SZOFTVEREK ÉRTÉKELÉSE
ISKOLAI SZEMPONTOK SZERINT

Szlávi Péter
szlavip@elte.hu

1999-2012

TARTALOM

ÁLTALÁNOS CÉLÚ SZOFTVEREK ÉRTÉKELÉSE ISKOLAI SZEMPONTOK SZERINT

A dolog érdekessége számunkra az, hogy

1. abban a ritka pillanatban, amikor szoftver választáskor döntéshelyzetben volnánk, akkor volna **szempontrendszer**, ami alapján összevethetnénk a választható szoftvereket;
2. a tematika összeállításkor ezen szempontok irányíthatják figyelmünket az adott szoftver „**gyenge pontjai**ra", amelyeket a normál hangsúlytól eltérően kell beillesztenünk az anyagba.

ÁLTALÁNOS MEGJEGYZÉSEK

Milyen szoftverekről lehet szó?

Az oktatást nyilván a (1) „**hétköznapok**", 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 cél**lal.

Oktatás elvárásai a szoftverekkel szemben

Alkalmazásához **ne** legyen szükség

1) mély, *speciális ismeretek*re (bár CAD-ismeret szükséges egy műszaki szki-ban, project manager – szervezési ismeretek szükségesek egy közgazdasági szki-ban, objektum-elvűséggel tisztában lenni egy informatikai szki-ból kikerülő számára hasznos),
2) *speciális, drága környezet*re (számítógép, periféria, operációs rendszer, szoftver stb., vö. winchester-/ memóriaméret-igény .Net, vagy Java futtatókörnyezet esetén, hálózati operációs rendszer [kiszolgáló oldali] Perl[1] vagy MySQL esetén); a „drágaság" nemcsak az egyszeri beruházásra vonatkozik, hanem a *fenntartásra* –lízingre, folytonos rendszergazdai teendőkre– is.

Alkalmasint szolgálhasson **útmutató**ul más, „rokon" szoftverek felé; kezdő eszközül „rokon" problémák megoldásához. Pl.:

1) szövegszerkesztő ⇒ kiadványszerkesztő, prezentáció-, honlapkészítés;
2) táblázatkezelő ⇒ adatbázis-kezelő, statisztikai rendszerek;

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

3) típusmegvalósítás embrionális foka (Logo, C, Pascal) ⇒ 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 puszta **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.

Észre kell vegyük, hogy ez a hozzáállás, tehát a „*feladatmegoldás algoritmikus alapon*” rendelkezik egyfajta „**univerzalitás**sal”, 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.doc/htm.)

1.3. Egy nyelvtípus megismerése

Egy részről fontos cél a különféle **programozási nyelvosztályok modellezése.** Pl.:

- a funkcionális nyelvekhez ⇒ Logo függvényes része (esetleg a Forth);
- a logikai nyelvekhez ⇒ Prolog;
- az automata-elvű nyelvekhez ⇒ a Logo vagy a Python[2] teknőcgrafikája, a COMAL[3], a Turbo/Free Pascal (Graph3 unit felhasználásával);
- tárgyközéppontúsághoz ⇒ Imagine, Scratch, Python, Java…;
- ...

Más oldalról hasznos lehet az **alkalmazói rendszerek**hez kapcsolódó nyelvek **modellezése***;* mint például a Quattro/Excel vagy a WinWord makrónyelvéből kinőtt WordBASIC, majd Visual BASIC, de gondolhatunk a TEX 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.

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. 6 **kulcs-szó** van csupán: LET, IF, GOTO, INPUT, PRINT, DIM.

Elborzasztó szintaktikai példák: BASIC – változó-„utókák": v%, v$, v!, v#, v&, mint **implicit** típusjelölők; Forth – **.**, **!**, **@**, ...; APL – **[**, **#**, ..., vagy Perl **$**xxx, **@**xxx, **&**xxx, **%**xxx, **@[**, **@_**, mint **implicit** struktúrajelölők*; C++ – cin >>, cout << szokatlan elnevezése és bontása a tevékenységének. (L. a mellékelt ábrákat)

```
:EVEKT0
 <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")

```
(~R∈R∘.×R)/R←1↓⍳R
```

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

[2] A nyelvleírás: http://www.python.hu/.
[3] A nyelvleírás: http://en.wikipedia.org/wiki/COMAL,
és egy jellegzetes példaprogram: http://www.josvisser.nl/opencomal/whitepaper1.txt

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 *túlhangsúlyozás*a 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**, és közismertségnek örvendenek. Természetesen az ottanival azonos értelemben használható, így világos a kifejeznivalója, és rövid.

Az alapszavak **programba illeszkedés**éről: **kiemelendők-e**, pl. nagybetűkkel (ELAN, COMAL, MODULA ...), vagy máshogyan (kisbetűkkel, mint a C-szerű nyelvekben, aposztróffal, mint az ALGOL 60)? Általában a **kis- és nagybetűk** megkülönböztetésének kényszere is fontos (stílust meghatározó) *módszertani kérdés*.

Kijjebb-beljebb kezdés: Python szükségtelenné teszi a struktúrazáró kulcs-szavakat, mivel egyértelmű a blokk-struktúra a kijjebb-beljebb kezdés miatt.

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. Hasonlóan igen sunyi hibákat okozhat bizonyos **alapszavak hiánya**, mint pl. a típusokat jelölők hiánya. Tipikus hiba: a hibaüzenetet sem okozó típuskeveredés az explicit **típusdeklaráció nélküli** nyelvekben, amelyekben dinamikusan és automatikusan definiálódik egy-egy adat típusa. (Lásd Perl, JavaScript.)

```
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); #eljáráshívás két tömbbel
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 A B C b=
```

Ugyanez apró különbségekkel (*cím*hivatkozással):

```
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); #eljáráshívás 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
```

Két Perl programrészlet

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

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

♣ Emlékeztetőül: $xxx – skalárváltozó; @xxx – tömbváltozó; %xxx – hash-táblázat…

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

*zőség, precízség ⇔ Program**írás**i 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 az első használatkor'* szokása. Pl. for (int i=0; i<n; ++i) {…}. Hajlok rá, hogy hasznosnak minősítsem, mivel nyomatékosítja (sőt garantálja is!), hogy csak a ciklusban használható!

A **szigorúság-kifinomultság** ára: lásd az Eiffel-t, az Adát vagy a korábbi nyelvek közül a PL/I-t, amelyek túlon-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. (A „hatás” és az „érték” ilyen szétválasztása komoly veszélyforrás!) Tehát ismét felvethetjük a '*tömörség vagy világosság*' kérdését.

2.1.3. Következetes programszerkezet, következetesség

Következetesség = kevés szabály (kevés kivétel) ⇒ kitalálhatóság.

Könnyen megjegyezhetők pl. az **elválasztó jelek**. Ellenpélda: Pascal *vessző, pontosvessző* használata a paraméterezésben (aktuális, ill. formális paraméterek), vagy mikor kell vessző, mikor lehet, mikor nem szabad (lásd IF-THEN-ELSE ⇒ ELSE előtt szintaktikus hiba, vagy FOR / WHILE … DO ⇒ DO után hibajelzés nélküli, igen sunyi szemantikus hiba).

```
Var x: …; y: …;
```

de

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

Két Pascal programtöredék

A szintaxisban érthetetlen „**környezetfüggőségek**” tapasztalhatók megint 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 „szokásostól” eltérő feltétel-értelmezése okozta bizonytalanság. De a C++ is szolgál rossz példával: gondoljunk a *típusdefiniálás*ra a struktúra, illetve pl. a tömbtípus definiálásának *eltérő szintaxis*ára (**struct** tipusnév{tip1 mező1; tip2 mező2; …} ⇔ **typedef** tip tipusnév[méret]).

Összetett szerkezetek eleje-vége jelzésének a kérdése. Pl. az 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. Nem vitás, az egy utasításos esetben alkalmazható egyszerűsítés (bármely nyelvben) igen veszélyes – ahogy az elválasztójelekkel kapcsolatban már szóba került! És egy jó példa: ELAN-ban **IF**-END**IF**, **REP**-END**REP**... Maple-ben: if-fi, do-od…

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

Másfajta következetesség jó példájaként dicsérhetjük a Perl értékadás-szerű operátorainak családját: „+=”, „–=”, „*=”, „/=”, „**=”, „%=”, „&=”, „|=”, … . (Ez hasonlóan meg van számos C-szerű nyelvnél.)

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 értéktípusára tett korlátozások miatti eljárásra történő kényszerű áttérésre Pascal esetén (bár a Free Pascalban már megengedett a rekordértékű függvény), vagy egy másik tipikus „konverziós” kényszerre: a többirányú elágazás ⇒ If-ekké; s még egyre: a ciklus kilépési feltétel negálására…

Jó példa a Perl „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 operátorszintaxisa:’**:=**’ (értékadás) helyett ’=’; ’=’ (azonosság) helyett ’==’. Különösen a korábban (2.1.2.) említett ’*értékadás a kifejezésben*’ szintaktikai megfelelés mellett!

2.2.2. Jó modellje nyelvosztályának (⇒ más nyelv)

Következetesen, érzékletesen tartalmazza azon jellemzőket, amik **lényegesek az osztálya szempontjából**.[4] A Logo „eklektikusságát” ebből a szempontból nem tartom ideálisnak: nem mutatja meg világosan sem az *automata*, sem a *funkcionális* nyelvek jellemzőit.

Legyen jó alap a **továbblépés**hez. 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 stílus**t is alaposan befolyásolja, ezért ez a szempont különösen fontos szerepet játszik a nyelvválasztásban.

[4] Ezt minden egyes nyelvosztályra érdemes lenne külön meggondolni.

A legfontosabb „stílusjegyek”:

a) **felülről lefelé programtervezés**, mint gondolkozást meghatározó stratégia – finomítások, paraméterek (vö. Pascal, ELAN, Perl),

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) **mentes az „illegális” és veszélyes lehetőségektől** (pl. a Logo-ban nem lehet(ne) értékadás; az algoritmikus nyelvekben GOTO, STOP, HALT, EXIT tilos, helyettük kifejezett *kivételkezelés* ajánlott; e lehetőség nélkül sajátos *kódolási szabályok* szükségeltetnek[5]),

e) **mentes 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 max. 2-jelű ⇔ ELAN szóközöket is tartalmazó „mondatnyi”; Perl „változó-előkéi” [$, @, %...]),

g) **modularitás** lehetősége pl. a típusmegvalósításhoz (pl. a Pascal include-ja, unit-ja, a Modula és a COMAL modulja, az Ada package-e, az ELAN pack-ja…)

Megjegyzés: a Pascal hiányzó *generic* fogalma részben pótolható az ellenőrzött *include*-dal, mindenestre szóba hozható pl. az Ada, C++ irányába mutató hiányként.

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

2.3.2. A használat közben – rendelkezik minimális kódhatékonysággal, nyitottsággal

Tartsa meg a „**téma-univerzalitás**át”! 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!

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

A **konkrét** gép/operációs rendszer **adottságaihoz** (minimális mértékig) **alkalmazkodjon** a nyitottsága által: (ko)processzor, memóriamodellek, grafikai lehetőségek (pl. JavaScript), egérkezeléshez rutinok...[6] Mindazon által ne függjön tőlük meghatározóan, kizárólagosan.

2.4. Fejlesztői környezet léte

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

Eddig a nyelvről *általában* esett szó. Ettől kezdve a **megvalósítás** mikéntje a legfontosabb kérdés, mellesleg tovább finomítjuk a nyelvi „jóság” szempontjait.

2.4.1. Integrált nyelvi környezet

Ellenpéldák a korai professzionális nyelvek: programszerkesztés + fordítás + futtatatás + programjavítás papíron + ugyanezek újra. (Bár bizonyos nyelvi környezetek ma[7] is ilyenek. Lásd UNIX-os Adát; vagy a szkript nyelveket, a böngészőben futó kódok nyelvi környezeteit általában.) Ezen némileg segítenek az „univerzális” (szintaxist ismerő) **programszerkesztők** az ún. *kódkiemelő* (code highlighting) szolgáltatásukkal. Pl. BSEdit, Note++…

Komplettebb **fejlesztői környezetek** is vannak: Geany – Pascal, PHP, Java, Python…; NetBeans – Java, PHP, C++, C#…; Code::Blocks – C, C++, MATHLAB;… Hasznos szolgáltatás az ún. *kódkiegészítés* (code completion), amely gyorsabbá és biztonságosabbá teheti a kódolást. Erre általában képesek az igényesebb fejlesztői környezetek.

2.4.2. Párbeszédesség a programfuttatásban

Interaktív futtatási lehetőségek: nyomkövethetőség (előre és vissza), ..., adatok menetközbeni lekérdezhetősége, módosíthatósága. Ez a programmal való „együttlélegzés” záloga; s ilyenformán a programozási kedvet, a program iránti „empátiát”, a fejlesztés hatékonyágának a növekedését szolgálja, így **oktatási** súllyal bír.

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

2.4.3. Súgók

Az **oktathatóság** szempontjából különösen nagy jelentőséggel bír a súgók léte, milyensége. Nagy nehézséget jelent, ha a súgás kimerül egy internetcímre való utalásban. (Pl. így volt a FreePascal, és a Lazarus „piacrakerülés” korai szakaszában.)

A *menűvezérelt* kívánja meg a legtöbb „humán beruházást”: hosszas keresést, rossz esetben nyelvi nehézségeket okoz (idegen nyelvű, félig lefordított, érthetetlenül fordított szövegekkel) → garantált kudarcélmény.

[6] Persze, ha ezek nincsenek a lényeggel ellentmondásban.
[7] Az ezredforduló táján.

Speciális billentyű vezérelt menük sokszor rendelkeznek a *helyzetérzékenység*gel, azaz olajozzák a problémamegoldást.

Nagyra becsülendő az a keretrendszer, amely rendelkezik *automatikus szintaxist mutató buborék*okkal (hint).

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*... ⇔ OEM[8]?)

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 *súgó 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 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 gép, annyi BASIC-**nyelvjárás** született, a számos szabványosítási kísérletek ellenére.

Érdekes a Pascal nyelv esetére gondolni. A Borlandék Turbo Pascalja a 1990-es évek elejéig nagy népszerűségnek örvendett. A Borland ódivatúnak minősítvén leállt a fejlesztésről, a keletke-

[8] 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.mspx és http://www.microsoft.com/hun/oktatas/oem.mspx.)

zett Pascal-űrt lassan a Free Pascal töltötte be. A Free Pascal sikere alighanem a **nyílt forráskódúság**nak köszönhető. Megjegyzem: a szárnybontogatás idején igen nagy kockázattal és sok nehézséggel járt a használata.

Egy nyelv körül kialakult **webes fejlesztői közönség** meghatározó lehet egy nyelv jövője szempontjából. (L. Python.)

A **böngészőben futó nyelvek** (szkript nyelvek) is könnyen terjedhetnek, hiszen „kéznél" van a futtató környezet; igaz, a fejlesztői környezet hiánya visszavesz a terjedés lendületéből.

2.6. Fejlesztői környezet finomabb részletei

A nyelv birtokba vételének ára... „Anyám, nem ilyen lovat akartam" 1.

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*. Alkalmasint a lexikális elem első néhány jele alapján felkínálja az odaillleszthető 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 bevitelt[9]… 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 igyekszik. Pl. a Turbo Pascal az objektum fogalmat teljesen alkalmatlan módon támogatja: minden kapcsolódó fogalmat (pl. metódus, property, „objektumhierarchia"…) kulcs-szószinten kell ismerni, semmi segítséget nem nyújt az ismerkedő számára. Ablakközéppontú programozáshoz elengedhetetlen a GUI-s, eszközpalettás, … fejlesztői környezet. (4GL)

2.6.2. Beépített futtató-/nyomkövető-rendszer

A Geany nem kínál nyomkövetési szolgáltatásokat, a Code::Blocks kínál, de hektikusan, NetBeans GUI[10] és a Turbo UI példásan jár a programozó kedvében.

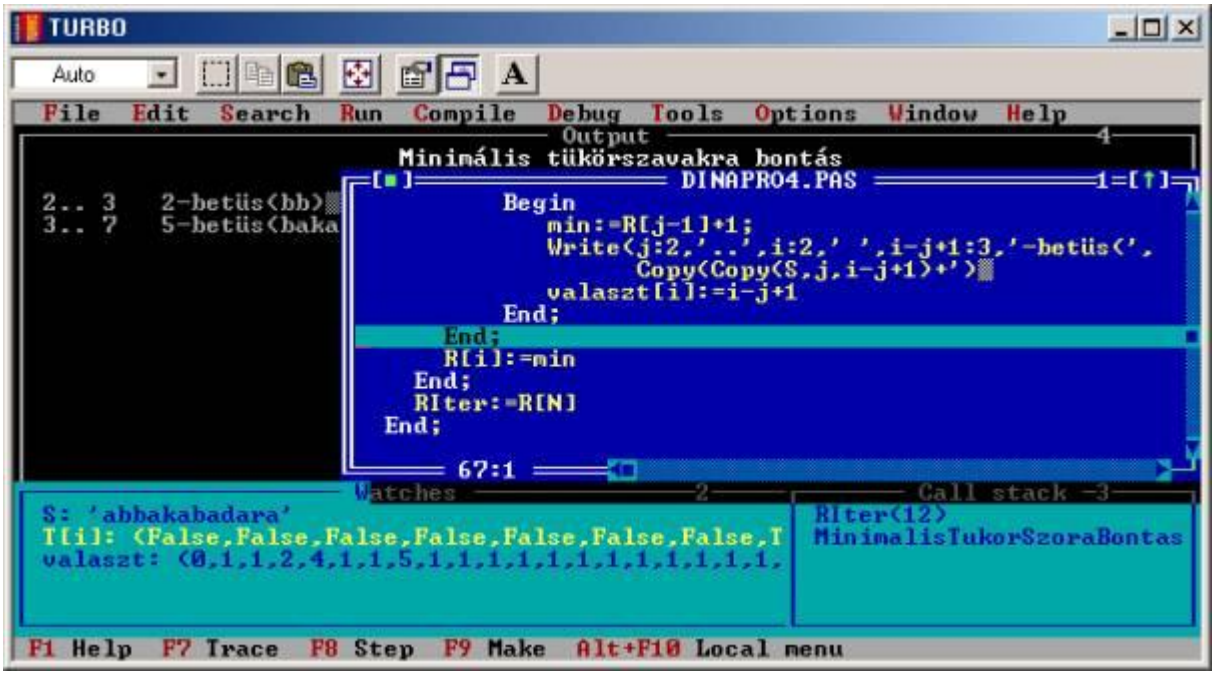
A nyomkövetés hiányára némi gyógyírt jelenthet a **feltételes fordítás**.

---

[9] Lásd http://people.inf.elte.hu/szlavi/InfoOkt/SzoftErt/LCNLogoPl.htm

[10] http://www.cs.uga.edu/~shoulami/sp2009/cs1301/tutorial/NetBeansDebuggerTutorial/NetBeansDebuggerTutorial.htm

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



2.6.3. Fordítási lehetőségek

Biztonságos vagy hatékony kód generálása közötti **választást** „lokálisan” (egy-egy kóddarabra fókuszálva) fölkínálja-e (opciók)? Automatikus és „kézi” **hibafelderítő mechanizmus**ok (indextúllépés, típussértés) legyenek beépíthetők, ill. kikapcsolhatók. (V.ö.: Turbo/Free Pascal, Code::Blocks[11] környezet fordítási opciókat és feltételes fordítási lehetőségeket.) Ez által ötvözhető a fejlesztői környezet programkészítést támogató funkciók és a végrehajtás hatékonysága (gyorsaság, kódtömörség).

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 [12]). Egy 4GL-környezet szinte kínálja ezt a modulokra épülő, „keretprogramos” oktatási stratégiát, amelyben az alkalmazói felületet kezelő modult készen adhatjuk a tanulóknak, s nekik „csak” a lényeget jelentő modul elkészítése a feladat.

2.6.4. Többplatformúság

A több operációs rendszeren futás képessége előnyös lehet, hiszen támogatja az esetleges operációs rendszer-változtatást. De figyelem! A végrehajtásnak **szemantikusan ekvivalens**nek kell lennie. Sajnos messze nem igaz ez –ma még– a böngészőben futó programokra. (Pl. JavaScript.)

---

[11] Érdemes a debug/release különbséget egy aprócska program segítségével érzékeltetni: http://people.inf.elte.hu/szlavi/InfoOkt/SzoftErt/Cpp_proba.zip

[12] Lásd http://people.inf.elte.hu/szlavi/InfoOkt/SzoftErt/ModularisPascalPl.htm

A *feltételes fordítás* lehetősége némi esélyt adhat a készítendő program többplatformuságának megőrzésére.

2.7. Stabilitás

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”**. Ilyen effektusok forrása igen sokrétű lehet. Van, amelyek nyelvi jellegűek, de számos a megvalósítással (akár a fordítóprogrammal, akár a fejlesztői környezettel) függ össze.

Egy nyelvvel összefüggő példa: a **kezdőérték**ekkel kapcsolatos. A deklarációban (a változó létrejöttekor) 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. a weben egy publikus JavaScript mintaprogramban találtam teljességgel fölösleges és szintaktikusan hibás sort is, amellyel (és amely nélkül is) helyesen fut a program.

Fordítási stabilitás alatt azt értem, hogy a fordítás sikeressége nem függ a fordító verziószámától. Rossz példa a Code::Blocks, amelynél tapasztalható, hogy a rendszerkönyvtár fájljainak tartalma változik (cserélődik), így egy korábban szintaktikusan helyes programot az újabb fordító hibásnak jelez. (Az ok prózai: bizonyos nyelvi elemek egy olyan header-állományba kerültek, amely korábban nem volt inklúdálandó.)

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 példaporgram másként működik Explorerben, mint FireFox böngészőben!) Sajnos nem csak a szkript nyelvek szolgálnak „jó” példával. Ui. a Free Pascal (≤v1.0.10) végtelen ciklusba esik textfájl-beolvasás közben, ha az utolsó sor végén nincs sorvégjel, míg a Turbo Pascal helyesen lefut.

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).

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

3.1. Free Pascal – Fejlesztői környezet: …… (pl. IDE, Geany, Lazarus, …)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.2. PHP – Fejlesztői környezet: …… (pl. ConTEXT, PhpED, …)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
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. 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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.4. JavaScript – Fejlesztői környezet: …… (pl.jQuery, …)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.5. Perl – Fejlesztői környezet: …… (pl. jEdit, Eclipse+EPIC, …)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.6. C++ – Fejlesztői környezet: …… (pl. Code::Blocks, QtCreator,…)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.7. C# – Fejlesztői környezet: …… (pl. Visual Studio,…)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.8. Visual C++ – Fejlesztői környezet: …… (pl. Visual Studio, …)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
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. 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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
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. 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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
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. 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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.12. Python – Fejlesztői környezet: …… (pl. IDLE, Eclipse, …)

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:										

3.13. Java – Fejlesztői környezet: …… (pl. JDK, Eclipse, …)

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
Saját szempont:										
Saját szempont:										
Saját szempont:										

3.14. Turbo Prolog *(a logikai nyelvre óvatosan kell alkalmazni a szempontokat)*

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										
2.6.4. Többplatformúság										
2.7. Stabilitás										
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 „**kategóriá**t” is megismerje: funkciók, típusfeladatok és megoldási lépések…

1.3. „Profi” alkalmazó

Munkakörök egy része (titkárnő, adminisztratív munkakörök stb.) **profi**bb **alkalmazói ismeretek**et 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/Özvegy 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, l. módszertani szempontból lényeges más vonatkozását is: 2.4.1.). Az ikonok **méretnövelhetősége** gyengénlátók iskolájában való alkalmazhatóság feltétele.

2.1.3. Súgó

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

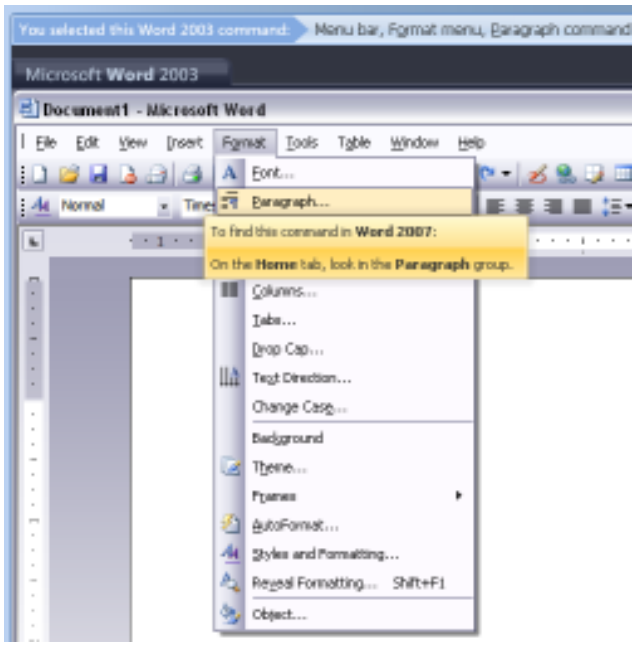
hasznavehetetlen a rosszul, **magyartalanul** („félmagyarul”) megfogalmazott súgó. Probléma, ha csak **idegen nyelv**en hozzáférhető. (A többnyelvűségnek van kihasználható oktatási jelentősége!) Szerencsés, ha jellegzetes **példák**at is tartalmaz. Jó példa az Excel-beli függvények némelyikéhez tartozó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

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

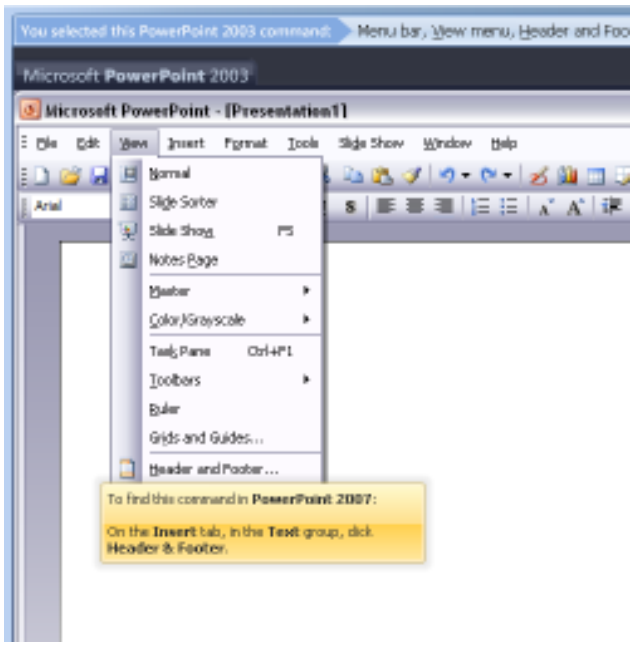
2.1.4. Uniformitás

„**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 megnehezítené 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 uniformítás 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 T_EX dokumentumszerkesztő igen **átgondolt lehetőségei** (matematikus akkurátusság tükröződik benne) és **igénytelen** „maj’meglátod’a’nyomtatáSután” ígéretű **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 ***animáció***s lehetőségekkel segítése, dinamizálása.

2.3. Teljesség

Jól „**modellezze**” azt az alkalmazói programcsaládot, amelynek tagja. Ez vonatkozik a szoftver „*filozófiá*jára”, a *funkció*ra és *fogalma*kra egyaránt. L. 3. fejezetet.

2.4. Rugalmasság, „testre szabhatóság”

Különösen fontos, hogy az **oktatás** adott fázisában a pillanatnyi célhoz, célközönséghez „idomítani” lehessen. A **hétköznapi alkalmazás**ban 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)ság***a* az oktatásban elemi elvárás.

A **menük szűkíthetősége**, **átcsoportosíthatóság**a 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/Jegyzettömb-Excel-Word-Paint...–, amelyben a *vágólap*on 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 bevezetésé**re, 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 **árnyoldalai**t sem! Hogy a leggyakoribb bosszantó automatizmust említsem: egy számozott felsorolás száma után kérés nélküli, szándék ellenére történő nagykezdőbetűre átváltás. Ennek a jó esetben egyszerűsítő szolgáltatásnak akkor van **használhatóságot növelő** szerepe, ha a használónak 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** *lehetősé*ge (UNDO). Jó példák az Office 97-beliek (meg a „leszármazottak”). Egyszerűbb szövegszerkesztők esetén (pl. Jegyzettömb) 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 *támogatás*t nyújt pl. a fájlnév-szerkezettel („vezeték-” és „keresztnév”, azaz kiterjesztés, sőt verziószám). (Lásd VMS verziószámos, 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 szerkeszthetetlenségére.

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 újraindításkor (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 platform**on is létezik változata. Pl. az OpenOffice.org/LibreOffice 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 változat*ban készített dokumentumot az aktuális verziójú szoftver *nem vagy rosszul tölti be*. Vö. uniformitással (2.1.4.).

Másik probléma: *„extrémebb” jelek* megváltozása, kiesése verzióváltáskor vagy formátumcsere 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 szöveg*tördelés, 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 **nyelv**én „szóljanak”. Érdemes elgondolkodni persze a „**többnyelvűség**” egyéb célú kihasználhatóságán. Az OpenOffice.org e szempontból is kiváló, hiszen 2008 tájékán 20 nyelven érhető el. (A LibreOffice 2012 tavaszán még „csak” 5 nyelvű. [libreoffice.hu])

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 (Jegyzettömb/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, körülfuttatás 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. Ábraszerkesztő 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* (mozgatás, színezés, forgatás, nyújtás, mintázás),
- *képméretezés, felbontás-megadás, színmélység-módosítás*,
- *kép-export, -import* (fájlfajták, esetleg scanner),
- *haladóbb grafikai műveletek* (előre-, hátratolás, 2 alakzat „fokozatos közelítése”, „effektusok”, színmodellek),
- *funkciók a 3D felé,* az *animáció felé*,
- *nyomtatás* (különféle minőségben és méretben).

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).

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. Intellisence						
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: *adattípusok* (numerikus, szöveges, dátum stb.) *altáblák* mint műveletek operandusai, *képletek* (alap aritm., stat., „prog.-tétel” stb.) *szerkesztési funkciók* *adatbázis-kezelés* *grafikák* *nyomtatás* (fej- és oldallécek, altáblázatok stb.).	.....	.....	.....	.....	.....	.....
Saját szempont:.....						
Saját szempont:.....						

Szövegszerkesztők

	Jegyzet-tömb	Word 2000/XP	Word 2007	Word-Perfect	T_EX	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: *szövegegységek* (jel, sor, paragrafus, lap stb.), *szerkesztési funkciók*, *nyelvi szolgáltatások*, *grafikák beilleszthetősége*, *nyomtatás* („előnyomtatás”=PREVIEW stb.), *haladó „könyvszerkesztési funkciók*”, *haladóbb szövegegységek* útban a DTP-felé, *multimédia szövegegységek* 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: *adattípusok* (numerikus, szöveges, dátum stb.) *adatbázis* objektumok (táblák, űrlapok, jelentések) *táblakapcsolatok* (1-1, 1-sok, sok-sok) *képletek* (mezőkben) *szerkesztési funkciók* *grafikák* *nyomtatás* (összegfokozatos lista stb.).	.....	.....	.....	.....	.....	.....
Saját szempont:.....						
Saját szempont:.....						

Ábraszerkesztők

	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. 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: *a priori 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:----------								