal inkább az idegbajét).

Betöltéskor „bizonytalan” (=nem saját formátumú) forrás esetén elvárható, hogy megerősítést, még jobb esetben, a választáshoz **döntést** kérjen a felhasználótól a forrás miben létéről, ahogy teszi pl. a Word és az Excel „idegennek” vélt szövegfájlok beolvasása közben.

2.5.4. „Azt kapsz, amit kapsz”

Valóban WYSIWYG, **mennyire** az? Kitartó rossz példa a MS Word 2007 (!), amelyben töréspontok („folyamatos”, „következő oldal” stb.) elhelyezése esetén gyakorta fordul elő az, hogy a képernyőn az anyag a beállításnak megfelelően, helyesen folytatódik a töréspont után, de a nyomtatott anyagban ettől eltérően (s persze rosszul).

Egy kis szójáték: What You *Get* Is What You *Think/Want*. Gondolhatunk a korábbi Word-ök frame-ful (azaz „keretbő”) dokumentumainak szerkeszthetőségére.

2.5.5. Stabilitás

Nem száll el semmilyen körülmények közt. Ellenpélda: a Word korábbi változatai „**keretbő**” dokumentumok esetén, vagy a Word 2000 **többfájlos** dokumentum ígérete: „izmosabb” fő- és aldokumentumok esetén. A Word-ök erre való hajlamának hivatalos elismerését jelenti az a funkció, amely ilyen események bekövetkezése utáni újrainduláskor (ami esetleg rögvest automatikusan is bekövetkezi) a baloldali panelon a (meg)sérült fájlokat felsorolja.

2.6. Kompatibilitás

2.6.1. Operációs rendszer-függetlenség

Kifejezett előnyt jelent, ha egy valamely platformon tanult szoftvernek **más platformon** is létezik változata. Pl. az OpenOffice.org Windows és Linux mellett Solaris-on, Mac-en is fut, míg az egyébként elgondolkodtatóan drága –alighanem a kihasználható képességeihez viszonyítva is túl drága– (Adobe) PhotoShop vagy (Corel PaintShop Pro csak Windows környezetben fut.

2.6.2. Változatfüggetlenség

Az **(alulról) kompatibilitás** szempont nem csak az oktatás szempontjából fontos. Bár természetesen komolyan nehezítheti a tanár dolgát, ha egy *korábbi változatban* készített dokumentumot az aktuális verziójú szoftver *nem vagy rosszul tölti be*. Vö. uniformitással (2.1.4.).

Másik probléma: „*extrémebb*” *jelek* megváltozása, kiesése verzióváltáskor vagy formátumcse-re alkalmával (pl. ékezetes betűk elromlása a ZurichCalligraphic betűtípus esetében, vagy doc ⇒ pdf/html áttéréskor a felsorolás jelek megváltozása) stb.

2.6.3. Hardverkörnyezet-függetlenség

Hardverkörnyezettől (nyomtató, monitor...) legyen **független** esetleg kis, paraméterátállítás árán! Néhány negatív példa: a Word-ben (bár nem csak az ő esetében!) *más nyomtató* választásakor *megváltozhat a szövegtördelés*, sőt a szöveg egy-egy része... (Lehet, hogy még hiányzó szabványok vagy létező szabványok be nem tartása áll a probléma mögött.)

2.6.4. Nyelvfüggetlenség

Az oktatás szempontjából alapvető elvárás, hogy a **menük és súgók** a diákok **nyelvén** „szóljanak”. Érdemes elgondolkodni persze a „**többnyelvűség**” egyéb célú kihasználhatóságán. Az OpenOffice.org e szempontból is kiváló, hiszen 2008 tájáig 20 nyelven érhető el.

3. Az egyes alkalmazói rendszer-osztályok értékelése

3.1. Táblázatkezelőkről

(Quattro/Excel/Lotus 1-2-3/OpenOffice:Calc)

Legfontosabb fogalmak:

- *adattípusok* (numerikus, szöveges, dátum stb.),
- *altáblák* mint műveletek „alanyai” (operandusai),
- *képletek* (alap aritmetika, statisztikai függvények, „programozási tételesek” [sum, max, look-up,...] stb.),
- *szerkesztési funkciók*,
- *adatbázis-funkciók*:
 - *keresés* kulcs-szerint,
 - *rendezés* kulcs-szerint,

Megjegyzés: fölhasználhatók az adatbázis-kezelők bevezetéséhez...

- *grafikonok, grafikai attribútumok* (fajták; kellékek: cím, címkék, tengelyszövegek és -léptékek stb.),
- *nyomtatás* (fej és oldallécek, altáblázatok stb.).

3.2. Szövegszerkesztőkről

(Jegyzetömb/Norton Editor/WordPerfect/Word/TEX/OpenOffice:Writer)

Legfontosabb fogalmak:

- *szövegegységek* (jel, sor, paragrafus, lap; és attribútumaik stb.),
- *szerkesztési funkciók*,
- *nyelvi szolgáltatások* (elválasztás, helyesírás-ellenőrzés, szinonima-, rövidítés-szótár stb.),
- *grafikák beilleszthetősége és viszonya a szöveggel* (elhelyezés, távolság, körülírtatás stb.),
- *nyomtatás* („megtekintés”=PREVIEW stb.),
- *haladóbb szövegegységek* (szekció, hasáb, rajzos „objektumok” mint pl. képletek, táblázatok stb.), útban a DTP felé,
- *haladó „könyvszerkesztési funkciók”* (automatikus tartalomjegyzék, tárgymutató-készítés stb.),
- *„interaktív”/multimédia szövegegységek* (hipertext-lánc, különféle médiájú „objektumok” stb.), útban a multimédia-, a weblapszerkesztés felé.

3.3. Adabázis-kezelőkről

(DBase III/Clipper/Fox Pro/Access/Oracle/OpenOffice:Base)

Legfontosabb fogalmak:

- *adattípusok* (numerikus, szöveges, dátum, esetleg „makroszkopikus” objektumok stb.),
- (elsődleges) *kulcs*,
- *adatbázis objektumok* (táblák, űrlapok, jelentések),
- *táblakapcsolatok* (1-1, 1-sok, sok-sok),
- *indexelés*,
- *szerkesztési funkciók*,

- „kuriózumok”: képletek (mezőkben), grafikonok,
- *nyomtatás* (összegfokozatos lista stb.).

3.4. *Ábrászerkesztő programokról*

(Paint/CorelDraw!/PhotoShop/Gimp/OpenOffice:Draw)

Legfontosabb fogalmak:

- *a priori objektumok* (pont, szakasz, törtvonal, hajlított ív, keret, doboz, ellipszisív, -lap),
- *rajzeszközök* (ecset, ceruza, kitöltő eszköz, szórópisztoly, szöveg) és attribútumaik,
- *objektum-műveletek* (mozgatus, színezés, forgatus, nyújtás, mintázás),
- *képméretezés, felbontás-megadás, színmélység-módosítás,*
- *kép-export, -import* (fájlajták, esetleg scanner),
- *haladóbb grafikai műveletek* (előre-, hátratos, 2 alakzat „fokozatos közelítése”, „effektusok”, színmodellek),
- *funkciók a 3D felé, az animáció felé,*
- *nyomtatás* (különféle minőségben és méretben).

Az alábbi táblázatoskat ki kell tölteni –csoportosan– úgy, hogy az egyes szempontokra 0-5 pont adható (0=nincs, vagy rossz; 5=kiváló), továbbá, a szempontok fontossági sorrendjét is 0-5 számokkal jelöljük ki (0=nem fontos, 5= nagyon fontos).

Pl.: „Egyszerűség” 5/4 (jelentése: Kiváló/Nagyon fontos)

Táblázatkezelők

	Quattro	Excel	Lotus 1-2-3	Calc		
Mennyire „szimpatikus” globálisan?						
Mennyire ismerem?						
1.1. Egyszerűség – menük						
1.2. Egyszerűség – ikonok						
1.3. Egyszerűség – súgók						
1.4. Egyszerűség – uniformitás						
2. Vizualitás						
3. Teljesség						
4.1. Menük és súgók						
4.2. Más szoftverek						
4.3. Makrók, programozás						
4.4. Intelligencia						
5.1. Visszavonás						
5.2. Mentés						
5.3. Megerősítés-kérés						
5.4. „Azt kapsz, amit kapsz”						
5.5. Stabilitás						
6.1. Platform-függetlenség						
6.2. Változatfüggetlenség						
6.3. Hardverkörnyezet-függetlenség						
6.4. Nyelv-függetlenség						
Legfontosabb fogalmak: <i>adattípusok</i> (numerikus, szöveges, dátum stb.) <i>altáblák</i> mint műveletek operandusai, <i>képletek</i> (alap aritm., stat., „prog.-tétel” stb.) <i>szerkesztési funkciók</i> <i>adatbázis-kezelés</i> <i>grafikák</i> <i>nyomtatás</i> (fej- és oldallécek, altáblázatok stb.).						
Saját szempont:						
Saját szempont:						

Szövegszerkesztők

	Jegyzet- tömb	Word 2000/XP	Word 2007	Word- Perfect	T _E X	Writer
Mennyire „szimpatikus” globálisan?						
Mennyire ismerem?						
1.1. Egyszerűség – menük						
1.2. Egyszerűség – ikonok						
1.3. Egyszerűség – súgók						
1.4. Egyszerűség – uniformitás						
2. Vizualitás						
3. Teljesség						
4.1. Menük és súgók						
4.2. Más szoftverek						
4.3. Makrók, programozás						
4.4. Intellisence						
5.1. Visszavonás						
5.2. Mentés						
5.3. Megerősítéskérés						
5.4. „Azt kapsz, amit kapsz”						
5.5. Stabilitás						
6.1. Platform-függetlenség						
6.2. Változatfüggetlenség						
6.3. Hardverkörnyezet-függetlenség						
6.4. Nyelv-függetlenség						
Legfontosabb fogalmak: szövegegyeségek (jel, sor, paragrafus, lap stb.), szerkesztési funkciók, nyelvi szolgáltatások, grafikák beilleszthetősége, nyomtatás („előnyomtatás”=PREVIEW stb.), haladó „könyvszerkesztési funkciók”, haladóbb szövegegyeségek útban a DTP-felé, multimédia szövegegyeségek útban a WWW-felé.						

Adatbázis-kezelők

	Clipper	Fox Pro	Access	Oracle	Base	
Mennyire „szimpatikus” globálisan?						
Mennyire ismerem?						
1.1. Egyszerűség – menük						
1.2. Egyszerűség – ikonok						
1.3. Egyszerűség – súgók						
1.4. Egyszerűség – uniformitás						
2. Vizualitás						
3. Teljesség						
4.1. Menük és súgók						
4.2. Más szoftverek						
4.3. Makrók, programozás						
4.4. Intellisence						
5.1. Visszavonás						
5.2. Mentés						
5.3. Megerősítéskérés						
5.4. „Azt kapsz, amit kapsz”						
5.5. Stabilitás						
6.1. Platform-függetlenség						
6.2. Változatfüggetlenség						
6.3. Hardverkörnyezet-függetlenség						
6.4. Nyelv-függetlenség						
Legfontosabb fogalmak: <i>adattípusok</i> (numerikus, szöveges, dátum stb.) <i>adatbázis objektumok</i> (táblák, űrlapok, jelentések) <i>táblakapcsolatok</i> (1-1, 1-sok, sok-sok) <i>képletek</i> (mezőkben) <i>szerkesztési funkciók</i> <i>grafikák</i> <i>nyomtatás</i> (összefokozatos lista stb.).						
Saját szempont:						
Saját szempont:						

Ábraserkesztők

	Paint	Corel Draw	Photo-Shop	Logo-Motion	Gimp	Draw	Google Picasa	
Mennyire „szimpatikus” globálisan?								
Mennyire ismerem?								
1.1. Egyszerűség – menük								
1.2. Egyszerűség – ikonok								
1.3. Egyszerűség – súgók								
1.4. Egyszerűség – uniformitás								
2. Vizualitás								
3. Teljesség								
4.1. Menük és súgók								
4.2. Más szoftverek								
4.3. Makrók, programozás								
4.4. Intellisense								
5.1. Visszavonás								
5.2. Mentés								
5.3. Megerősítéskérés								
5.4. „Azt kapsz, amit kapsz”								
5.5. Stabilitás								
6.1. Platform-függetlenség								
6.2. Változatfüggetlenség								
6.3. Hardverkörnyezet-függetlenség								
6.4. Nyelv-függetlenség								
Legfontosabb fogalmak:								
<i>a priori</i> objektumok (pont, szakasz stb.),								
rajzeszközök (ecset, ceruza, ..., szöveg),								
objektumműveletek (mozgatás, ...),								
képméretezés, felbontás-megadás ...,								
kép-export, -import,								
haladóbb grafikai műveletek (előretolás ...),								
funkciók a 3D felé, animáció,								
nyomtatás (minőség, méret stb..)								
Saját szempont:.....								
Saját szempont:-----								

