**Szakdolgozat**

# Középiskolai programozás oktatás vizuális környezetben

*Törley Gábor*

**Témavezető:** ***Szlávi Péter***

**ELTE IK**
**Informatika Szakmódszertani Csoport**
**2005.**

# *Tartalom*

# Kivonat

A középiskolában, az informatikaoktatáson belül kevés idő jut a programozásra, ráadásul nem is áll első helyen a diákoknál sem. Sőt, az érettségi követelmények csökkentésével, a közép szintű informatika-érettségi nem is kéri számon a diák programozástudását.

Szakdolgozatomban bemutatom a középiskolai programozás célját, és egy új módszer előképét villantom meg.

Az első részben a jelent mutatom be. Arról értekezem, hogy miért használható hatékonyan az ún. *módszeres programozás*, a módszereken túl kitérek arra is, hogy miért jó választás a Pascal, mint első magasszintű nyelv. Az első rész központi részében arról írok, hogy a módszeres programozásra és a Pascal nyelvre alapozva, milyen reményeink lehetnek a programozás-oktatás terén, ha az eddigi „megszokott" DOS-os Turbo Pascal környezetről Delphire váltanánk. A két nyelv tulajdonságait hasonlítom össze, a módszeres programozásnál megismert *Bemenet – Feldolgozás – Kimenet* funkcióhármasának a szemszögéből. Külön kitérek a Delphi számunkra fontos nyelvi jellemzőire és új szolgáltatásaira, különös figyelemmel az eseménykezelésre és a hibakezelésre. Az első rész végén a programozásoktatás célját és mai helyzetét mutatom be a hatályos tanterveken keresztül.

A dolgozat második része egy pszichológiai vizsgálat és annak kiértékelése. A vizsgálat célja, hogy mérhetővé tegye, számszerűsítse a diákok hozzáállását, attitűdjét a programozáshoz, illetve felmérje tanulási „szokásaikat" és az iskolai-otthoni környezet számunkra lényeges jellemzőit. Valamint összefüggéseket keressen a programozás eredményességét – vélhetően – meghatározó tényezők között. Választ keres arra továbbá, hogy milyen változtatásokkal – a módszerekre és egyéb, tanítási körülményekre vonatkozóan – lehetne a jelenlegi helyzeten gyökeresen javítani.

Szakdolgozatom befejező részében egy leendő tankönyv első fejezetei látnak napvilágot. A könyv tanítási módszere ötvözi az első részben bemutatott módszeres programozást és a Delphi rendszerének tulajdonságait. Nyelvezetében fiatalos, azzal a nem titkolt céllal, hogy így próbálja még érthetőbben megtanítani, és a diákhoz közelibbé vinni a programozást.

# 1 Bevezetés

Rohanó világunkban egyre nagyobb teret hódít az informatika. Ma már megengedhetetlen, hogy valaki számítógépes ismeret nélkül kerüljön ki az Életbe. Így a közoktatásban is, az új igényekhez alkalmazkodva, nagyobb szerepet kapott az informatika oktatása, mint akárcsak 5-10 évvel ezelőtt.

A tantárgyat nem lehet ízekre szedni, és nem mondhatjuk azt, hogy „csak az alkalmazói ismeretek kellenek" vagy „csak programozni tanuljon meg a gyerek". Szükséges, hogy a tanulók naprakész, és használható tudást szerezzenek, továbbtanulási igényeiknek megfelelően, ha az utóbbi feltétel megoldható.

Szakdolgozatomban a programozási oldalt szeretném górcső alá venni. Meg kívánom mutatni, hogy a programozás sokkal több mindent rejt magában, mint maga a kódolás, hogy nem azért „kell" a diákoknak evvel foglalkozni, hogy megnehezítsük az életüket, hanem mert olyan gondolkodásmódot tudnak a programozás alaplépéseivel elsajátítani, amire szükségük lesz mindennapi életük során. Ezt a gondolkodásmódot nevezik *algoritmikus gondolkodás*nak. A hangsúlyt ebben az írásban nem erre szeretném fektetni, bár, sokszor elő fog jönni, hanem arra, hogy olyan lehetőségeket mutassak be, amelyek által élvezhetőbbé és hatékonyabbá lehet tenni a programozás tanítást/tanulást. Fontos szempont ez! Gyakorlótanításom folyamán tapasztaltam, hogy még egy informatikai szakközépiskolában is az egyik legnehezebb része a számítástechnika tárgyaknak a programozás, főleg azok számára, akik a későbbiekben nem evvel szeretnének foglalkozni.

## 1.1 Miért pont a Pascal legyen az oktatás nyelve?

### *Programozás – algoritmizálás*

Sok gimnáziumban, szakközépiskolában, és a felsőoktatásban is, első magasszintű nyelvi környezetként a Borland által fejlesztett Turbo Pascalt használják.

Még mielőtt rátérnék a miértekre és hogyanokra, távolabbról indítom gondolataimat. Azt a kérdést szeretném először megvizsgálni, hogy mi a célja egy programozási nyelv tanításának. Két fajta szélsőségbe eshetünk: eltúlozhatjuk az algoritmikus gondolkodás szerepét, s így abba a tévedésbe eshetünk, hogy a programozás *absztrakt* tevékenység, így nyelvfüggetlenül végezhető ennek tanítása. A másik téveszme, ami talán „tévesebb" az előzőnél, a *nyelv* szerepét hangsúlyozza túl. E szerint informatika = programozás = programozási nyelv.[1] **[SzZs]** Ezért alakulhattak ki ilyen beszélgetések: – *„Te mit tanulsz*

[1] Szlávi Péter: Szoftverek értékelése iskolai szempontok szerint 2. o. – http://digo.inf.elte.hu/~szlavi/InfoOkt/SzoftErt.html

*programozásból?" – „Pascalt"*[2] Valahol a fenti két téveszme között kell meglelnünk az arany középutat.

Többféle módszer szerint oktatták, oktatják a programozást, vannak jó és nem annyira jó módszerek. Dolgozatom nem hivatott ezek között különbséget tenni, most azt a módszert részletezem, amely – véleményem szerint – a leghatékonyabban használható a gimnáziumi programozás oktatás terén. Ezt hívják módszeres, *algoritmus-orientált* módszernek.

Ez a tanítási metódus is – mint néhány másik – a programírás *teljes* folyamatát tekinti át[3]:

- feladat-meghatározás, specifikáció;
- algoritmus- és adatstruktúra-tervezés, az algoritmus helyességének belátása;
- kódolás;
- tesztelés;
- hibakeresés, hibajavítás;
- hatékonyság- és minőségvizsgálat;
- dokumentálás.

Az egyes résztevékenységek között az algoritmus előállítása az elsődleges feladat. Erre helyeződik a legnagyobb hangsúly a tanítás folyamán, s a többi részben is megjelennek az algoritmus-orientált elemek.

Az algoritmus-előállítás alapja a szisztematikus felépítés. Első lépése az általános feladattípusok, s azok általános megvalósítási sémái, az úgynevezett programozási tételek felismerése.

Második lépésként a konkrét feladatokat szükséges visszavezetnünk valamely tételre vagy tételekre. Így megállapítható, hogyan alkalmazhatóak ezek a tételek. Ebben rejlik a módszer különlegessége. Ugyanis párhuzamosan vizsgálja a specifikációt és az algoritmust, mindkettőben meggondolva, hol és mit kell aktualizálni.

Harmadik lépésként foglalkozhatunk a tételek összeépítésével. Azaz olyan feladatokat oldunk meg, ahol egymással szoros „együttműködésben" kell használni több tételt a megoldás folyamán.

Az adatszerkezet tervezése itt a programozási módszertan szempontjai szerint kerül feldolgozásra. Tehát az adatstruktúra lényegében egy típus, specifikálással, a struktúra reprezentálásával és a műveletek implementálásával.

A kódolás során sok, konkrét vagy általánosabb, programozási nyelvtől függő döntés születhet. Ez a módszer a döntések többségét szintén algoritmikus szemlélettel fogja meg. Például ilyen módon tárgyalja a rekurzív és iteratív algoritmusok közötti átalakítás módszerét, illetve a szokásos, programozás során előforduló vezérlés-szerkezetek (elágazások,

[2] Szlávi Péter – Zsakó László: Programozás tanítási módszerek 5. o. – http://digo.inf.elte.hu/~szlavi/ProgModsz/SzlaviZsako.pdf
[3] Szlávi Péter – Zsakó László: Programozás tanítási módszerek 2-3. o.

ciklusok, eljárások, függvények, operátorok…) egymással történő megvalósítását. A módszer ezen kívül ebben a fázisban foglalkozik avval, hogyan lehet a programot minél inkább a felhasználó „barátjává" tenni (pl.: menük, ablakok megvalósítása). A tesztelési módszereken belül szintén foglalkozik a specifikációra épített „fekete doboz", és az algoritmusra épített „fehér doboz" módszerekkel.

A hatékonyságvizsgálatot is az algoritmusra alapozva tárgyalja ez a módszer. Ez azt jelenti, hogy a végrehajtási idő, a helyfoglalás és a bonyolultság szempontjából általános feladatosztályokat fogalmaz meg (hasonlóan a programozási tételekhez), s algoritmikus sémákat, ötleteket ad, a hatékonyabbá írás érdekében (pl. a sorozat részekre osztása alkalmazható a logaritmikus keresés, a quicksort rendezés, a párhuzamos minimum- és maximum kiválasztás algoritmusánál).

Mivel itt a programozási nyelv nem játszik elsődleges szerepet, így a programozási ismeretek felépítését nem befolyásolja túlzottan, így az evvel a módszerrel tanuló programozó nem lesz nyelvhez kötött.

Ha itt megállunk, akkor nem értünk célt. Szükségünk van arra, hogy az elkészült algoritmust kipróbáljuk a gyakorlatban. Ehhez szükségünk lesz egy programozási környezetre/nyelvre.

Milyen szempontok alapján válasszunk nyelvet? Jelen esetben az lenne a fontos, hogy a nyelvet könnyen lehessen tanítani egy kezdő programozó számára. Tehát, magát a nyelvet fogjuk vizsgálni, s nem valamelyik implementációját.[4]

Tehát kezdőknek szeretnénk tanítani a programozást, ezért fontos, hogy az első benyomások jók legyenek, ezért fontos azt megvizsgálni, hogy milyen könnyű megírni az első programot.

## *A nyelv megítélésének szempontjai*

Nem mellékes, hogy mennyire értelmesek a programnyelv *alapszavai*. A Forth nyelv ilyen szempontból „kiváló" ellenpélda.

```
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***

Sokan szokták a C nyelvet ajánlani, mert rövidíti a begépelést, világosak az alapszavak, az egyes programrészek jól elkülöníthetőek egymástól ({ és }), de kezdőként nehéz mit

[4] Szlávi Péter: Szoftverek értékelése iskolai szempontok szerint 2. o.

kezdeni a következőkkel: '+i', 'i+', '++i' vagy 'i++'. Egyes nyelvek kiemelik alapszavaikat (Elan, Modula).

Érdemes megfontolni, mikor, mi a jobb? Világosan kifejezni az alapszavakat, vagy „ahogy tetszik" írni őket. A programírás egyszerűsége nagyban függ a programozási stílustól, azaz a kifejezőkészségtől és a precízségtől.

Sokat segít egy kezdő programozónak, ha könnyen át tudja látni és meg tudja jegyezni a programszerkezetet. A BASIC ilyen szempontból „túl sokat" enged, „túl egyszerű": pl. változók deklarációja elhagyható vagy szabadon elhelyezhető. Nem kérdéses, hogy hasznos-e ez az engedékenység!

Érdemes megvizsgálni a programszerkezet következetességét: pl. könnyen megjegyezhetőek-e az elválasztójelek? A Pascal bizonyos vonásai jól példázzák ezt. Ő ui. nem következetesen várja el a vesszőket, ill. pontos vesszőket. Nehéz egy kezdőnek megérteni, s így megjegyezni: mikor kell vesszőt és pontosvesszőt használni a paramétereknél (formális, ill. aktuális paramétereknél), vagy mikor kell pontosvessző az utasítás végére, s mikor nem (If-Then-Else).

Összetett programszerkezetek eleje-vége jelzése jó, ha „beszédes". Hasznos segítség a programszerkezet „feltérképezésében", a programműködés megértésében. Jó példa: ELAN (If-Endif, Rep-Endrep). A Pascal e tekintetben sem kifogástalan. Sokféle utasítászárójelet-párt használ: Begin-End, de Record-End, Case-End, Repeat-Until.

Fontos szempont, hogy a nyelv ne térjen el lényegesen az algoritmikus nyelvtől. A kódolás érdekében ne kelljen lényegesen átalakítani programunkat. Így egyszerűen lehet kódolni, és tanulni a nyelvet.

A fentebbi felsorolásból sejthető, hogy nincs „tökéletes nyelv", mégis választanunk kell! Tehát eljutottunk fejezetünk egyik fő kérdéséhez:

## *Miért jó a Pascal, mint első nyelv?*

A kérdés megválaszolásához nézzünk egy feladatot és annak egy megoldását! Legyen adott egy egészekből álló számsorozat. Az a feladat, hogy döntsük el, van-e köztük páros szám!

Tegyük fel, hogy specifikáltuk a programot. Ebből kiderül, hogy a bemeneteink egy darabszám lesz, ami megadja a sorozat elemszámát, illetve egy tömbben tárolható sorozat, benne N db egész számmal, a kimenetünk pedig egy logikai érték.

Tehát ismerjük a „főszereplőket", és van egy definíciónk a megoldásra: azokat a számokat tekintjük párosaknak, amelyek kettővel osztva 0 maradékot adnak (ha „X" egy ilyen szám: X mod 2=0 ⇔ „X" páros). E tudás birtokában kezdjük el írni az algoritmust[5]:

[5] Az ELTE IK µLógia nevű jegyzetsorozatában követett algoritmusleíró szokásaitól anyiben eltérek, hogy a sorvégi kommenteket „//" jellel vezetem be.

**Program** Páros:

| | |
|---|---|
| **Konstans** | MaxN: Egész(10) |
| **Típus** | tTömb=Tömb(1..MaxN: Egész) |
| **Változó** | N: Egész; T: tTömb //Ezek a bemeneti adatok |
| | Talált: Logikai //Utóbbi a kimenet |

Beolvasás(N,T)
Feldolgozás(N,T,Talált)
Kiírás(Talált)
**Program vége.**

**Eljárás** Beolvasás(**Változó** N: Egész; T: tTömb):
Be: N [N≥0] //Mivel N a darabszámot határozza meg
Be: T(1..N)
**Eljárás vége.**

**Eljárás** Feldolgoz(**Konstans** N: Egész; T: tTömb; **Változó** Talált: Logikai):
**Változó** I: Egész
I:=1
Talált:=Hamis
**Ciklus amíg** I≤N **és nem** Talált
Talált:=(T(I) páros)
I:+1
**Ciklus vége**
**Eljárás vége**

**Eljárás** Kiírás(**Konstans** Talált: Logikai):
**Ha** Talált **akkor** Ki: Találtam páros számot
**különben** Ki: Nem találtam számot
**Eljárás vége.**

Pascal nyelven ekképp néz ki:

```
Program Paros;

Const MaxN=10;

Type
   tTomb=Array[1..MaxN] Of Integer;

Var
   N: Integer;
   T: tTomb;
```

Bemeneti adatok

```
    Talalt: Boolean;
```
← Kimenet

```
Procedure Beolvas(Var N: Integer; Var T: tTomb);

    Var
          I: Integer;

    Begin
          Writeln('Hány elemből áll a sorozar?');
          Readln(N); {Lehetőség van ellenőrizni N értékét, hogy jól
                      lett-e megadva, de ez bonyolítja a kódot}
          Writeln('Kérlek, írd be a számokat!');
          For I:=1 To N Do
                Begin
                      Readln(T[I]);
                End;
    End;

Procedure Feldolgoz(N: Integer; T: tTomb; Var Talalt: Boolean);

    Var
          I: Integer;

    Begin
          I:=1;
          Talalt:=False;
          While (I<=N) And (Not Talalt) Do
                Begin
                      Talalt:=((T[I] Mod 2)=0) {T[I]paros}
                      Inc(I);
                End;
    End;

Procedure Kiiras(Talalt: Boolean;);

    Begin
          If Talalt  Then Writeln('Találtam páros számot!');
                     Else Writlen('Nem találtam páros számot!');
    End;

Begin
    Beolvas(N,T);
    Feldolgoz(N,T,Talalt);
    Kiiras(Talalt);
End.
```

← Előre definiáltuk a változókat

Alapszavak „értelmessége” tekintetében pozitív példa a Pascal. Ha valaki tud kicsit is angolul, a kódból egyértelművé válik számára, mit is szeretne a program (pl. Read – Olvas, False – Hamis stb.). Kiemelésüket tekintve sokszor a környezet nyújt segítséget. A Pascal nyelv alapszavait a Borland által készített Turbo 7.0-s változat pl. más színnel emeli ki (a példában kék színnel).

A programszerkezet könnyen követhető és memorizálható. Elkülönített részek vannak a konstansoknak, a típusdefiníciónak, változók deklarálásának, a programtörzs helyének, minden eljárásban és függvényben. A „sorrendiség” is – a maga módján – logikus, bár az algoritmusbeli létrejöttéhez képest fordított! Csak olyat tudunk felhasználni, amit korábban már definiáltunk. Tehát, a „szervezés” tekintetében nagyon hasonlít az algoritmusra.

A fentiek után felmerülhet bennünk a kérdés: mi szükség van az algoritmusra, ha úgyis ennyire közel áll az amúgy is nélkülözhetetlen kódhoz? A programozás tanításnak nem lehet az a célja, hogy a programozni vágyó tanulót egy nyelvre tanítsa meg. Mint korábban írtam, nem helyes, ha „Pascal programozókat" vagy „C programozókat" képzünk. Az algoritmikus alapon való tanításnak az a lényege, hogy egy univerzális eszközt adjunk a programot írni vágyók kezébe. Így, mint korábban utaltam rá, nem válik a tanuló „nyelvfüggővé". Ha ebből az elvből indulunk ki, jó választás lesz a Pascal, ugyanis nincs „távol" az algoritmikus nyelvtől, tehát nem túlzottan bonyolultak a nyelv kódolási szabályai. Ezek az eltérő „szabályok" pl. a korábban említett, nem túl egységes paraméterkezelése (mikor kell vesszőt vagy pontosvesszőt használni). Ezekből az „anomáliákból" nincsen sok, és könnyen meg lehet tanítani a tanulókat „bánni" velük.

Vegyük észre, hogy amiben az algoritmikus nyelv (első látásra) kidolgozatlannak látszik, a beolvasást/kiírást tekintve, éppen ebben térnek el egymástól legmarkánsabban a procedurális nyelvek; sőt a Delphi programba – a maga „vizualitása" miatt – átvihetetlen lenne a beolvasás/kiírás esetleg „jobban kicsiszolt" megoldása. Az algoritmus számunkra lényeges részei „stabilnak" tekinthetőek.

E Pascal nyelvet elemző gondolatsor lezárásaként felteszem a kérdést: vajon naprakész tudást adunk-e a tanulóknak, ha a Turbo Pascal környezetben tanítjuk programozni? Nem lenne-e jobb más környezetet használni, ami még a tanulást/tanítást is hatékonyabbá tenné? A kérdésre a következő fejezet adja meg a választ.

## 1.2 Mit remélhetünk a Delphitől?

A Delphi nem tekinthető önálló, új nyelvnek. Helyesebb azt mondani, hogy egy Pascal nyelven alapuló, a Turbo Pascallal szoros nyelvi rokonságban álló, új programozási környezet. A „Windows-os" korszakkal együtt érkezett közénk, amikor felmerült az igény Windows alapú programok fejlesztésére. A Pascalt némileg kibővítették, de sajátos logikájához illeszkedő szintaxissal. Tehát pusztán nyelvi szempontból a Pascallal azonos elbírálás alá esik. Azaz a Delphi választása nyelvi szempontból helyeselhető.

A környezet teljesen más. Két programnyelvi generáció telt el a Turbo Pascal születése óta. Akkor még épp, hogy csak elkezdtek beszélni az objektumorientáltságról, vizuális környezetről.[6] Ma már ebbe „születnek bele" a programozási nyelvek és környezetek.

A lényegi kérdés: előnyös vagy hátrányos lenne, ha a tanulók első programozási nyelvi környezetként nem a Turbo Pascalét, hanem a Delphiét ismernék meg?

Egyértelmű, hogy más módszerekkel kellene tanítani a programozást, ha vizuális fejlesztőkörnyezetet használunk. Amíg a Turbo Pascalban a beolvasás/kiírás az algoritmus szerves – néha elbonyolódó – része, a vizuális környezetben ezt a funkciót maga a keretrendszer teszi hozzá a készülő programhoz. Habár, végeredményben Delphiben is változókkal fogunk operálni, a megoldás gyakorlati menete mégis más. A felhasználó annyit

[6] Továbbiakban is – engedve a nem túl precíz szóhasználatnak –: *vizuális környezet*ről fogunk beszélni, jól lehet *grafikus felhasználói felület* (Graphic User Interface = GUI) lenne a pontosabb kifejezés.

lát csak, hogy valamit beírt a szerkesztőmezőkbe, majd megnyomott egy gombot, és a végén egy párbeszédablak közölte vele a program kimenetét. Nagyon leegyszerűsítve ez történik. A programozás egésze szempontjából nem sokban különbözik ez a Turbo Pascalétól. Amiben más, azt a fejlesztő rendszerek filozófiája közti különbség hozza magával. Amikor a felhasználó kitöltötte adatokkal az űrlapot, rákattint egy gombra (GUI), ennek hatására lefut egy ún. OnClick esemény (OOP)[7]. Ebben az eseményvezérlőben kell megírnunk azt, amit eddig „lényegi programként" emlegettünk a Turbo Pascal használata közben. Magyarul: „konkrétan" *nem mi* olvasunk be, és *nem mi* írunk ki „jól megszokott" utasításokkal, hanem a GUI által szolgáltatott beépített eszközök.

Azt kell „megtanulniuk" a diákoknak, hogy egy szerkesztőmezőt, s annak az értékét (a beírt szöveget) a programból úgy tudják elérni, mint egy közönséges változót. Tehát, amikor a felhasználó beírta a szöveget, az már „ott van", s fel lehet használni a programban. Innentől kezdve „gyerekjáték" a feldolgozás, mert ugyanazt kódolja le a tanuló, mint eddig a „hagyományos", Turbo környezetben.

Fentebb szó volt arról, hogy Turbo Pascalban nagyon elbonyolíthatja a kódot a beolvasás „bolondbiztossá" tétele. Delphiben elegánsabban lehet ellenőrizni a beolvasás helyességét. Nem szükséges hosszú, speciális kóddal bajlódni, és/vagy fordítási direktívákat kapcsolgatni, hanem elég megismertetni a tanulóval az ún. *kivételkezelés*t – akár algoritmikus alapokról indulva. Nézzünk erre egy példát, amelyben az algoritmusleíró nyelvet „igazítom" ehhez a problémához! (Továbbra is vastagon szedve emelem ki a kulcs-szavakat.)

```
Eljárás BeGomb_kattint:
  Változó
    Szám: Egész

  VédettRész
    VédettTörzs
      Szám:=Szöveg_Egésszé(BeSzöveg)
      Ki: Szám
    HibaKezelés
      KonvertálásiHiba esetén    Ki: „Konvertálási hiba történt!"
  VédettRész vége
Eljárás vége.
```

Egy szövegmezőből (ami – természetesen – szöveg típusú) szeretnénk számot beolvasni. Kivételkezelés nélkül nem fog elszállni a program, de elég „barátságtalan", ráadásul angol nyelvű, hibaüzenetet kapnánk illegális inputra. Kivételkezelést használva viszont mi tudjuk a kezünkbe venni ezt, ráadásul roppant egyszerűen. A kivételkezeléssel megírt programrész – hasonlóan a Pascalban, a direktívák közötti részhez – védetté válik, magyarul, ha értelmetlen adatot kap a program, nem fog „fejre állni". A Delphi rendszere jó néhány beépített kivételosztállyal, úgy is fogalmazhatnék: hibacsoporttal rendelkezik. (Pl. ilyen a példában szereplő konvertálási hibák, matematikai hibák stb.) Az ún. *Védett Törzsben*

[7] OOP = Objektum Orientált Programozás

elvégezzük a kívánt feladatot, és ha hiba történne (jelen példában ez a konvertálásnál fordulhat csak elő), egyből átugrik az ún. *HibaKezelés* részbe, ami tájékoztatja a felhasználót a hibáról. Tehát, csak akkor fog kiírásra kerülni a szám, ha tényleg számot adtunk meg.

Delphiben ugyanez:

```
procedure TKivetel.gBeClick(Sender: TObject);

Var
  Szam: LongWord;

Begin
 try
  Szam:=StrToInt(edSzam.Text); //Itt lehet "elcsúszni", emiatt
                               //        kell a kivételkezelés
  ShowMessage('Beolvastam, hiba nélkül: '+IntToStr(Szam));
 Except
  on EConvertError do MessageDlg('Konvertálási hiba történt! Nem
                              számot adtál meg!',mtError,[mbOK],0);
 end;
end;
```

Vegyük észre, hogy szituációtól függően a „Ki.” kódjaként hol „ShowMessage”, hol „MessageDlg”-t használtunk. A különbség a szituáció ismeretében könnyen érthető, és megjegyezhető.

Nézzünk egy másik példát arra az esetre, ha olyan hibát kell lekezelnünk, ami nem szerepel a Delphi kivételosztályaiban. Írjunk egy függvényt, ami kiszámolja egy számnak a négyzetgyökét, ám közben le is ellenőrzi, jó-e a szám:

**Függvény** Gyök(a: Valós): Valós
  **Típus**
    GyökvonásHiba ∈ Kivételek **Osztálya**

  **VédettRész**
    **HibaDefiníciók**
      a<0 **esetén** HibaGenerálás(GyökvonásHiba,”Negatív szám a gyökjel alatt!”)

    **VédettTörzs**
      Gyök:=Négyzetgyök(a)
    **HibaKezelés**
      GyökvonásHiba **esetén** Ki: Hibaüzenet
                    Gyök:=0
  **VédettRész vége**
**Függvény vége.**

Mivel olyan hibát akarunk lekezelni, amit nem ismer a rendszer alapértelmezésben, új változót vezetünk be, *GyökvonásHiba* néven. Az első blokk, ami az előző példához képest új, e hiba definiálását végzi. Egy ilyen hibadefiníciós rész – általános esetben – végig veszi

azokat az eseteket, amelyek ha bekövetkeznének, megtörténne az előbb említett „fejre állás”. Olyan, mint a matematikában egy függvény értelmezési tartományának vizsgálata. Amennyiben valamelyik eset fönnáll, „generál” egy hibát. A *HibaGenerálás* eljárás része a fejlesztői környezetnek, nem kell megírnunk. Ezek után *csak akkor* megy bele az ún. *védett törzsbe*, ha nem generálódott semmilyen hiba, különben *HibaKezelés* rész fut le, ahol a program tájékoztatja a felhasználót a hibáról.

A Delphi-„átirat”:

```
Function Gyok(Var a: Real):Real;

Type
    GyokvonasHiba = Class(Exception);

Begin
       try
              If a<0 Then
                 raise GyokvonasHiba.Create(
                                  'Negatív szám a gyökjel alatt!');
                 Gyok:=sqrt(a);
       except
             On Hiba:GyokvonasHiba Do
             ShowMessage(Hiba.Message);
             Gyok:=0;
End;
```

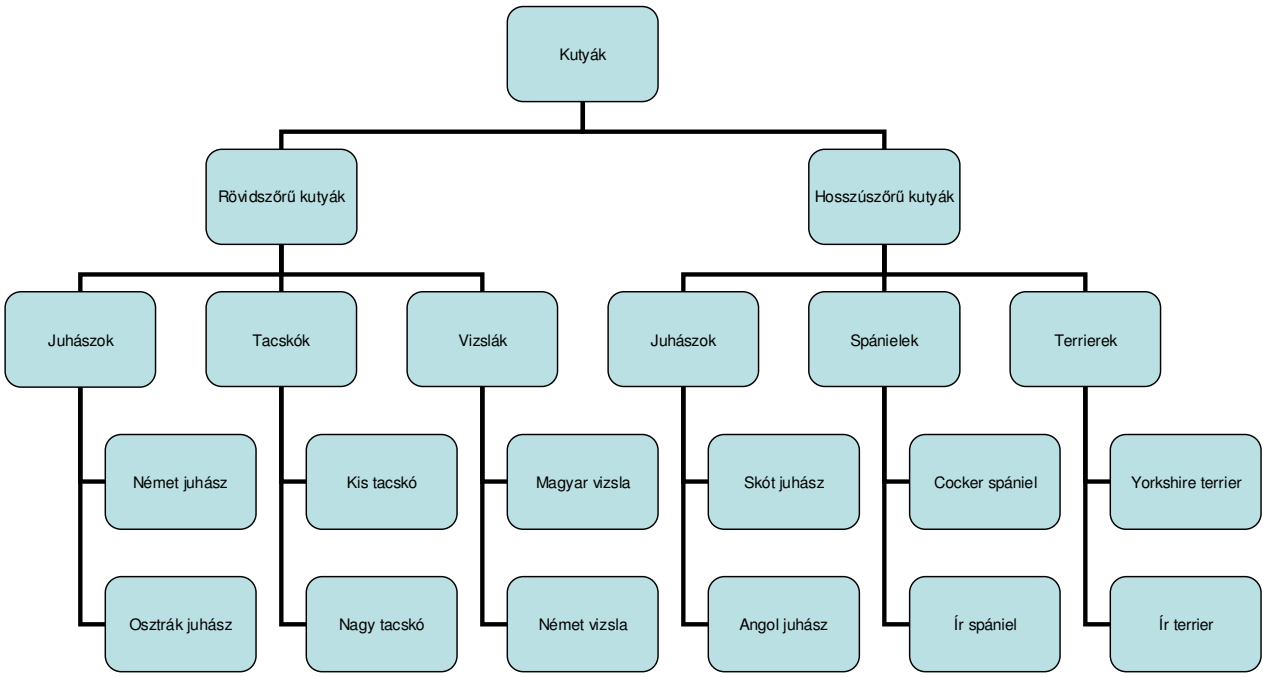
Két részből áll a kivételkezelés. Az első rész – a kritikus pont –, mikor olyan értéket kaphat a változó, amitől – normális esetben, kivételkezelés nélkül – elszállna a program. Csak akkor lép be a hibát lekezelő blokkba – a második részbe –, ha az első blokkban hiba történt. Az algoritmus alapján megtanítható a diákoknak, hogy mi is az a kivételkezelés, mikor fut le a „Hiba ellenőrzése” utáni blokk.

A kiíratás is a vizuális fejlesztőrendszer „mássága” miatt különbözik. El kell dönteni, hogy milyen ún. *komponens*re szeretném kiírni az eredményt. Ezek után a kiírás módja (azaz az utasítás, vagy OOP szóhasználatával mondva: ún. metódus neve és paraméterei) a komponens tulajdonságaitól fog függeni.

Ami különösen fontos, hogy a beolvasáson és a kiíratáson kívül (a kódolás szempontjából) minden ugyanaz lesz, mint eddig. Lényegében, ugyanazt a kódot fogják a tanulók megoldásként begépelni.

Az előzőekben az OOP néhány kulcsfogalmát kimondva vagy kimondatlanul használnunk kellett. Kérdésként vetődik fel, hogy szükséges-e gimnáziumban megtanítani az objektum orientáltság fogalmát? Nem lesz-e túl bonyolult egy kezdő programozó számára?

Véleményem szerint elmagyarázhatók e fogalmak egy ilyen korú tanulónak Számos jó és könnyen érthető analógia található az „objektumság” bemutatására. Lássunk egy példát erre! Az alábbiakban a Kutya objektum, helyesebben a Kutya objektumosztály, röviden osztály megközelítését írom le:




*A „Kutya objektum-osztály"*

Vannak a Kutyák, bizonyos tulajdonságokkal (mondjuk: ugatás, színe, farokhossz…). A Hosszúszőrű kutyákról még tudjuk, hogy hosszúszőrűek, a Juhászokról, hogy milyen állatokat terelnek… és így tovább. Ezen keresztül akár az öröklődést is meg lehet mutatni.

A Jedlik Oktatási Stúdió egyik tankönyve[8] ugyancsak egy vizuális környezeten keresztül tanítja a programozást. Elképzelhető olyan megközelítés is, amelyben „elhallgatható" az objektum orientáltság.

Ami biztos, hogy ha Delphivel oktatnánk a programozást, naprakészebb gyakorlati ismereteket tudnánk átadni tanulóinknak. Mindezek mellett grafikus alapokon „folytatódna" a programozás a LOGO után, így megmaradna a „vizuális élmény".

## 1.3 Mai elképzelés a középiskolák programozás oktatásáról

Kezdetben, az informatika használatát a programozás jelentette. A hardverek és szoftverek fejlődésével egyre több felhasználóbarát szoftver került forgalomba. Ma az informatikai eszközök használata nem igényel programozási ismereteket. Az oktatásban a programozást kiszorítják az alkalmazói ismeretek. Pedig sokak szerint a programozás ugyanolyan fontos, mint a matematikai ismeretek egy mérnök életében.[9]

Nyilvánvaló, hogy a középiskolások csak egy része fog „hivatásszerűen" programozással foglalkozni az iskola befejezése után. Tehát a programozástanításnak nem lehet az az elsődleges célja, hogy profi programozókat képezzen. Erre ott vannak az egyetemi és OKJ-s képzések. Középiskolában legfontosabb célja a programozás oktatásának: egy sajátos gondolkodásmódot (az algoritmikus gondolkozását) elsajátíttatni, amit fel tud használni a

[8] **[FSz]**
[9] Sípos Marianna: A programozásoktatás megújulása a Visual Studio .NET kínálta lehetőségekkel

mindennapi élet számos területén. Szakdolgozatom elején írott módszernek ez a gyakorlati „kimenete": a tanuló lássa, hogy a körülötte zajló „Élet" folyamatait le lehet írni algoritmusokkal, fel lehet bontani elemi lépésekre, s így könnyebben megérti működésüket.

Mégis mennyi időnk van programozást tanítani? Az Oktatási Minisztérium elképzelése szerint (NAT 2003.), felső tagozatban, 5. osztályban heti fél, 6-8. osztályban heti egy óra van informatikára[10]. Ez nagyon kevés! Ha a Mozaik kerettantervrendszer szerint nézzük az óraszámokat, jobb eredményre jutunk: 5-6., 8. osztályban heti egy-egy, 7., 10-11. osztályban heti másfél és 9., 12. osztályban heti két óra.[11] Látható, hogy a hangsúly az utolsó négy éven van. Ez a tanterv csak a 2004/2005-ös tanévben lépett életbe, s felmenő rendszerben kerül bevezetésre.

Jelenleg a kerettanterv szerint tanul a diákok túlnyomó többsége, ahol minden eddiginél kevesebb óraszámot adtak az informatikának: 5. osztályban heti fél, 7-8. osztályban heti egy, és 9. osztályban heti két óra. Ilyen helyzetben nagy kihívás elvárható minőségben oktatni. „A középiskolai kerettanterv leginkább egy lóversenypályához hasonlít: sokat akar nagyon kevés tanórában. Ahhoz képest viszont, hogy valójában milyen számítástechnikai tudásra lenne szükség, a kerettanterv kevés" nyilatkozta Péterfi Árpád a HVG-nek 2002-ben.[12]

Ennek a kevés órának a jelentős része alkalmazói ismeretek tanítása, kisebb hányada informatika-elmélet, s csak az órák negyedét-ötödét tudjuk arra használni, hogy programozni tanítsuk diákjainkat. Nem vagyunk könnyű helyzetben.

[10] Magyar Közlöny 2004/68/11 szám, 96. o.
[11] Magyar Közlöny 2004/68/11 szám, 600. o.
[12] Péterfi Árpád a XV. kerületi Dózsa György Gimnázium számítástechnika tanára nyilatkozta a HVG 2002. áprilisi Háló különkiadásában a Módszertelenül, Számítástechnika-oktatás a középiskolában c. cikkben. Nádori (2002) Péter: Háló, Kalauz az Internethez, HVG 2002. április 27-i számának melléklete.

# 2 Kérdőíves vizsgálat

## 2.1 A cél

A vizsgálat célja, hogy mérhetővé tegye, számszerűsítse a diákok hozzáállását, attitűdjét a programozáshoz, illetve felmérje tanulási „szokásaikat" és az iskolai-otthoni környezet számunkra lényeges jellemzőit. Valamint összefüggéseket keressen a programozás eredményességét – vélhetően – meghatározó tényezők között. Választ keres arra továbbá, hogy milyen változtatásokkal – a módszerekre és egyéb, tanítási körülményekre vonatkozóan – lehetne a jelenlegi helyzeten gyökeresen javítani.

## 2.2 Módszerek és minta

Budapest több középiskolájából 75 diák töltötte ki a 32 kérdésből álló kérdőívet. A megbízhatóságot az ún. Cronbach-féle alfa (α) együtthatójával ellenőriztem.[13] A kérdőív alapjául két publikált kérdőív szolgált: az egyik gyerekek tanulási stílusát[14], a másik a tanulási stratégiáját vizsgálta.[15]

A kérdőív nyolc témakörben vizsgálódik. Vizsgálja a tárgy (programozás, ha nincs, akkor az informatika tárgy) megítélését, annak tekintetében, hogy elegendő-e a meglévő óraszám és a tartalmi elvárások szintje. A kérdések másik csoportja a diákok motiváltságát teszi nagyító alá. Megfogalmaztam kérdéseket a tanulási szokások tisztázása érdekében, amelyek főleg azt vizslatják, hogy a tanuló használ-e algoritmust, illetve, hogy segítséget nyújt-e neki az algoritmus a kódolási folyamatban. A motivációval összefüggő témában, a tanuláshoz való hozzáállással kapcsolatban is találhatók kérdések. Bizonyos kérdések azt firtatják, hogy a diákok találkoztak-e már vizuális fejlesztőrendszerrel, s hogy szívesen használnák-e. A maradék három témakör az iskolai és az otthoni környezetet, illetve a versenyeken való részvétel „mértékét" vizsgálja. Lássuk a kérdőívet!

## 2.3 A kérdőív

### Kérdőív informatika órán belül programozást is tanuló középiskolásoknak

Olvasd el figyelmesen az alábbi mondatokat! Döntsd el, hogy a válaszok közül melyik jellemző rád és azt a számot/választ jelöld meg! Itt nincsenek helyes vagy helytelen válaszok. Csak a saját véleményed fontos.

1 = ***Egyáltalán nem*** jellemző rám; egyáltalán ***nem*** tartom ***igaz***nak.
2 = ***Nem*** jellemző rám; ***nem*** gondolom ***igaz***nak.

[13] `[SPSS,CM]`
[14] Dr. Szitó Imre: A tanulási stratégiák fejlesztése (Iskolapszichológia 2.), ELTE Eötvös Kiadó, Budapest, 2003.
[15] Dr. Balogh László: Tanulási stratégiák és stílusok, a fejlesztés pszichológiai alapjai, 7-12. o., KLTE, Debrecen, 1993. Egyetemi jegyzet

3 = Nem tudom eldönteni, igen is, nem is.
4 = Jellemző rám; ***igaz***nak gondolom.
5 = ***Nagyon*** jellemző rám; ***nagyon igaz***nak tartom.

A 3-as választ csak ritkán használd!

### *"Személyes" adatok*

Nemed: Férfi Nő

Hányadikba jársz?:

Hány informatika/programozás órád van egy héten?:

Iskolatípus: Gimnázium Szakközépiskola Informatikai szakközépiskola

### *A tárgy megítélése*

1. Több informatika/számítástechnika órára lenne szükség, mint most. 1 2 3 4 5
2. Több programozásra lenne szükség, mint most. 1 2 3 4 5
3. A tanórai elvárások nagy része felesleges. 1 2 3 4 5

### *Motiváció*

4. A programozás a legnehezebb része az informatika/számítástechnika óráknak. 1 2 3 4 5
5. Élvezem a programozást (nem feltétlenül az órán). 1 2 3 4 5
6. Fel tudom használni mindennapi életemben a programozáson tanultakat. 1 2 3 4 5
7. Egyetemi/főiskolai tanulmányaim folyamán szeretnék programozást tanulni. 1 2 3 4 5
8. Szívesen keresnék pénzt később programozással. 1 2 3 4 5
9. Érdekesnek találom a programozás-órát. 1 2 3 4 5

10. A programozás tanulása közben gyakran kedvet kapok, hogy utánanézzek dolgoknak. 1 2 3 4 5

### *Tanulási szokások*

| | Tanultam | Nem tanultam | Fogok tanulni | |
|---|---|---|---|---|
| 11. Tanultál-e (fogsz-e tanulni) algoritmust írni? | | | | |
| 12. Az algoritmusírás segít abban, hogy közelebb jussak a feladat megoldásához. | 1 | 2 | 3 | 4 5 |
| 13. Az algoritmus szükséges, hogy biztonsággal tudjam kódolni a programot. | 1 | 2 | 3 | 4 5 |
| 14. A tankönyv (programozásból) sokat segít (segítene) a tananyag megértésében. | 1 | 2 | 3 | 4 5 |

### *Tanuláshoz való hozzáállás*

15. Mindig igyekszem megérteni a dolgokat, még ha először ez nagyon nehéznek látszik is. 1 2 3 4 5

16. Amit tanulok programozásból, azt igyekszem kapcsolatba hozni a saját tapasztalataimmal. 1 2 3 4 5

17. Azt szeretem, ha a tanárok sok szemléltető példát, saját tapasztalatot említenek, hogy megértessék velünk a dolgokat. 1 2 3 4 5

18. Hogy jobban megértsem, amiről tanulok, a mindennapi tapasztalataimmal igyekszem kapcsolatba hozni. 1 2 3 4 5

### *Vizuális környezetre való "áttérés"*

| | Igen | Nem |
|---|---|---|
| 19. Találkoztál már korábban vizuális fejlesztőrendszerrel (pl. Borland Delphi, Microsoft Visual Studio)? | | |

| | | | | |
|---|---|---|---|---|
| 20. Jó (lenne), ha vizuális fejlesztőkörnyezetben tanulok (tanulnék) először programozni. | 1 | 2 | 3 | 4 5 |

21. Ha választanom lehetne, szívesebben írnék „Windows-os” programokat. 1 2 3 4 5

### *Iskolai környezet*

22. Elégedett vagyok az iskola számítógép-ellátottságával. 1 2 3 4 5

23. Elégedett vagyok az iskola szoftver-ellátottságával. 1 2 3 4 5

24. Jól használhatóak a labor/gépterem gépei az iskolai munkára. 1 2 3 4 5

### *Iskolán kívüli, „otthoni” környezet*

25. Édesanyád vagy édesapád használ számítógépet a munkahelyén? Igen Nem

26. Édesanyád vagy édesapád használ számítógépet otthon? Mindennap Hetente Soha

27. Te használsz otthon számítógépet? Mindennap Hetente Soha

28. Mire használod az otthoni számítógépedet (többet is bejelölhetsz)? Programozás | Szövegszerkesztés | Játék | Zene | Grafika | Internet | Semmire

29. Milyen operációs rendszer van az otthoni számítógépeden (többet is bejelölhetsz)? Windows 95 | Windows 98 | Windows Me | Windows 2000 | Windows XP | Windows 2003 | Mac OS X | Linux | Unix | BeOS | Egyéb | Nem tudom

30. Ha elakadsz otthon egy számítógépes problémánál, kapsze segítséget családtagjaidtól? Gyakran | Hébe-hóba | Soha | Nem kérek segítséget

***Versenyeken való részvétel***

31. Szoktál-e részt venni program-ozási versenyen? Gyakran Próbáltam már Nem is akartam Nem is tudok róla

32. Szoktál-e részt venni alkalmazói versenyen? Gyakran Próbáltam már Nem is akartam Nem is tudok róla

A célcsoport középiskolák 7-11. évfolyamának egy-egy osztálya, gyakorlati csoportja. Egyik számítástechnika órán töltötték ki ezt a kérdőívet, amelyet a világhálón érhettek el[16], s az eredményeket egy szerver tárolta el. Innen dolgoztam fel a teszt-eredményeket.

A kiértékelés folyamán kiszámoltam a kérdések, kérdéscsoportok átlagát és szórását, továbbá a válaszok százalékos eloszlását.

## 2.4 Előzetes megjegyzések a kiértékeléshez

Több területen várok a válaszok között összefüggést. Tapasztalataim szerint a fiúkat jobban érdekli a programozás, és az iskolatípus megválasztása is hatással lehet a motivációra, hozzáállásra. Az iskolatípus és az iskolai „informatikai" környezet között is sejtek összefüggést, hiszen egy informatikai szakközépiskola – várhatóan – többet áldoz informatikai eszközökre, mint egy általános gimnázium.

Érdekes megvizsgálni az iskolatípus és a tanulási szokások, illetve a tárgy megítélése, a motiváció és a hozzáállás kapcsolatát. Várhatóan egy informatikai középiskolában több időt szánnak programozásra. Meg kell vizsgálni, hogy az otthoni környezetnek van-e hatása a motivációra, hozzáállásra, illetve a motiváció kapcsolatát azzal, hogy mire használja szabadidejében a számítógépet.

## 2.5 A tesztek kiértékelése

A 3. kérdésnél fordítottan számoltam a pontokat, azaz, ahol a válaszadó diák az 5-öst adta válaszul, ott egy pontot adtam, ugyanis így függ jobban össze az előtte levő két kérdéssel: pl. ha valaki egyetért azzal, hogy túl nagy az elvárás (ezt jelentené az 5-ös válasz), akkor valószínűsíthető, hogy nem szeretne több órán részt venni programozásból és informatikából. Hasonló megfontolásból számoltam ugyancsak fordítottan a 4. kérdést, itt a kérdéscsoport további kérdéseivel függ így jobban össze.

Eredmények: 75 diák írta meg a kérdőíveket. A mintát 36%-ban lányok, 64%-ban fiúk alkották. Négy évfolyamból tevődtek ki a diákok: 17%-uk hetedik, 32%-uk nyolcadik, 40%-uk kilencedik és 11%-uk tizenegyedik osztályba jár.

[16] http://ttkhok.elte.hu/kergezerge/kerdoiv/kerdoiv.html

A kérdéscsoportonkénti átlagok a követezők szerint alakultak:

| Kérdéscsoport neve | Átlag | Kérdéscsoport szórása |
|---|---|---|
| Tárgy megítélése | 2,70 | 0,94 |
| Motiváció | 2,40 | 1,04 |
| Tanulási szokások | 2,99 | 0,92 |
| Hozzáállás | 3,18 | 0,87 |
| Vizuális környezet | 3,37 | 0,92 |
| Iskolai környezet | 3,81 | 0,97 |
| Otthoni környezet | 4,35 | 0,53 |
| Versenyek | 2,96 | 0,72 |

## 2.6 A teszteredmények elemzése

### *Amik a számok mögött vannak*

A magas szórások mutatják, hogy nem volt egységes a válaszadók véleménye, s hogy a válaszok túlnyomó többsége a szélső tartományból (1-2 vagy 4-5) kerültek ki. Ez alól csak az otthoni környezet számszerűsíthető eredményei a kivételek.

Kicsit gyengébb, mint közepes a tárgy megítélése. A válaszok eloszlását figyelembe véve (lásd Függelék 1.1) látszik, hogy a válaszadók kb. fele nem szeretne több informatika/programozás órát, s ezek mellett a válaszadók 42%-a úgy gondolta, hogy nem feleslegesek a tanórai elvárások.

A motivációval komoly problémák vannak. Átlagosan a diákok több, mint a fele válaszolt „negatívan” erre a kérdésre. Ebben a kérdéscsoportban volt a legmagasabb a szórás. A magas szórás – mint fentebb említettem – a válaszok szélsőségesebb megoszlását mutatja.

A tanulási szokások eredménye némi bizonytalanságot mutat, amely abból következhet, hogy a kérdőívet kitöltők 2/3-a 7-9-esek, s nem tudtak véleményt formálni ezekben a kérdésekben. Ezt alátámasztani látszik az évfolyamok szerinti kiértékelés. A 12. és a 13. kérdés diagramjain látszik a csökkenő „nem tudom” hányad, ahogy felsőbb évfolyamba járók diagramjai felé haladunk (lásd Függelék, 1.2). Az algoritmus használatát általában a diákok fele hasznosnak találta, de a harmada nem tudta ezt eldönteni. A 14. kérdés válaszainak eloszlása figyelemre méltó erejű: a diákok alig harmada gondolja azt, hogy hasznos számára a tankönyv a programozás tanulásában.

A hozzáállás 3 feletti értéke meglepő a motiváció sokkal alacsonyabb átlaga mellett. A kérdéscsoport négy kérdése közül kettő szólt a saját tapasztalatokról (16. és 18.). Ezekkel az állításokkal diákok fele nem értett egyet. Ez adódhatna ugyancsak a mintát alkotó tanulók fiatalságából: nincs még kellő saját tapasztalatuk. Az évfolyamonkénti vizsgálódás cáfolta ezt a feltevést, ugyanis a teljes minta eredményihez hasonlatosak jöttek ki. Ezek szerint általánosan igaz, hogy valami ok miatt nem vagy tudják, vagy nem akarják a tanulók a programozás-órán tanultakat kapcsolatba hozni a saját tapasztalataikkal. Közel kétharmaduk vélte úgy, hogy igyekszik megérteni a tárgy „nehéz részét" is, ezért szükségét érzik sok szemléltető példának.

A megkérdezett tanulók 57%-ka nem találkozott vizuális környezettel, mégis több, mint 40%-uk gondolta vagy érezte úgy, hogy jó lenne, ha első magasabb szintű programnyelvét vizuális fejlesztőkörnyezetben tanulná, illetve szívesebben írna „Windowsos" programokat. A 20. kérdésre érkezett nagyarányú (37%) „nem tudom" válasz okát is a tapasztalat hiányában látom, és alátámasztja a 19. kérdés válaszainak az eloszlását. Ha évfolyamok szerint vizsgáljuk meg az eredményeket, hasonló következtetésre jutunk. Hozzá szükséges azonban tennem, hogy a vizuális rendszert „támogatók" aránya, az évfolyamban való feljebb haladással növekedik.

Az iskolai környezet megítélése – általában – jónak mondható. A diákok átlag 2/3-a értett egyet vagy gondolta igaznak a kérdéscsoportban feltett állításokat.

Az otthoni környezet számszerűsíthető eredményei azt mutatják, hogy a számítógépnek fontos szerepe van a családban. A szülők és gyermekeik is hetente többször használják, és a diákok 40%-a gyakran kér segítséget családtagjaiktól. A számítógép-használat fontosságát mi sem jelzi jobban, mint az, hogy a tanulók szüleinek mindössze 11%-a nem használ számítógépet a munkahelyén. Kitűnt, hogy a kitöltők a számítógépet leginkább játékra, zenére és internetezésre használják. A használt operációs rendszerek közül toronymagasan a Windows XP „nyert".

A versenyekkel kapcsolatos kérdésekre adott válaszok átlaga azt mutatja, hogy a diákok nagy része nem szeretne versenyekre menni. Ez következik a nem túl magas motiváltságból.

## *Kapcsolatvizsgálat*

Több kérdéscsoport között vártam összefüggést, s habár néhány átlagos eredmény arra enged következtetni, hogy egyes kérdéscsoportok között magasabb korreláció várható, számomra meglepő korrelációs értékek születtek:

| **Korrelációszámítások** | |
|---|---|
| Tárgy megítélése – Motiváció | 0,52 |
| Tárgy megítélése – Hozzáállás | 0,57 |
| Motiváció – Hozzáállás | 0,49 |

| **Korrelációszámítások (folytatás)** | |
|---|---|
| Motiváció – Vizuális környezet | -0,01 |
| Hozzáállás – Vizuális környezet | -0,05 |
| Motiváció – Versenyek | 0,22 |
| Hozzáállás – Versenyek | -0,07 |
| Tárgy megítélése – Versenyek | 0,14 |

A tárgy megítélése, a motiváció és a hozzáállás közötti magasabb korreláció igazolta várakozásaimat. Ezek után meglepőek a(z alacsony) negatív korrelációk.

Az volt a feltevésem, hogy a jobb hozzáállással rendelkező és motiváltabb tanulók könnyebben nyitnak az új dolgok felé. Ezzel szemben a korreláció alacsony értéke azt sugallja, hogy a kevesebb motivációval rendelkező diákok foglalkoznának szívesebben a vizuális környezettel.

Érdekesség, hogy a motiváció milyen jól korrelált a hozzáállással, mégis a két kérdéscsoport eredményei igen különbözően korreláltak a versenyekkel kapcsolatos kérdések eredményeihez. Ennek lehet oka a fentebb említett „tapasztalatlanság” a fiatalkor miatt. A magas különbség mégis arra enged következtetni, hogy a válaszadók számára fontosabb az a versenyek szempontjából, hogy szeressék a tárgyat, és értelmet találjanak benne. Emiatt lett alacsony a tantárgy megítélése és a versenyek közötti korreláció.

### *Megbízhatóság*

A teszt megbízhatóságát a Cronbach-féle alfa ($\alpha$) együtthatójával mértük, melynek értéke 0,887692825. Így igazolta a kérdőív megbízhatóságát, a fenti kérdéscsoportok közti korrelációkat illetően.[17]

## 2.7 Következtetések – teszt-tervek

Néhány vizsgálni szükséges szempont mérését – pl. az iskolatípus és a motiváció – a kicsi minta, illetőleg a rossz eloszlás miatt nem tudtuk elvégezni. Az egyik legsúlyosabb probléma a motiváció a programozás iránti alacsony volta. Ezen – véleményem szerint – segíthet a vizuális rendszerre való áttérés, ugyanis, az első lépések, amely sokat számít, lényegesen könnyebbek, mint egy DOS-os programozási környezetnél.

A válaszadók korosztálybéli eloszlása sem kedvezett a „tisztább” eredménynek, ugyanis – vélhetően – a tapasztalatlanságuk miatt sok kérdésre negatívan válaszoltak.

A kérdőív jövője így körvonalazódik: sokkalta nagyobb mintára lesz szükség, több fajta szempont szerinti csoportosítással. Ilyen csoportosításra gondolok:

Vizuális vagy nem vizuális fejlesztőrendszer segítségével tanulja-e a programozást. Ez választ ad arra a kérdésre, hogy a „nyelvosztály” használatától függ-e az attitűd.

[17] **[SPSS,CM]**

Fölrajzi elhelyezkedés. Egy iskola lehetőségeit, „informatikai környezetét” erősen befolyásolja, hogy az ország keleti vagy nyugati felében, a fővárosban vagy vidéken található-e az adott intézmény.

Iskolatípus szerint. Az iskolatípus befolyásolja a tanítható órák számát, így a tanítás és a felhasznált eszközök minőségét. Függ-e ettől a tantárgy megítélése és a programozáshoz való attitűd?

A vizsgálat eredményei alapján születne meg – többek közt – egy tanítási módszertan, aminek egyik kézzel fogható „terméke” az a tankönyv lesz – ennek terveiről szól a szakdolgozat harmadik része –, amely – a vizuális fejlesztőrendszerek várhatóan általánossá váló egyikének– a Borland Delphi rendszerét felhasználva építi fel a programozást. Ezt a módszertant az ELTE-IK informatikatanár-képzésének első lépcsőjénél is fel lehet használni.

# 3 Egy leendő tankönyv terve

Szakdolgozatom utolsó nagy része egy jövőbeni könyv egyes fejezeteinek kidolgozása. Céljai, módszerei megegyeznek a dolgozatom bevezetésében leírtakkal: nagy hangsúlyt fektet az algoritmusra, nem a nyelvet tartja központi szereplőnek. Gyakorlati „kimenetként" a Borland Delphi 6 Personal fejlesztőrendszert használja.

A tankönyvet „gyakorlatiasra" tervezem. Az egyrészt ezt jelenti, hogy a tankönyv a diák (olvasó) megszerzett tudására nagyban épít; itt nem csak a szaktárgyi tudásra gondolok, hanem a mindennapi életből szerzett tapasztalataira is. Másrészt azt is jelenti, hogy a könnyebb érthetőség kedvéért egyszerűen, „fiatalos nyelvezettel" fogalmazok, és személyesebb stílussal írok, ügyelve – természetesen – a szakszavak megfelelő használatára.

## 3.1 Bevezetés

Még mielőtt megismernéd a környezetet, s elkezdenél programozni, szükséges, hogy megismerkedj pár alap építőkővel! Először is szeretném, ha megértenéd, mi fán terem egy program. Biztos láttál már programokat: autóverseny, szövegszerkesztő, rajzolóprogram. Külsejük és céljaik különböznek, de egy közös van bennük: valamely feladatokat, problémákat oldanak meg. Pár egyszerű példa, amit a tankönyv elolvasása és megtanulása után tudni fogsz: szövegben keresni, bizonyos szavakat kicserélni egymással, ki tudsz színezni egy alakzatot, stb. Tehát, azt mondhatjuk, hogy a program egy nehezebb problémának a megoldása. Ebben a könyvben csak nagyon egyszerű feladatokat fogunk majd megoldani.

Hétköznapi életed folyamán is szembe találod magadat problémákkal, feladatokkal. Most nem a matematika házi feladatra gondoltam, hanem sokkal egyszerűbbre. Például át akarsz menni egy olyan jelzőlámpás kereszteződésen, ahol „nyomógombos" jelzőlámpa van. Mit teszel? Tudom, át fogsz menni. Próbáld meg végiggondolni, miket is csinálsz ekkor. Először megállsz, majd megnyomod a gombot, aztán addig vársz, míg zöld nem lesz a lámpa, s végül átmész az út túloldalára. Persze, mikor erre a cselekvéssorra gondolsz, nem mélyedsz ennyire bele, hanem egyszerűen „lefuttatod". Olyan, mint ha magadon végrehajtanád a „Jelzőlámpánál való átkelés" programot.

Ugyanígy, mikor, pl. a szövegszerkesztőben a „Keresés" menüpontra lépsz, akkor is sok pici lépés hajtódik végre. **Mikor ezeket a pici lépéseket felsoroljuk, végrehajtásuk sorrendjében** – mint előbb a jelzőlámpánál –, **ezt a felsorolást *algoritmusnak* hívjuk.** Hasonlóan le tudnád írni algoritmussal, hogy miképp jutsz el otthonról az iskoládba. Algoritmusok, pontosabban receptek tárházát tartalmazza minden szakácskönyv.

Az algoritmus elkészítése lesz a legnagyobb feladat, mikor meg szeretnél oldani egy problémát. Ám, egy magyar nyelven írott algoritmust nehezen értene meg a számítógép, pedig érdemes lenne azon futtatni, főleg, ha egy nagyobb problémát szeretnél megoldani. Ezért születtek meg a programozási nyelvek, amik – végső soron – egy algoritmusnak a

leírása, csak egy másik „nyelven”. Meg kell jegyeznem, hogy a programozási nyelv még mindig nem a gép nyelve, szükségünk lesz még valamire, egy tolmácsra. Ezt a tolmácsot fogjuk *fordítónak* (angolul: compiler, ejtsd kompájler) hívni. A fordítóval majd később bővebben is meg fogsz ismerkedni.

Fontos dolog, hogy a számítógép nem ember (ezzel, gondolom, nem mondtam sok újat). Másképp „gondolkodik”, mint Te vagy én. Előző jelzőlámpás algoritmusunkat kicsit finomítani kell ahhoz, hogy „közelebb” vigyük a számítógéphez, hogy minél könnyebben lehessen ebből valamelyik programozási nyelven megírni a programot:

Megállsz.
Megnyomod a ***Gomb***ot. (Ennek eredménye: a ***Gomb*** be van nyomva.)
Addig vársz, amíg ***Lámpa***≠zöld.
Átmész az út túloldalára.

Világos, hogy a gombnak és a lámpának vannak különböző állapotaik. Egy gomb tud benyomva és „kinyomva” lenni, egy lámpa tud piros és zöld lenni. A programozásban az ilyen tulajdonságú **dolgokat, amiknek meg tudjuk változtatni az állapotát, más szóval: értékét, *változóknak* hívjuk**. Minden általunk használt **változónak van egy neve**, úgy mint neked és nekem, **ezt hívjuk *azonosítónak***. Nagyon fontos tulajdonsága, hogy ***egyértelműen azonosítja*** a változót. Előző példában két azonosítóval dolgoztunk: *Lámpa* és *Gomb*.

Ezeknek a változóknak lesz még egy fontos tulajdonságuk. Minden fogalmat, tárgyat be tudsz tenni egy halmazba, jellemezni tudod egy gyűjtőfogalommal. Például: a cinege, a búbos banka és a hattyú *madarak*; a kutyák, macskák, kígyók és a madarak *állatok*, vannak egész számok, racionális számok, negatív számok, stb. **Ugyanígy az általunk használt változókat is be tudjuk sorolni ilyesfajta halmazokba, amit *típusnak* fogunk hívni**. Példánkban, a Lámpa nevű változó tartozzon a *Jelzőlámpák*, a Gomb nevű változó pedig a *Nyomógombok* típusba.

Tehát, pl. minden olyan változó, amely a Jelzőlámpák típusba tartozik, rendelkezni fog azokkal a tulajdonságokkal, amelyekkel egy jelzőlámpának rendelkeznie kell. Ezen tulajdonságok tárháza szinte végtelen, legkönnyebben a számhalmazokkal lehetne elmagyarázni. Mi jellemzi például a természetes számokat: egészek és pozitívak. Csak ennyi jellemezné őket? Nem, ugyanis az azonos típusú elemeket „kölcsönhatásba” lehet hozni egymással. **Ezt a kölcsönhatást hívjuk *műveletnek***. Milyen műveletek vannak a természetes számok halmazában: az összeadás és a szorzás. Nagyon fontos tulajdonsága minden műveletnek, hogy a művelet eredményének is benne kell lennie ugyanabban a halmazban. Ha jobban belegondolsz, más matematikai művelet nem lenne jó, igaz? Hiszen a kivonással, ha egy kisebb természetes számból kivonsz egy nagyobbat, negatív számot kapsz, ami már nincs benne a számhalmazban (pl.: 3-5=-2). Ugyanígy jársz az osztással. Nem minden osztásnak van egész végeredménye (pl.: 5/2=2,5). A programozás világában a műveleteket tágabb értelemben használjuk, mint a matematikában. Ezzel a fejezet végén fogsz megismerkedni.

Tovább szükséges finomítanunk algoritmikus „nyelvünket”, hogy még közelebb vigyük a számítógép gondolkodásához. Hasonlóan az autóvezetéshez, a program „vezetésének”, vezérlésének is vannak szabályai. Ilyen például a már jól ismert jelzőlámpánál való át-

haladás. Tegyük fel, hogy van egy autód. Éppen haladsz egy jelzőlámpás kereszteződés felé. Mikor mehetnél át rajta? Erre könnyű a válasz, csak akkor, ha zöld a lámpa. Így lehet kifejezni algoritmikus nyelven:

**Ha** Lámpa=zöld **akkor** Átmegyek,
**különben** Megállok.

**Az effajta „Ha … akkor … különben …" szerkezeteket *elágazásnak* nevezik**. Azért kapta ezt a nevet, mert valamilyen feltétel teljesülésének függvényében teszünk valamit. A példában láthatod, ha zöld a lámpa, akkor átmész (ez az egyik lehetőség), ha nem zöld a lámpa, megállsz (ez a másik lehetőség). Hasonlóan lehetne leírni azt, hogy pl. Érd városa felé akarsz letérni az autópályáról. Akkor térsz csak le, ha meglátod az „Érd" – táblát. Ezt így lehetne leírni:

**Ha** Tábla=„Érd" **akkor** Lekanyarodok.

Az előző példa kicsit különbözik az előzőtől, ugyanis csak az egyik lehetőség esetén tettünk valamit. Ebben az esetben nincs szükség a különben ágra, mert ha nem az Érd fele mutató tábla mellett mész el, azt teszed, amit eddig: mész tovább.

Elég nehezen lehetne egy utazást algoritmussal leírni, ha csak ennyi eszközöd lenne. Ki tudja, hányszor kellene leírnod: 'Ha Tábla=„Érd", akkor Lekanyarodom' – sort. Ez úgy nézne ki a gyakorlatban, hogy minden táblánál meg kellene állnod, és megnézned, mi van ráírva. Eléggé unalmas és lassú.

Az a probléma, hogy folyton megállsz, és ellenőrzöl, pedig csak *addig* akarsz továbbmenni, folyamatosan nyomni a gázpedált, *amíg* nem érted el az „Érd" feliratú táblát. Mikor ugyanazt a folyamatot, sokszor meg szeretnéd ismételni egymás után (táblafigyelés, majd ha nem Érd, továbbmenni, nyomni a gázpedált), *ciklusnak* nevezzük. Így lehetne kifejezni algoritmussal:

**Ciklus amíg** Tábla≠„Érd"
    Továbbmegyek.
**Ciklus vége**

Mit is jelent? Azt, hogy addig megyek tovább az úton, figyelve a táblákat, amíg nem értem el az Érd felé mutató táblát. Fontos, hogy az olvashatóság és az érthetőség kedvéért mindig jelezd az algoritmusban, hogy mely részek tartoznak egy ciklusba! **A Ciklus … és a Ciklus vége között a *ciklusfeltétel* és a *ciklusmag* található**.

Ugyanezt a problémát így is fel tudnád írni:

**Ciklus**
    Nem kanyarodok le.
**Amíg** Tábla≠„Érd"
**Ciklus vége**

Ez a megoldás is helyes, mégis különbözik az elsőtől. Az a fő különbség, hogy az első megoldásnál a *ciklusba való belépés* feltételét, a másodiknál a *ciklusban való maradás*

feltételét ellenőrizte az algoritmus. Az első megoldást hívjuk *elöl tesztelős*, a másodikat *hátul tesztelős* ciklusnak (annak megfelelően, hogy a ciklusmag előtt vagy után ellenőrizzük a feltételt).

Figyeld meg a két ciklus működése közötti különbséget! Az elöl tesztelős ciklus lehet, hogy egyszer sem fut le, ha a belépési feltétel hamis lesz a ciklusba való első belépés előtt. Ezzel szemben, a hátul tesztelős ciklus egyszer legalább le fog futni, azért, mert – ahogy fentebb olvashattad –, ez a fajta ciklus a ciklusban való maradás feltételét ellenőrzi.

Biztosan kerültél már olyan helyzetbe, mikor kerestél egy utcát, és miután megkérdeztél egy járókelőt, ezt a választ kaptad: „Az ötödik utcánál fordulj jobbra!" Így lehet a járókelő válaszát leírni algoritmussal:

```
Sétálok.
Meglátom az első utcát.
Sétálok.
Meglátom a második utcát.
Sétálok.
Meglátom a harmadik utcát.
Sétálok.
Meglátom a negyedik utcát.
Sétálok.
Meglátom az ötödik utcát.
Jobbra fordulok.
```

Ezt még egész „olcsón" megúsztad, milyen sokat kellett volna akkor írnod, ha a járókelő a tizedik utcát jelöli meg! Ebben a példában a „Sétálok" tevékenységet írtuk le sokszor, ezen lehetne javítani:

```
Ciklus UtcaSzám:=1-től 5-ig
    Sétálok.
    Meglátom az UtcaSzám-adik utcát.
 Ciklus vége
Jobbra fordulok.
```

Mit is jelent ez: elsétálok az ötödik utcáig, és jobbra fordulok. Ebben az esetben pontosan lehetett tudni, hányszor fog lefutni a ciklus. **Az ilyen ciklust *számlálós ciklusnak* hívjuk**. Mikor ilyen ciklussal dolgozol, szükséged lesz egy *ciklusváltozóra* (példában az *UtcaSzám*), ez alapján fogja „tudni" a ciklus, hogy hányadszor futott le, s hogy mikor kell kilépni belőle, hogy tényleg csak annyiszor fusson le a ciklusmag, ahányszor szükséges.

Szoktál napirendet írni vagy határidőnaplót használni? Órarended, napirended biztos van. Egy napod leírását könnyen lehet algoritmizálni. Gyors példa: Felkelsz reggel, fürdesz, reggelizel, elmész suliba, beülsz az első órára,… és így tovább. Ha arra kérnélek, hogy írd le egy heted algoritmusát, sok ismétlődő dolog lenne benne, pl. a felkelés, fürdés, amit minden reggel elvégzel vagy az utazás az iskolába, ami majd' minden reggel ugyanaz. Az effajta

sokszor ismétlődő folyamatoknak adhatsz egy külön nevet, azaz ún. *eljárásokká* „fűzheted” össze őket, így csak egyszer kell kifejtened. Íme egy példa, egy lehetséges napirend:

Napirend algoritmus – eljárásokkal

Reggeli dolgok.
Utazás suliba.
Iskola.
Utazás haza.
Ebéd.
Délutáni dolgok.
Esti dolgok.

**Eljárás** Reggeli dolgok
  Felkelés.
  Reggeli.
  Elindulás suliba.
**Eljárás vége**

**Eljárás** Utazás suliba
  Séta a buszmegállóig.
  **Ciklus**
    Várakozás.
  **Amíg nem** jön a busz.
  **Ciklus vége**
  Felszállás.
  **Ciklus**
    Buszozás.
  **Amíg nem** jön az iskola megállója.
  **Ciklus vége**
  Leszállás.
**Eljárás vége**

**Eljárás** Iskola
  Bemegyek az iskolába.
  Elmegyek a matekterembe.
  Matekóra.
  Elmegyek a magyarterembe.
  Magyaróra.
  Elmegyek az énekterembe.
  Énekóra.
  Elmegyek a tornaterembe.
  Tornaóra.
  Elmegyek a számítástechnika-terembe.

```
    Számítástechnika-óra.
    Elmegyek a földrajzterembe.
    Földrajzóra.
    Kimegyek az iskolából.
  Eljárás vége

  Eljárás Utazás haza
    Séta a buszmegállóig.
    Ciklus
      Várakozás.
    Amíg nem jön a busz.
    Ciklus vége
    Felszállás.
    Ciklus
      Buszozás.
    Amíg nem jön az otthon megállója.
    Ciklus vége
    Leszállás.
  Eljárás vége

  Eljárás Délutáni dolgok
    Tanulás.
    Szundikálás.
  Eljárás vége

  Eljárás Esti dolgok
    Vacsorázás.
    Internetezés.
    Lefekvés.
  Eljárás vége
```

Ez a példa segíteni fog abban, hogy a hetirended algoritmusát le tudd írni.

Biztosan tanultál már a függvényekről matematikából. Pontosan úgy működnek a programozás világában, mint ahogy a matematikában. Fontos tulajdonsága, hogy a függvénynek van egy visszatérési értéke. Pl. ha $f(x)=x^2$, akkor *f(5) visszatérési értéke 25*.

Algoritmus szempontjából, az írásmódot tekintve, nagyon fog hasonlítani az eljárásra. A naprendes példánál maradva: az órát lehet függvényként használni. Nézzünk egy példát, írjunk egy „Szundikál” nevű eljárást:

```
Eljárás Szundikál
   Ciklus amíg Óra≠17:00
     Alvás.
   Ciklus vége
 Eljárás vége

Függvény Óra: Egész
   Óra:=Pontos Idő
 Függvény vége
```

Magyarra lefordítva: addig alszol, ameddig nem csörög a vekker 17 órakor. Nagyon fontos, hogy mikor függvényt írsz, **legkésőbb az utolsó sorában értéket kell adnod neki!**

Az előző példa Óra függvénye igen egyszerű volt, egyszerűbb, mint az előtte példaként emlegetett $f(x)=x^2$, ugyanis utóbbinak volt egy paramétere, míg a másiknak nem. A paraméternek nagyon fontos a szerepe, ugyanis ettől függ a függvények az értéke. Az f(5) és az f(3) más értéket ad (előbbi 25-öt, utóbbi 9-et). Algoritmusainkban az eljárások és a függvények ugyanígy paraméterezhetőek. Első példánkban nézzük meg, hogyan lehetne algoritmikus nyelven megfogalmazni az $f(x)=x^2$ függvényt:

```
Függvény f(x: Egész):Egész
   f:=x^2
 Függvény vége
```

Az f függvény kap egy bemeneti paramétert, majd visszaadja ennek a paraméternek a négyzetét. Pontosan úgy viselkedik, ahogy megszokhattad matematika órán!

Vajon mit jelenthet az „x: Egész”. **Így jelöljük azt, hogy *az x változó Egész típusú***, azaz, az *x* változó csak egész szám lehet.

Emlékszel még a fejezet elejére, mikor az utcán való átkelést írtuk le algoritmikus nyelven? Tegyük fel, olyan környéken laksz, ahol rengeteg nyomógombos jelzőlámpás kereszteződés van, ezért, érdemes lenne külön eljárást fabrikálni belőle:

```
Eljárás Átkelés(Változó Gomb: Nyomógombok, Lámpa: Jelzőlámpák)
   Megállsz.
   Ha Gomb≠Benyomva akkor
         Megnyomod a Gombot. (Ennek eredménye: a Gomb be van nyomva.)
   Ciklus amíg Lámpa≠zöld
     Vársz.
   Ciklus vége
   Átmész az út túloldalára.
 Eljárás vége
```

Mit is csinálsz? Megállsz a kereszteződésben. Ha még nincs megnyomva a gomb, megnyomod, majd vársz, míg zöld nem lesz a lámpa. Nézd csak meg, elöl tesztelős ciklust használtunk, tehát, abban az esetben nem kell várnod egy pillanatot sem, ha már megérkezésedkor zöld volt a lámpa. Legvégül átmész az úton.

Az eljárásnak két paramétere volt. Az első előtt (*Gomb*) új kulcsszót olvastál: *Változó*. Mire utal ez? Mindjárt választ kapsz rá. Amikor paraméterezünk egy eljárást, akkor három fajta paramétert kaphat. Vannak a bemeneti paraméterek (pl. ilyen az x az $f(x)=x^2$ függvénynél), ahol az eljárás vagy a függvény ***csak felhasználja, de nem változtatja meg*** azokat. A kimeneti paraméterek attól különböznek ettől, hogy a függvény vagy az eljárás ***megváltoztatja*** az értéküket, ezért tettük a második példában a paraméter neve elé a *Változó* kulcsszót. A paraméterek harmadik fajtája az ún. ki-/bemeneti paraméterek, amelyek rendelkeznek mindkét tulajdonsággal: ***felhasználja és megváltoztatja értéküket.***

Elérkeztünk az első fejezet legutolsó, s talán legnehezebb témájához, ahol az eddig tanultakat egybe gyúrjuk. Ahhoz, hogy megértsd a következő fejezetben bemutatandó fejlesztőkörnyezet működését (ebben fogjuk elkészíteni programjainkat, lefuttatni az algoritmusainkat), szükség lesz ahhoz, hogy megismerkedj az *osztállyal* és az *objektummal*. Nem olyan ördöngös ez, mint amilyennek hangzik.

Maga az *osztály* nagyon hasonlít egy halmazhoz. Benne levő dolgok, azaz *objektumok* tulajdonságait határozza meg. Így például osztálynak tekinthetőek az emberek, a könyvek, az autók… stb. Írjuk fel pl. az Emberek és a Könyvek osztályát![18]

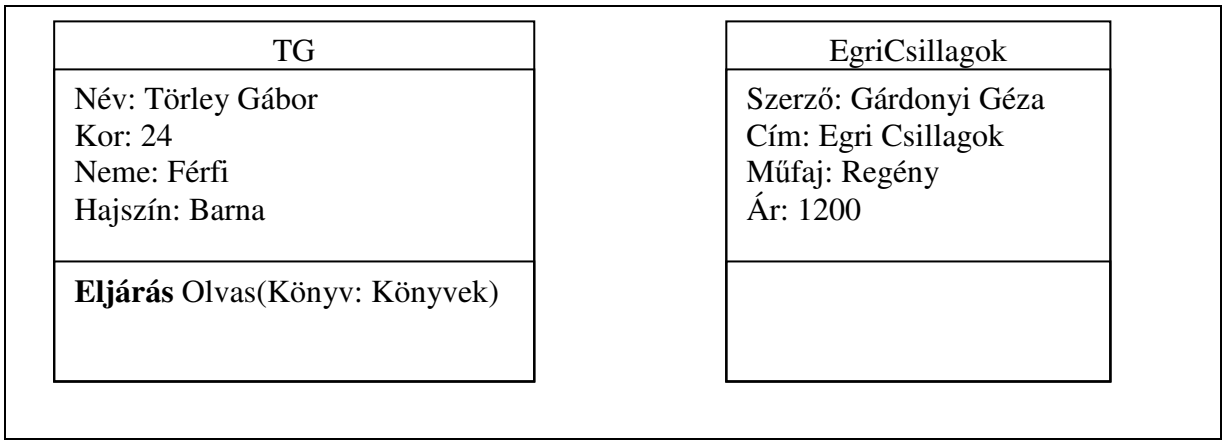

Minden embernek és könyvnek vannak őt meghatározó tulajdonságai. Embernek a neve, kora, haja színe; egy könyvnek a szerzője, műfaja. Ezek a tulajdonságok minden egyes osztályba tartozó objektumra jellemzőek.

Egy osztályba nemcsak tulajdonságokat tudunk beírni, ugyanúgy, mint ahogy általában egy emberről nem csak a nevét, haja színét ismerjük, hanem azt is, hogy milyen tevékenységeket szokott végezni. Például, minden ember szokott könyvet olvasni. Bővítsük ki az *Emberek* osztályt:

[18] **[BB]**


Az Emberek osztály *tulajdonság*ai (másként: *attribútum*ai) mellett megjelent egy úgynevezett *metódus*, az Olvas nevű.

Most beszéljünk az objektumokról. Az objektumok az osztályba tartozó dolgok. Mint pl. Te vagy én. Mi egy-egy példányai, azaz objektumai vagyunk az *Emberek* osztálynak. Nézzünk egy-egy példát:




Látható, hogy az *EgriCsillagok* nevű objektum minden olyan tulajdonsággal rendelkezik, amivel a *Könyvek* osztály, csak immáron konkrét értékekkel.

Egy kérdés maradt már csak megválaszolatlan: hogyan tudjuk mindezt beleilleszteni az algoritmikus nyelvünkbe. Nézzünk egy lehetséges nyelvbővítési kísérletet, amelyben új páros kulcs-szavak lesznek: TulajdonságDefiníció, MetódusDefiníció, MetódusKifejtés.

```
Osztály Emberek

  TulajdonságDefiníció
    Név: Szöveg
    Kor: Egész
    Neme: Nemek
    Hajszín: Színek
  TulajdonságDefiníció vége
```

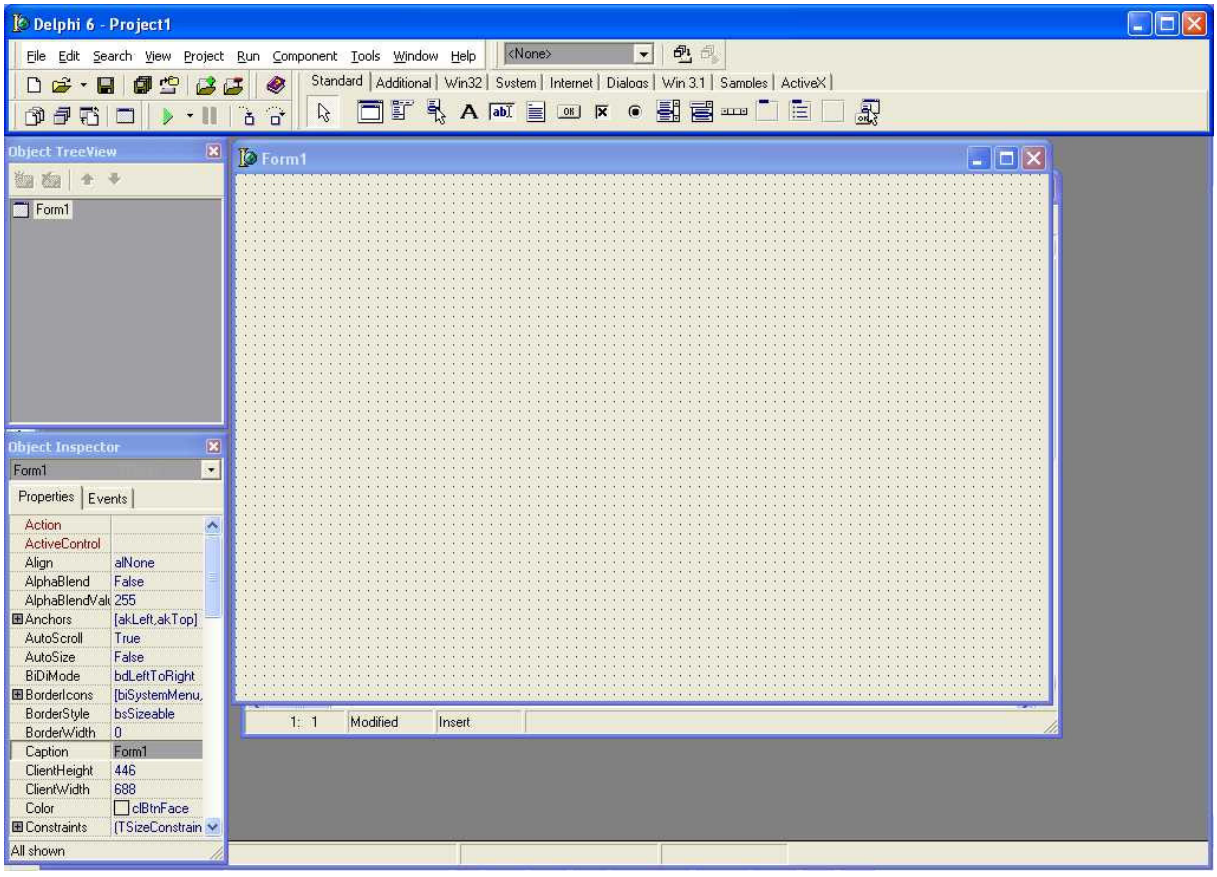
```
MetódusDefiníció
  Eljárás Olvas(Könyv: Könyvek)
MetódusDefiníció vége

MetódusKifejtés
  Eljárás Olvas(Könyv: Könyvek)
    Ciklus
      Olvasok.
    Amíg jól esik
    Ciklus vége
  Eljárás vége
MetódusKifejtés vége
Osztály vége
```

A következő fejezetben megismerkedünk a Borland Delphi 6 fejlesztőkörnyezettel, ahol gyakorlatba fogunk tenni néhány algoritmust.

## 3.2 Ismerkedés a környezettel

Alapértelmezés szerint a programot a Start menü/Borland Delphi 6/Delphi 6 –ra való kattintással tudod elindítani.




*1. ábra: A Delphi, harcra készen.*

Látszólag négy, egyébként öt elkülöníthető ablak jelenik meg. Felül van a menüsor, közülük sokkal találkozhattál már korábban. Alatta találod az ún. komponensek listáját, kategóriánként összegyűjtve. Ahogy haladunk előre a könyvben, egyre több komponenssel ismerkedünk meg, így ezeket nem most mutatnám be.

Mik is ezek a komponensek? Egyfajta építőkövek, pl. ilyen egy gomb, egy címke, egy szerkesztőmező, de ilyen egy menü is. Ezek a komponensek képesek értéket adni, kapni, „kommunikálni” egymással. Rendelkeznek tulajdonságokkal, metódusokkal, hiszen ezek is mind-mind egy-egy objektumot képviselnek. Egy konkrét gomb a Gombokét, egy konkrét címke a Címkékét stb.

A képernyő bal oldalán két kisebb ablakot láthatsz. A felsőben egy faszerkezetet látsz, kezdetben egy elemmel, Form1 néven. Ebben fogod látni a programod által felhasznált komponenseket.

Az alatta levő ablakban az ún. Objektumkezelőt láthatod. Itt tudod leolvasni és módosítani a kijelölt komponens tulajdonságait.

A képernyőn a legnagyobb területet a program felülete, az ún. Form foglalja el. Ide tudod majd feltenni a komponenseket, és programod tervezési fázisában már azt fogod látni, amilyen lesz a kész program külleme.

Az ötödik ablak, az ún. programszerkesztő, a programfelület mögött bújik meg. Itt tudod majd beírni a programkódot, tudniillik a kellő metódusok kódját.

A menürendszert és a komponenseket az elkészített programokon keresztül fogjuk megismerni. Első lépésként írjunk egy nagyon egyszerű programot:

**1. feladat** Írj programot, melyben a szerkesztőmezőbe beírod a neved (pl. Pisti), majd az OK gombra kattintva egy ablak kiírja, hogy „Üdvözöllek, Pisti!”. Használj címke, szerkesztőmező és gomb komponenst a megoldáshoz!

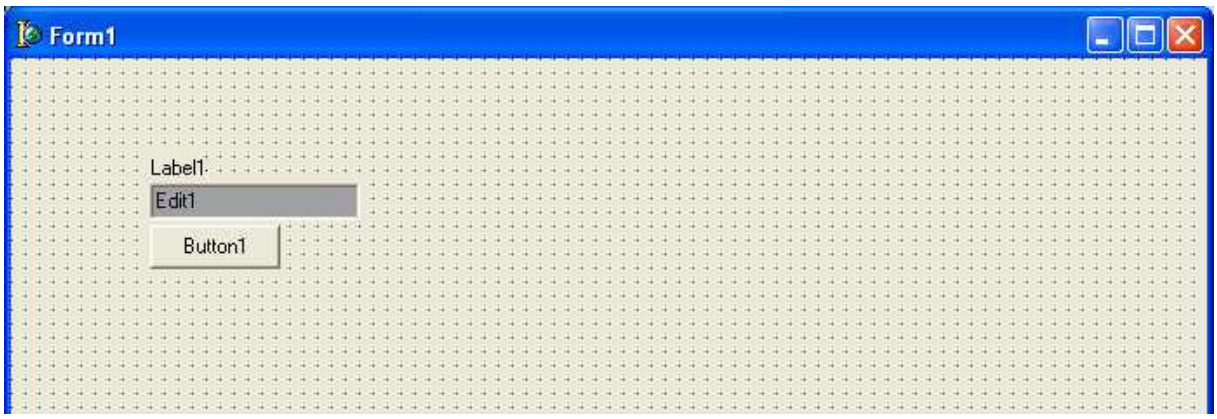
Nem olyan bonyolult ez a feladat, mint amilyennek első látásra tűnik! Első lépésként gondoljuk végig, miket kell felhasználnunk: kell egy szerkesztőmező, amibe a felhasználó beírhatja a saját nevét, és egy gomb. Ezen kívül érdemes lesz még használni egy címkét, amivel tudtára adhatjuk a felhasználónak, mire is való a szerkesztőmező. Általában igaz, hogy ha egy feladatnál szerkesztőmezőt kell használni, címkét is érdemes lesz feltenni a programfelületre.

Első lépésként rakjuk fel a kívánt komponenseket!


*2. ábra: A Menüsor.*

Kattints a Standard fülre, és az itt megjelenő ikonok közül a pirossal bekeretezettekre lesz szükségünk. Sorrendben: címke (label, ejtsd: lébl), szerkesztőmező (edit) és gomb (button, ejtsd: bátn). Kattints rá valamelyikre, majd kattints a programfelületre, és méretezd

addig a komponenst, amíg szükséges. Miután felraktál mindent, a 3. ábrán láthatóhoz hasonló kép fog eléd tárulni.




*3. ábra: Készülő programunk külleme.*


*4. ábra: Objektumkezelő.*

Amikor felpakoltad a komponenseket, láthattad, hogy az Objektumkezelő feletti ablakban a fa több elemmel bővült – pontosan azokkal, amiket felpakoltál.

Eddig minden szép és jó, az egyik fő baj az, hogy nem túl „beszédesek” még a feliratok. Kattints rá a címkére. Láthatod, hogy az Objektumkezelő tartalma változott. Ez az ablak tárolja a kiválasztott objektumnak minden tulajdonságát. Most kettőt ismernénk meg:

Caption (ejtsd: kepsön) – azaz felirat, szöveg típusú. Ezt láthatja a felhasználó kiírva.

Name (ejtsd: néjm) – azaz név, ugyancsak szöveg típusú. Ezen a néven lehet elérni a kódból ezt a komponenst.

A szerkesztőmező esetén a felirat szerepét a text – azaz a szöveg „játssza el”.

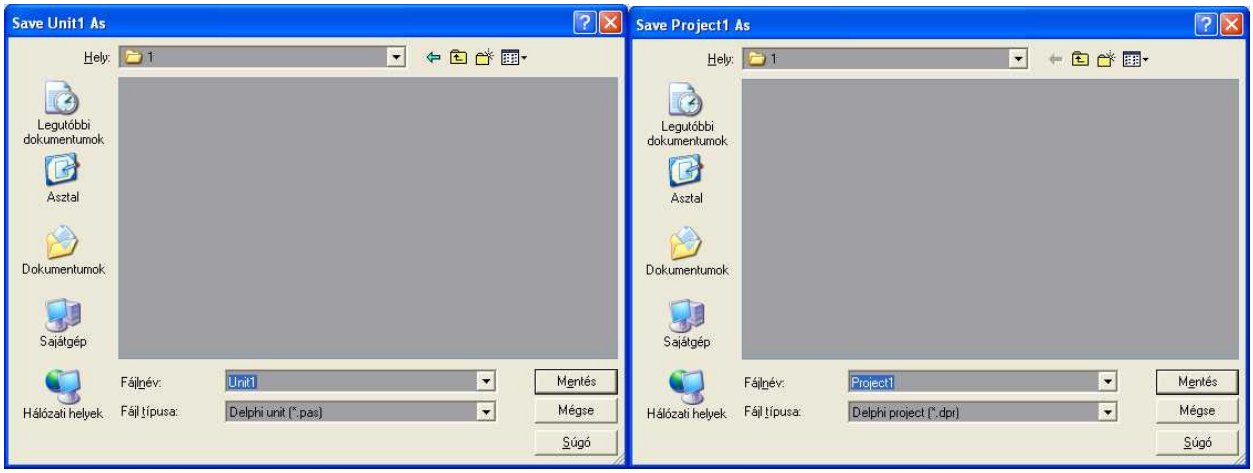
Ha visszaemlékszel ez előző fejezet végére, mikor az osztályról beszéltünk, ott is az objektumoknak különböző tulajdonságaik voltak. Ezeket a tulajdonságokat mutatja meg az Objektumkezelő. Egy ilyen tulajdonság átírása nagyon egyszerű: belekattintasz és átírod.

Egy lehetséges megoldást mutat az 5.ábra.


*5. ábra: A megoldás felé.*

Baloldalon láthatod az objektumok neveit, jobb oldalon a programunk küllemét. Láthatod, hogy egy változó neve két részből áll. Az első része, az úgynevezett *prefix* utal a változó „típusára”, pontosabban, jelen esetben arra, hogy milyen komponens is az. Charles Simonyi, magyar származású programozó vezette ezt be.[19] Használata nem kötelező, de sokat segít a kód olvashatóbbá tételében. Ebben a példában a „c_” a címkére, a „g_” a gombra az „sz_” pedig a szerkesztőmezőre utal. Tehát a c_Név objektum a „Hogy hívnak?” kérdés (címke) megjelenítéséért felelős. A g_Mehet pedig a programhasználó „gomb-lenyomását” közvetíti a program felé. Az sz_Név objektum feladatát próbáld kitalálni magad!

Ezen a ponton mentsük el a programunkat! Furcsa módon, két mentést kér tőlünk a környezet:




*6. ábra: Mentés*

Névnek ismét válasszunk valami beszédes nevet. Unit1 helyett (amit felkínál a rendszer) pl. u_elso (ahol az „u_” a unitra (ejtsd: junit) utal); Project1 (ejtsd: prodzsekt) helyett pl. Elso_prg.

[19] **[FSz]**

Vajon miért kell „így”, több fájlba menteni? Mi az a „project” és mi az a „unit”?

Egy példán keresztül meg fogod érteni. Tegyük fel, hogy osztálykirándulást szerveztek (ez a projekt, a „nagy feladat”). Több része van egy ilyen feladatnak. Valaki megszervezi az utazást, valaki a szállást, míg más valaki a programokért fog felelni. Ezekből a részekből, „egységekből”, azaz unitokból tevődik össze a teljes kirándulás. A Delphivel nagy programozócsapatok (is) szoktak dolgozni, több ember egy projekten. Mindegyik írja a saját unitját, a saját részét, majd egyszer csak összegyűlnek, és összerakják, és akkor lesz kész a teljes program.

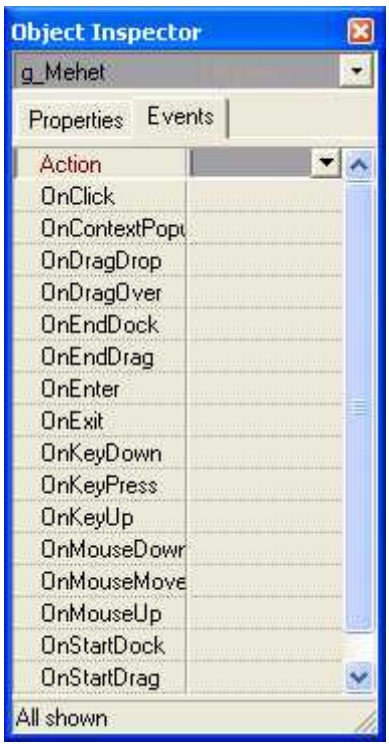
Látványos ponthoz érkeztünk első programunkkal kapcsolatban. Lefordítjuk és lefuttatjuk. Kattints a menüsoron levő zöld „lejátszás gombra”!


***7. ábra: Fordítsuk le a programot!***

Emlékszel még az előző fejezetre? Ott is beszéltünk röviden a fordításról. Ekkor fordítja a környezet a számítógépnek is érthető „nyelvre” azt, amit alkottunk. Amikor ez megtörténik, elkezd futni a program: megjelenik egy ablak. Pont olyan, amilyet megterveztünk. Pedig, még nem is csináltunk semmit, csak „feldobtunk” néhány komponenst, mégis van egy futó programunk! Próbáld ki! Írd be a neved a szövegmezőbe! Kattints a gombra! Nem csinál semmit, igaz? Persze, mert még nem mondtuk meg, pontosabban nem írtuk meg azt a részt, hogy a gombra való kattintást követően mi történjék.

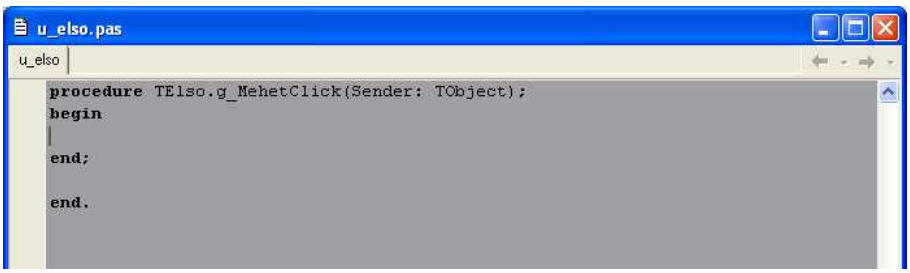
Ehhez meg kell ismerkednünk egy új fogalommal: az *eseménykezelés*sel. Zárd be a futó programot, majd kattints rá a programfelületen a gombra. Láthatod, hogy az Objektumkezelőnek két füle van: Properties (ejtsd: propertíz) – Tulajdonságok és Events (ejtsd: ívüntsz) – Események. Kattints az utóbbira!

Esemény lehet bármi. Egérkattintás, egérmozgás, egy program futása befejeződött, leütöttél egy billentyűt. Úgy lehetne ezt elképzelni, mint egy pókot a hálójával. A pók is várja a rovarokat, hogy beleszálljanak hálójába. Amint egy belerepül, megtörténik az esemény, a pók lemegy, és behálózza. Mikor előbb próbáltál a gombra kattintani, azért nem történt semmi, mert nem volt „kifeszítve a háló". A „hálót" azzal feszítjük ki, hogy megmondjuk a környezetnek, mit tegyen, ha egy esemény bekövetkezik. Nekünk most az OnClick (ejtsd: onklikk) eseményre lesz szükségünk. Ha ide írunk valamit, akkor fog lefutni, ha rákattintunk a gombra. Fontos megjegyezni, hogy több objektumnak is lehet OnClick eseménye, nem csak a gombnak.




*8. ábra: A Gomb eseményei.*

Kattints kétszer az OnClick melletti sorba. Automatikusan átrak a környezet a programszerkesztőbe. Hirtelen egy eljárás belsejében találjuk magunkat:

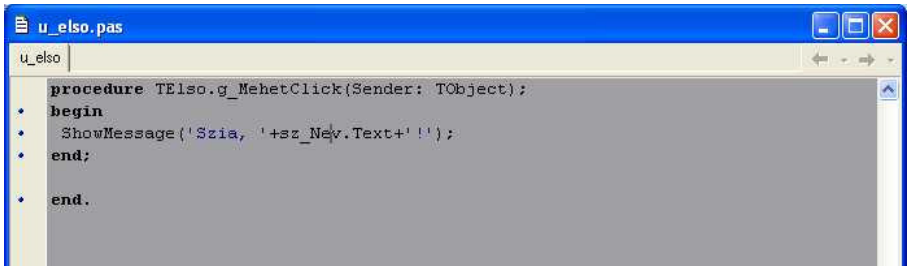



*9. ábra: Az OnClick esemény „belseje".*

Az előző fejezet második felében tanultunk az eljárásokról. Delphiben minden esemény lekezelése egy-egy ilyen eljárás „keretein belül" hajtódik végre. Nagyon hasonlít ahhoz, amit az algoritmusnál írtunk, csak az „eljárás" szó helyett a környezetben a „procedure" (magyarul eljárás, folyamat) szót kell majd használni. Egy darab sort fogunk mindösszesen beírni. Ahhoz, hogy egyszerűen kiírassunk valamit a képernyőre, a *ShowMessage* (ejtsd: sómeszidzs) parancsot kell használni. Egyetlen paramétere van a parancsnak, az a szöveg, amit szeretnénk kiíratni. Hol lesz ez a szöveg tárolva? A szövegmezőben, amit így tudunk elérni: *sz_nev.Text*. Általánosságban is így fogunk hivatkozni az objektum akármelyik tulajdonságára vagy metódusára: *objektumnév.tulajdonság*.

Tehát egy szöveg típusú változót kell majd kiírnunk, illetve a ShowMessage parancs is szöveg típusú változót vár paraméterül. Pontosan mit is szeretnénk kiírni? Például azt, hogy „Szia, Pisti!". Ebből csak a „Pisti"-t tartalmazza a szövegmező. Mit lehet tenni?

Ahhoz, hogy megoldjuk ezt a feladatot, meg kell, hogy ismerkedjünk a szövegek „összeadásával”, az ún. *konkatenáció*val. Nagyon egyszerű: „motor”+„vonat”=„motorvonat”! A szövegünk, amit szeretnénk megjeleníteni, fel lehet osztani három részre: „Szia,”+„Pisti”+„!”. Innen már látszik a végső megoldás:




*10. ábra: A hiányzó láncszem*

Két fontos dolgot láthatunk: Ha direktben szeretnénk szöveget kiíratni Delphiben, akkor ’ ’-k közé kell tenni! Változót egyszerűen tudunk kiíratni, a nevével. A másik, s talán a legfontosabb dolog: Delphiben minden sor után pontosvesszőt „;” kell tenni (vannak kivételek, de erről majd később olvashatsz).


*11. ábra: Működik!*

Ha mindezt beírtad, mentsd el, majd fordítsd le a programot. Csodák-csodájára működni fog!

## 3.3 További komponensek: rádiógomb és jelölőnégyzet; az elágazás

A következő feladatban két másik, sűrűbben használt komponenssel fogunk megismerkedni:

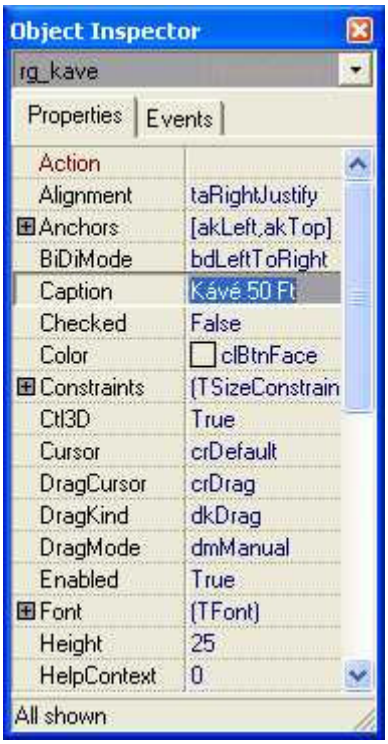
**2. feladat:** Készíts egy kávéautomatát szimuláló programot, amely a választásodtól függően kiszámolja, mennyi pénzt kellene bedobnod. Némely termékhez lehet „extra” dolgokat választani. Pl.: Kávénál: lehet hosszú kávét inni, extra cukrot vagy tejet kérni. De teánál csak extra tejet vagy cukrot és forró csokinál csak extra cukrot lehessen kérni. Legyen olyan ital is, amihez nem lehet semmi extrát kérni. Ezek a plusz dolgok, természetesen, plusz pénzbe kerülnek, vedd figyelembe az ár kiszámításánál! Felhasználandó komponensek: címke, rádiógomb, jelölőnégyzet, gomb.


*13. ábra: Jelölőnégyzet és Rádiógomb. Rájuk lesz szükség.*

Először ismerkedjünk meg a két új komponens tulajdonságaival. A Rádiógombot (radiobutton, ejtsd: rédióbátn) többedmagával szokták használni. Nevét a régi rádiók frekvenciasáv-választó gombjáról kapta. Ezeknek ez a tulajdonságuk, hogy a gombsorból ***csak***

***egy*** lehet benyomva, a többi nem. Megoldásunk során az italok kiválasztásához fogjuk használni, hiszen, mindig egy fajta italt szeretnénk inni, és tud kiadni az automata.

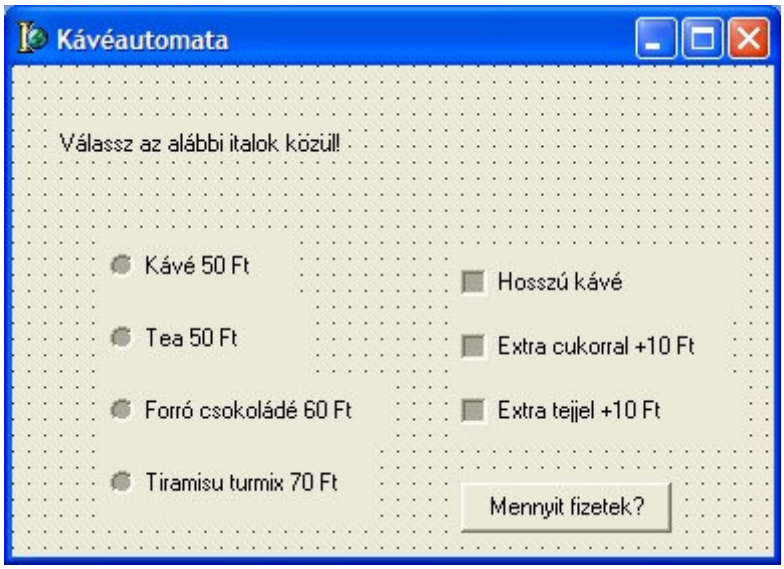
A jelölőnégyzet (chechbox, ejtsd: csekbox) esetében nincsen „létszámbeli” megkötés. Vagy be van „ikszelve”, vagy nincs.

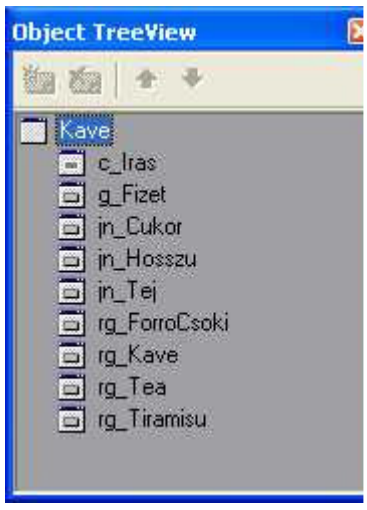



*13. ábra: A Rádiógomb tulajdonságai*

Használatában nagyon hasonlít a két komponens egymáshoz. Mindkettőjüknél a feliratot és a nevet az előző feladatban megismert módon kell megváltoztatni. Azonban, egy másik tulajdonságra is szeretném felhívni a figyelmet, a *Checked*-re (ejtsd: csekd, magyarul: bejelölve). Ennek a tulajdonságnak két értéke lehet: *True* (ejtsd trú) és *False* (ejtsd: fólsz) (magyarul: igaz vagy hamis). Az ilyen elemeket *logikai típusúak*, mivel csak *két értéket* vehetnek fel. A 15. ábrán láthatod azt az állapotot, mikor az összes komponens a helyére kerül.

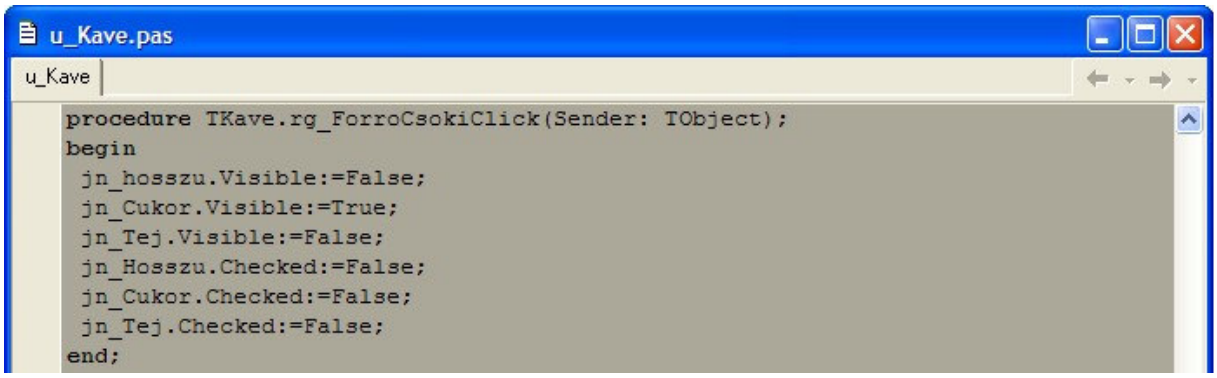

*14. ábra: A Jelölőnégyzet tulajdonságai*




*15. ábra: Ilyen lesz…*

Még nem vagyunk készen. Milyen feladataink vannak még? Meg kell írnunk a „szabályokat”, azaz, milyen italhoz milyen extrát lehet kérni, illetve a pénzkiszámoló eljárást. Ha megnézed valamelyik rádiógomb objektumkezelőjében az eseményeit, találkozhatsz a már „jól ismert” OnClick eseménnyel. Ide lehetne megírni minden rádiógombhoz, hogy mely jelölőnégyzetek jelenjenek meg, s melyek ne. Általában minden komponensnek van egy *visible* (ejtsd: vizibl, azaz látható) névre hallgató tulajdonsága. Ez is logikai típusú, tehát igaz (true) vagy hamis (false) értékeket tud felvenni. Érdemes még kezdőértéket („kezdőállapotot”) adni a jelölőnégyzeteknek, ezek ne legyenek bejelölve, mert így egyszerűbben tudjuk majd az árat kiszámolni. A 16. ábrán láthatsz egy példát, a forró csoki OnClick eseményére:




***16. ábra: A forró csoki OnClick eseménye***

Látható, hogy a jelölőnégyzetek közül csak az Extra cukor fog látszani, és egyik sem lesz bejelölve. Ha ebben az állapotában lefordítod a programot, és lefuttatod, láthatod, hogy attól függően, hogy melyik italt választod, jelennek meg az extrák.

Már csak az ital árát kiszámító eljárást kell megírnunk, ami akkor fog lefutni, mikor a gombra kattintasz. Írjunk hozzá algoritmust!

**Eljárás** g_Fizet_Kattint

**Változó**
Ár: Egész

Ár:=0
**Ha** rg_Kávé=Benyomva **akkor** Ár:=Ár+50
**Ha** rg_Tea=Benyomva **akkor** Ár:=Ár+50
**Ha** rg_ForróCsoki=Benyomva **akkor** Ár:=Ár+60
**Ha** rg_Tiramisu=Benyomva **akkor** Ár:=Ár+70
**Ha** jn_Cukor=Bejelölve **akkor** Ár:=Ár+10
**Ha** jn_Tej=Bejelölve **akkor** Ár:=Ár+10
Ki: Összesen <Ár> Ft-ot kell fizetned!
**Eljárás vége**

Az eljárás nevéből kitűnik, hogy a „g_Fizet” gomb OnClick eseményéhez tartozik. Végig ellenőrzi a rádiógombok és a jelölőnégyzetek állapotát, és ez alapján kiszámolja az

árat. Látszik, hogy csak akkor fog az áron változtatni, ha az adott rádiógomb vagy jelölőnégyzet meg van nyomva/be van jelölve. Érdemes az „Ár”-at lenullázni az elején, mert semmi sem garantálja, hogy helyes érték fog állni a változó helyén, mikor az létrejön. A Delphi nyelvén ekképp néz ki:

u_Kave.pas

u_Kave

```
procedure TKave.g_FizetClick(Sender: TObject);

Var
 Ar: Byte;

begin
 Ar:=0;
 If rg_Kave.Checked Then Ar:=Ar+50;
 If rg_Tea.Checked Then Ar:=Ar+50;
 If rg_ForroCsoki.Checked Then Ar:=Ar+60;
 If rg_Tiramisu.Checked Then Ar:=Ar+70;
 If jn_Cukor.Checked Then Ar:=Ar+10;
 If jn_Tej.Checked Then Ar:=Ar+10;
 ShowMessage('Összesen '+IntToStr(Ar)+' Ft-ot kell fizetned!');
end;
```

*17. ábra: A „g_Fizet” gomb OnClick eseményének eljárása*

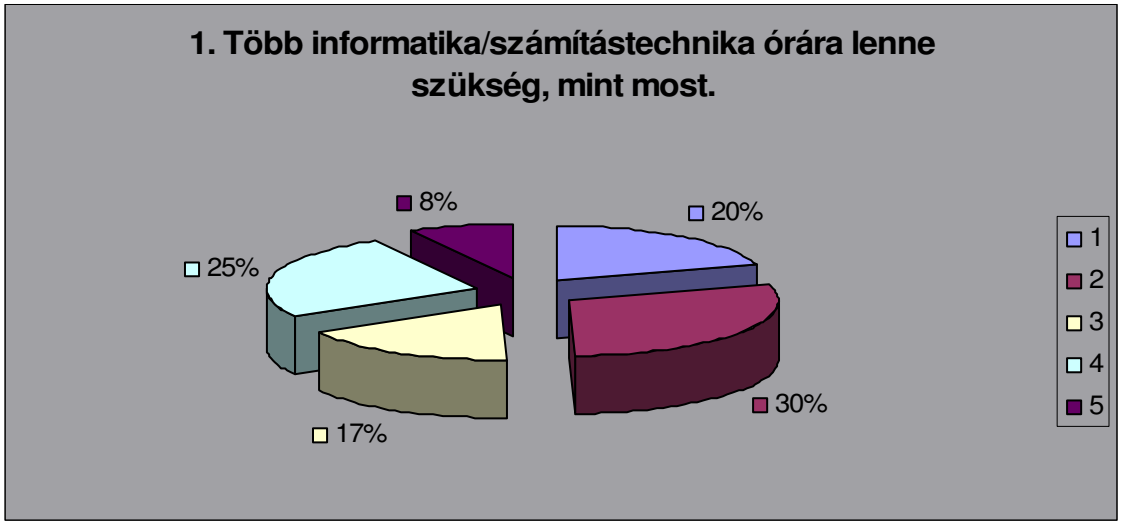
Az ár kiíratásához fűznék egy megjegyzést: Azt megtanultuk az előző feladatnál, hogy a *ShowMessage* eljárás csak szöveg típusú elemet tud kiírni. Az „Ar” viszont szám, ezért kell, hogy felhasználjuk az *IntToStr* függvényt, aminek bemeneti paramétere egy szám, és eredménye ugyanaz a szám – csak szövegként. Tehát átalakítja a számot számjegyekből álló szöveggé.

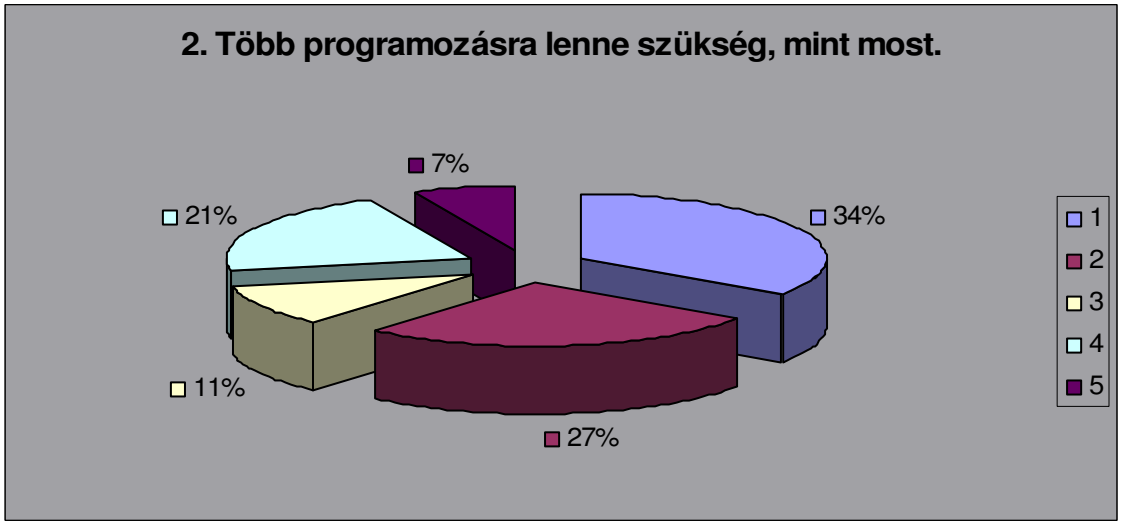
Ha beírod, és ezek után lefuttatod a programot, majd a gombra kattintasz, ki fogja írni a választott ital árát.

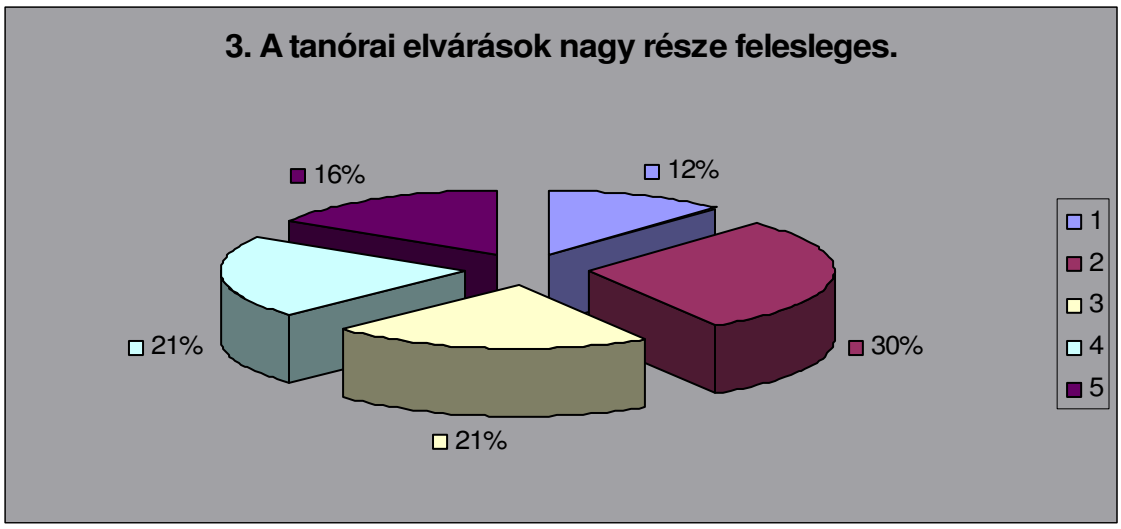
Ebben a feladatban arra is fény derült, hogy a Delphi nyelvén hogy lehet elágazást írni, de erről, és a többi korábban megismert vezérlési szerkezet „Delphis” megfelelőjéről a következő fejezetben olvashatsz.

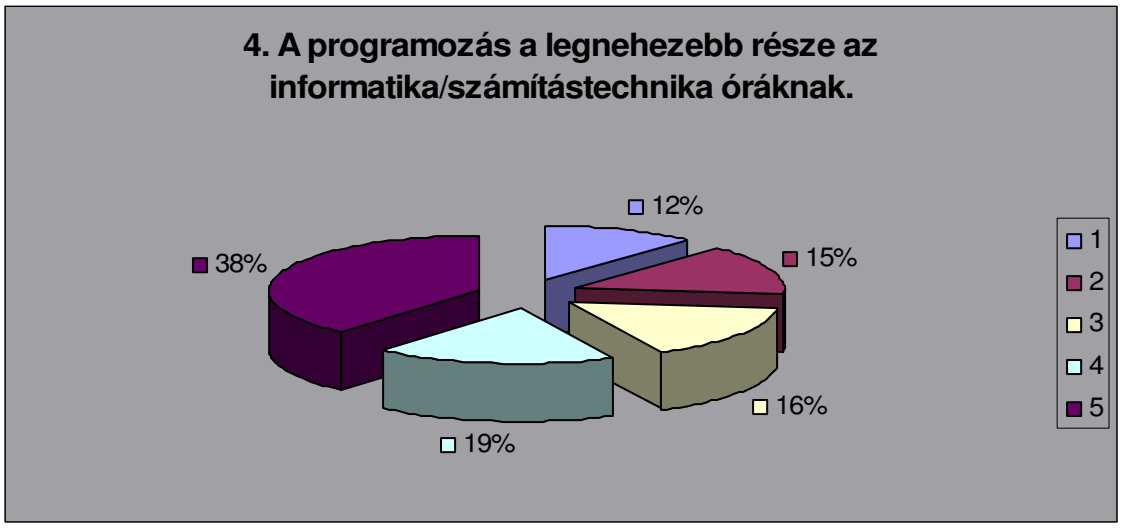
# *Függelék*

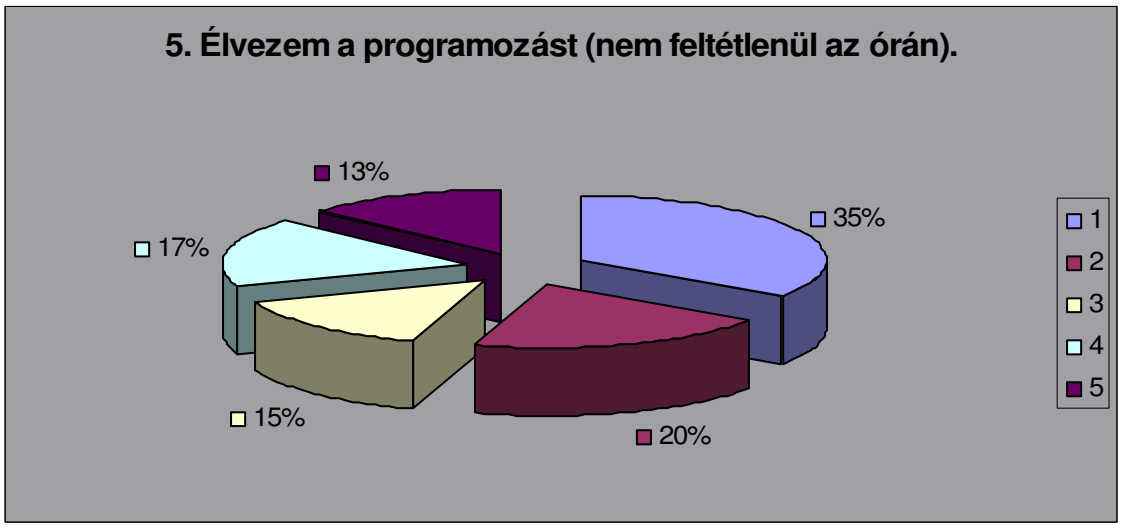
## 1.1 A kérdőív válaszainak eloszlása a teljes mintát tekintve

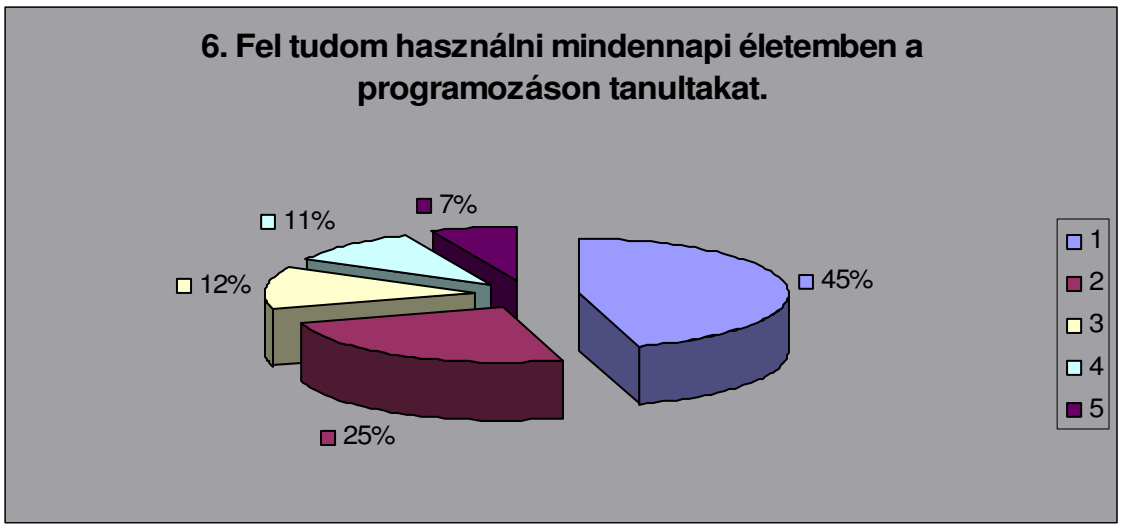


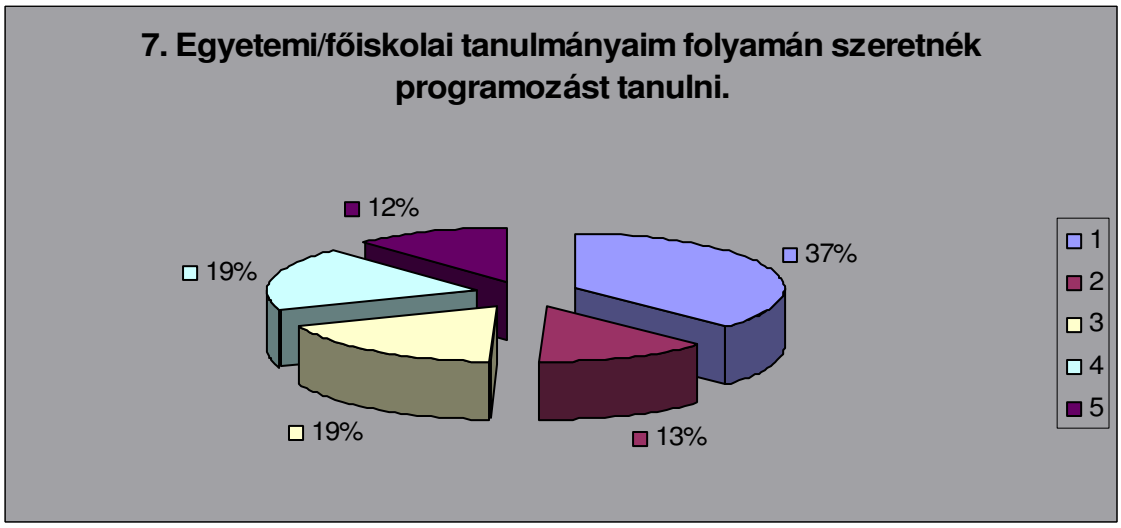



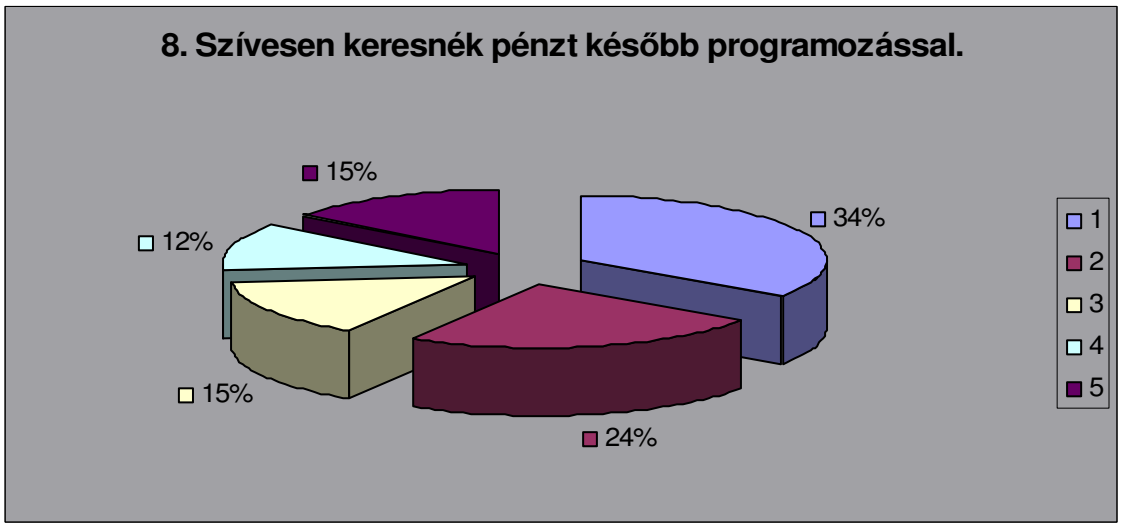

3. A tanórai elvárások nagy része felesleges.
16%
12%
21%
30%
21%
1
2
3
4
5

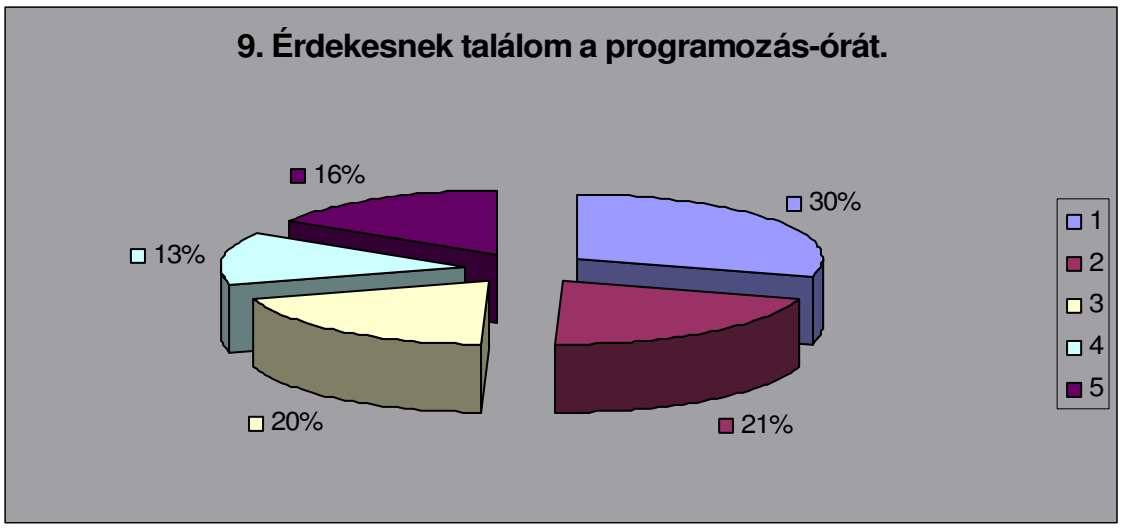

4. A programozás a legnehezebb része az informatika/számítástechnika óráknak.
12%
38%
15%
16%
19%
1
2
3
4
5

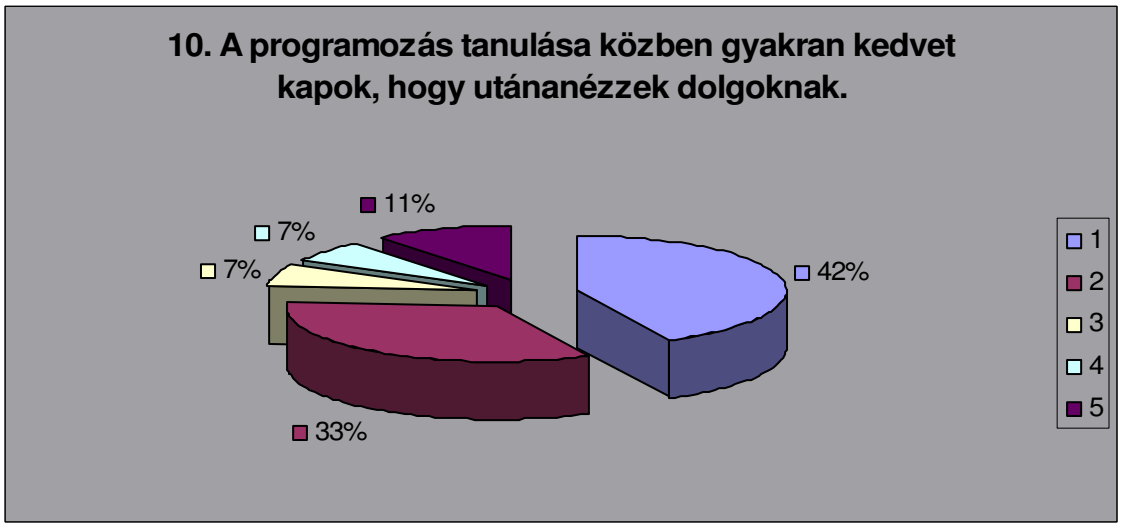

5. Élvezem a programozást (nem feltétlenül az órán).
13%
35%
17%
15%
20%
1
2
3
4
5

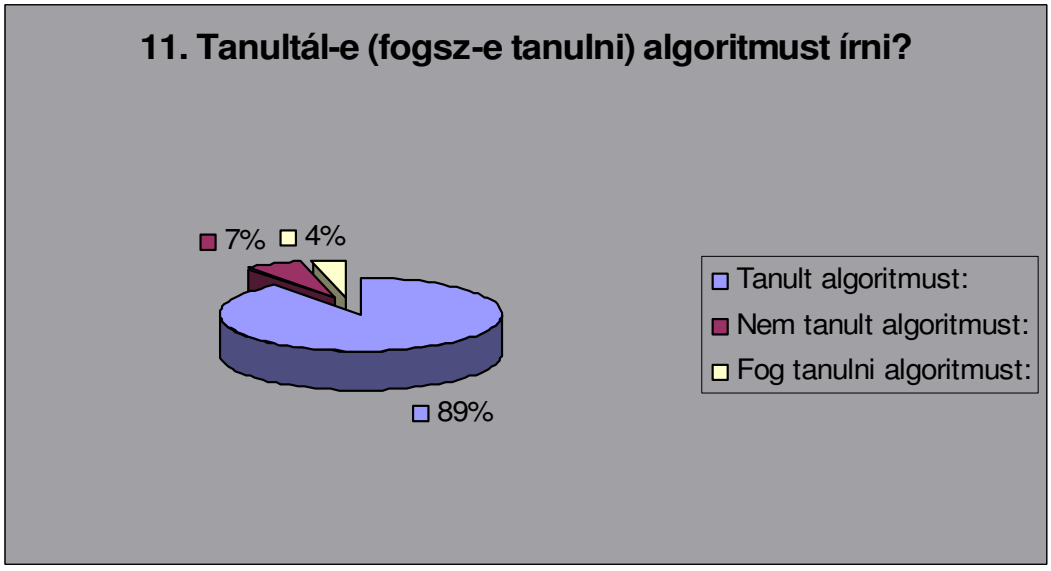

6. Fel tudom használni mindennapi életemben a programozáson tanultakat.
7%
11%
12%
45%
25%
1
2
3
4
5

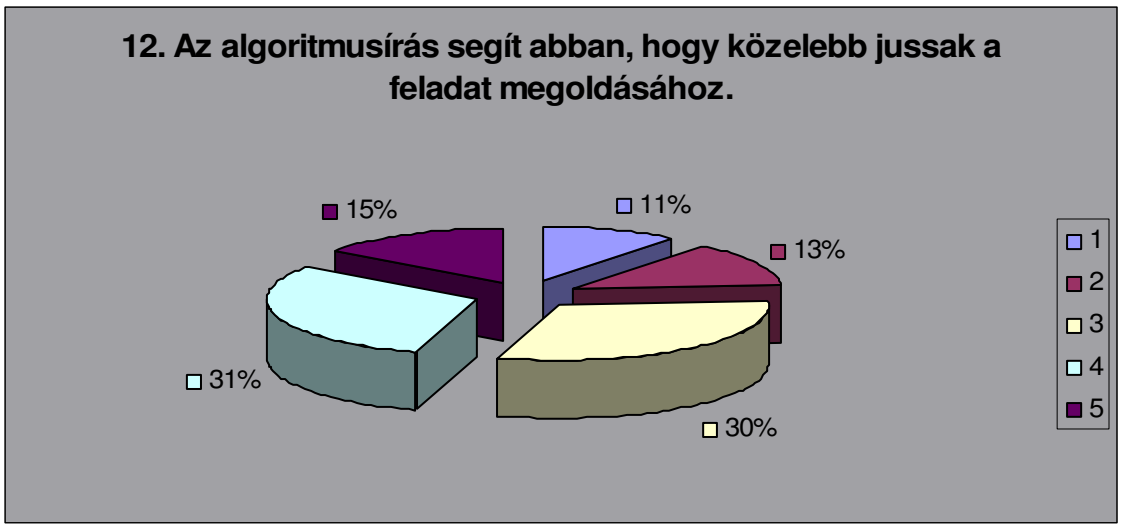

7. Egyetemi/főiskolai tanulmányaim folyamán szeretnék programozást tanulni.
12%
19%
37%
19%
13%
1
2
3
4
5

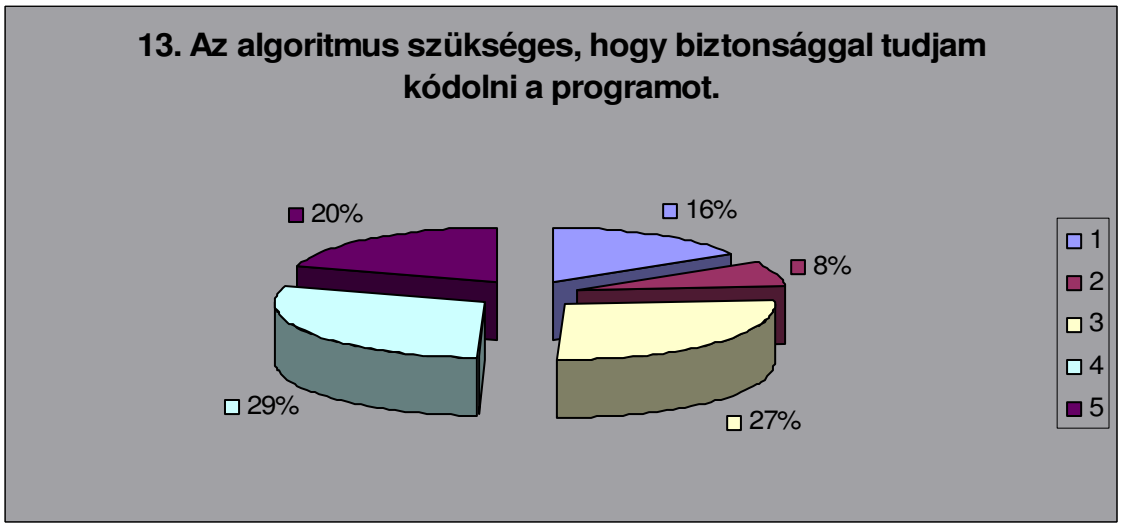

8. Szívesen keresnék pénzt később programozással.
15%
12%
34%
15%
24%
1
2
3
4
5

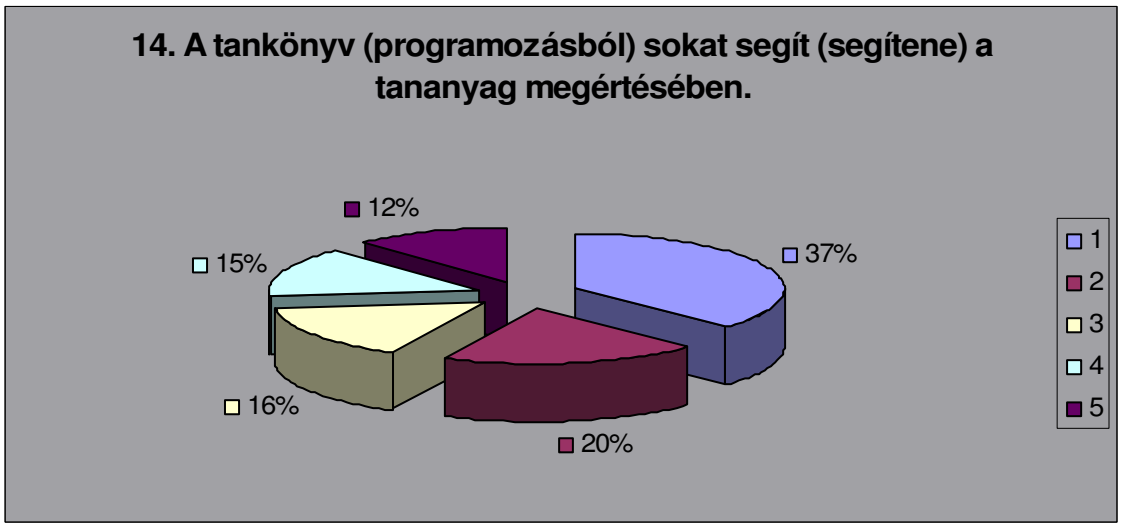

9. Érdekesnek találom a programozás-órát.
16%
13%
20%
30%
21%
1
2
3
4
5

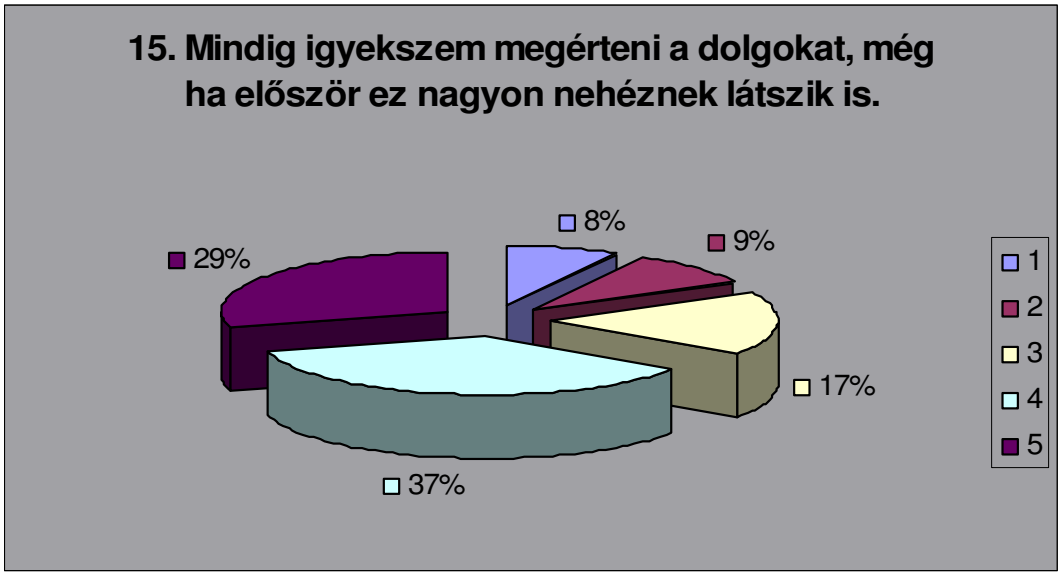

10. A programozás tanulása közben gyakran kedvet kapok, hogy utánanézzek dolgoknak.
11%
7%
7%
42%
33%
1
2
3
4
5

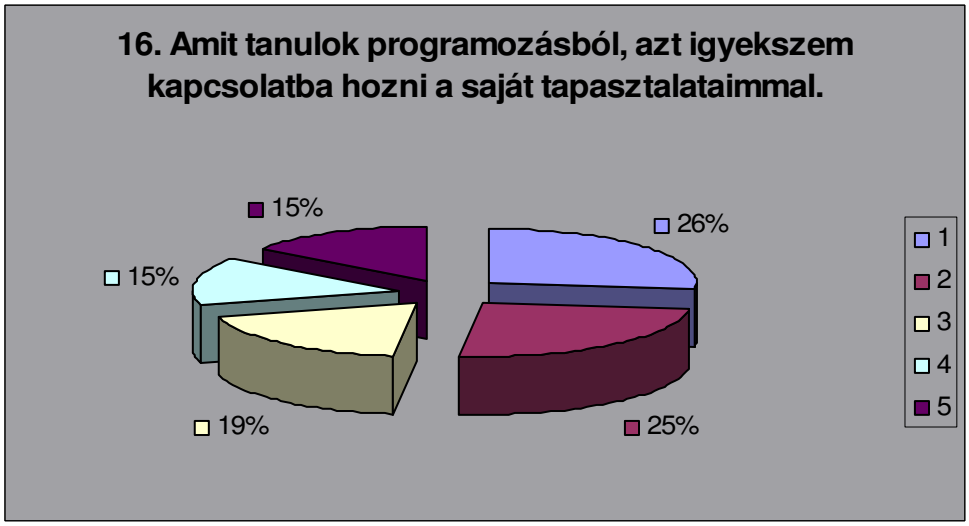

11. Tanultál-e (fogsz-e tanulni) algoritmust írni?
7%
4%
89%
Tanult algoritmust:
Nem tanult algoritmust:
Fog tanulni algoritmust:

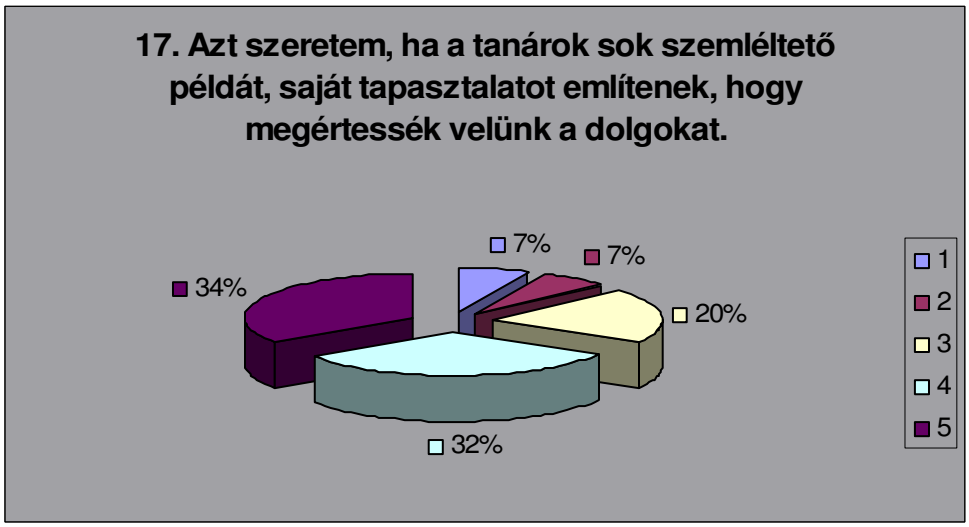

12. Az algoritmusírás segít abban, hogy közelebb jussak a feladat megoldásához.
15%
11%
13%
31%
30%
1
2
3
4
5

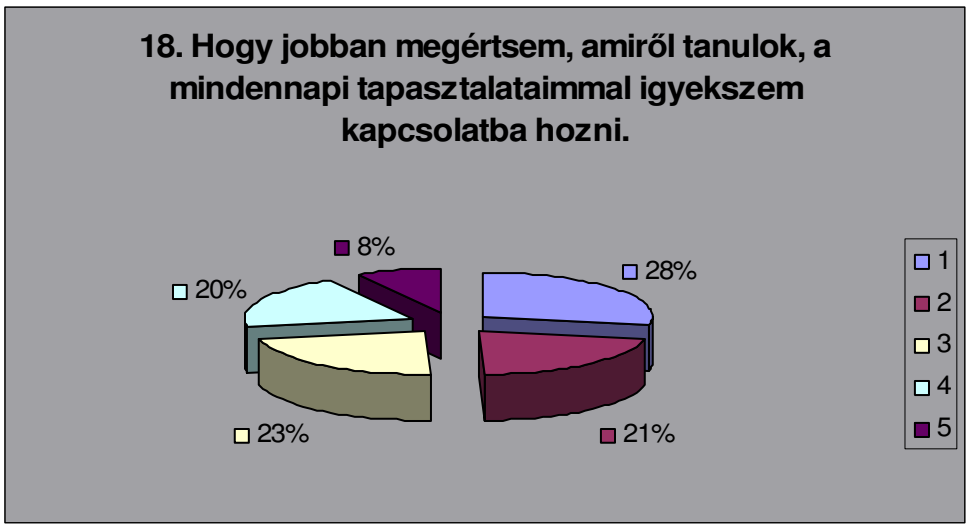

13. Az algoritmus szükséges, hogy biztonsággal tudjam kódolni a programot.
20%
16%
8%
29%
27%
1
2
3
4
5

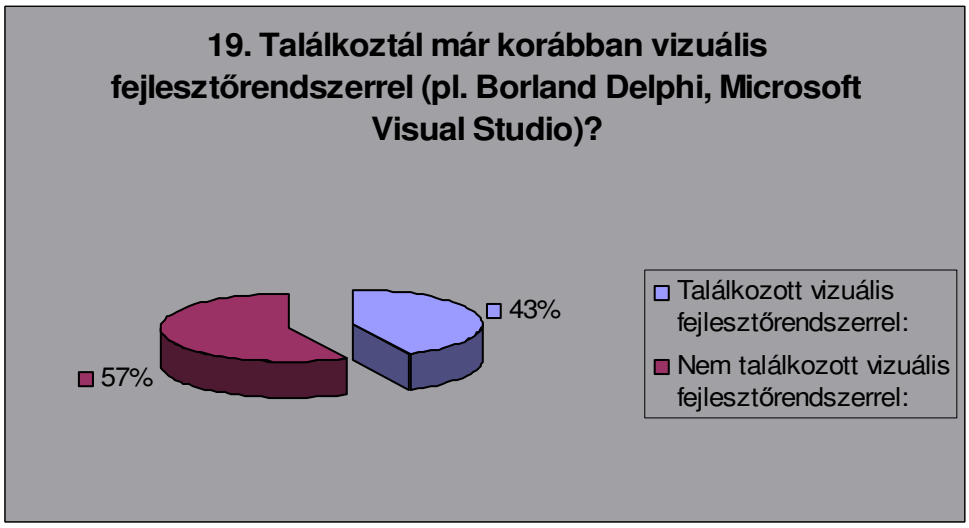

14. A tankönyv (programozásból) sokat segít (segítene) a tananyag megértésében.
12%
15%
37%
16%
20%
1
2
3
4
5

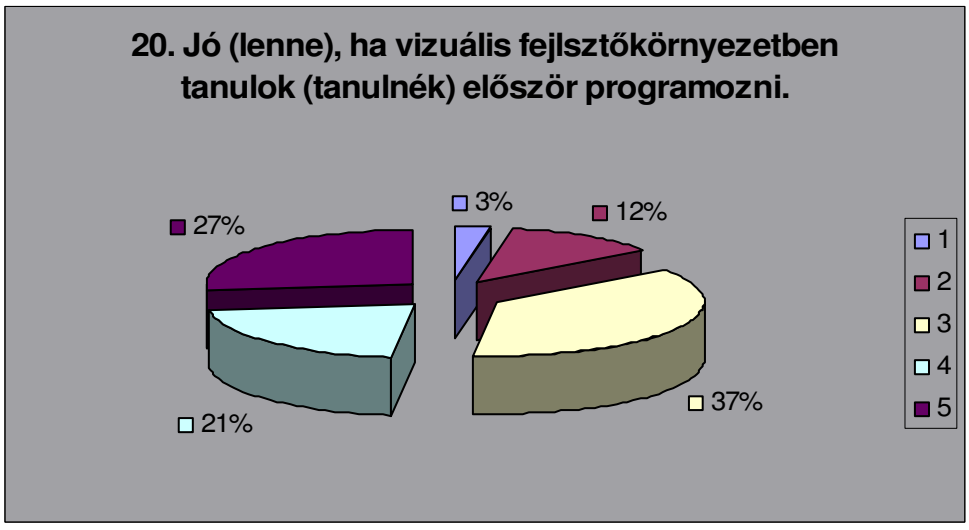

15. Mindig igyekszem megérteni a dolgokat, még ha először ez nagyon nehéznek látszik is.
8%
9%
29%
17%
37%
1
2
3
4
5

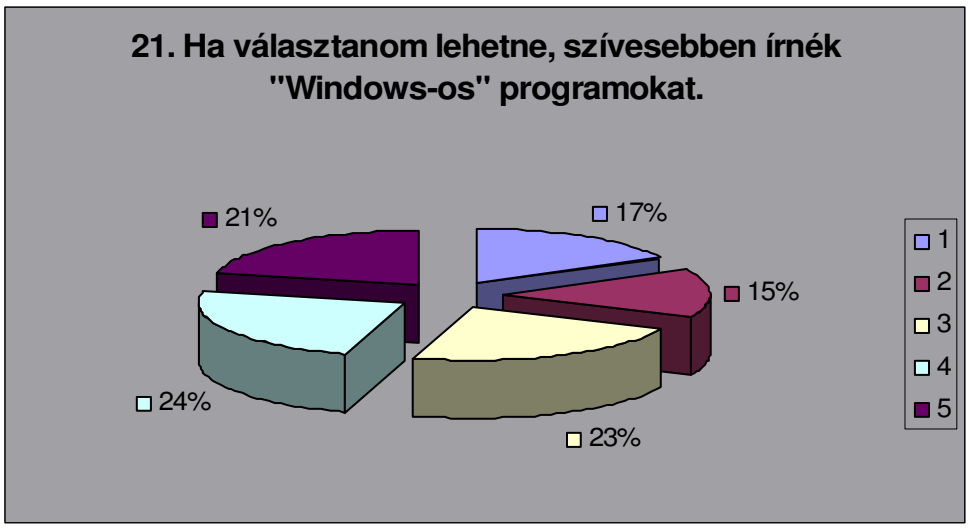

16. Amit tanulok programozásból, azt igyekszem kapcsolatba hozni a saját tapasztalataimmal.
15%
26%
15%
19%
25%
1
2
3
4
5

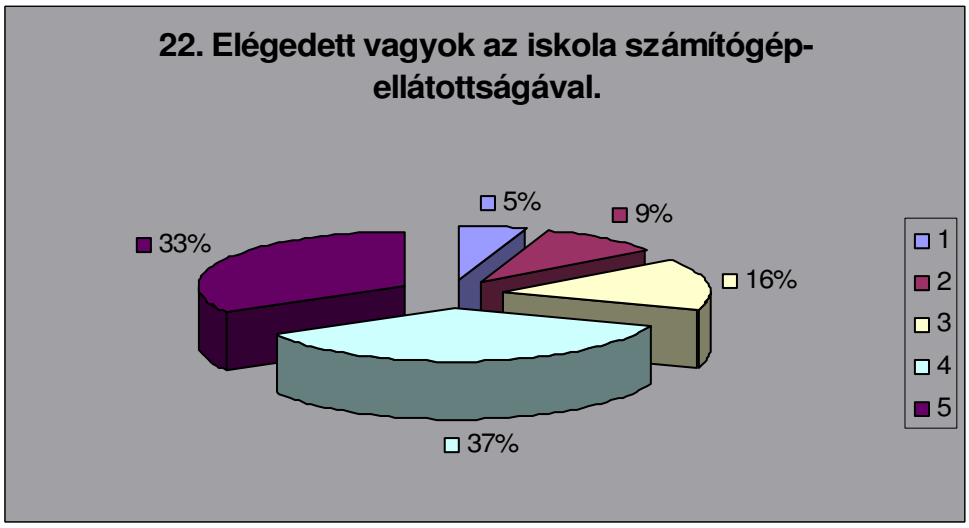

17. Azt szeretem, ha a tanárok sok szemléltető példát, saját tapasztalatot említenek, hogy megértessék velünk a dolgokat.
7%
7%
34%
20%
32%
1
2
3
4
5

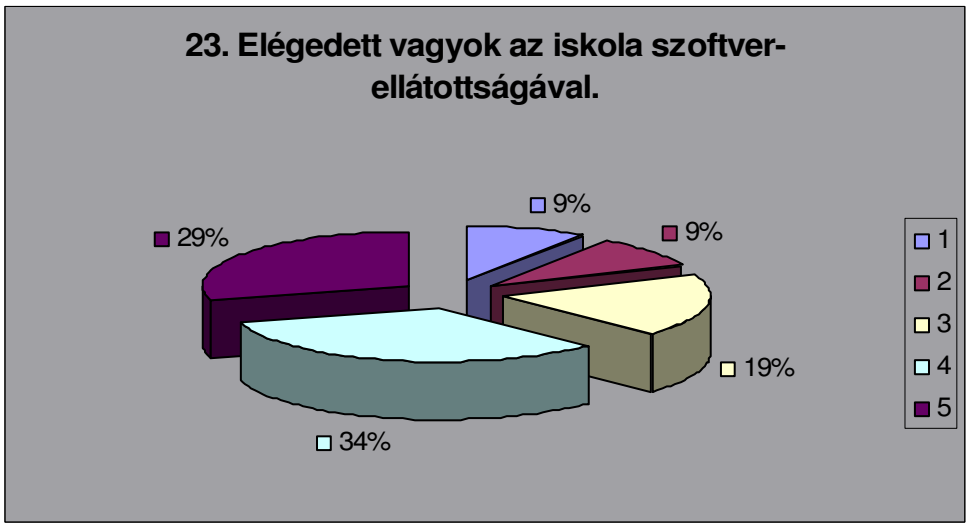

18. Hogy jobban megértsem, amiről tanulok, a mindennapi tapasztalataimmal igyekszem kapcsolatba hozni.
8%
28%
20%
23%
21%
1
2
3
4
5

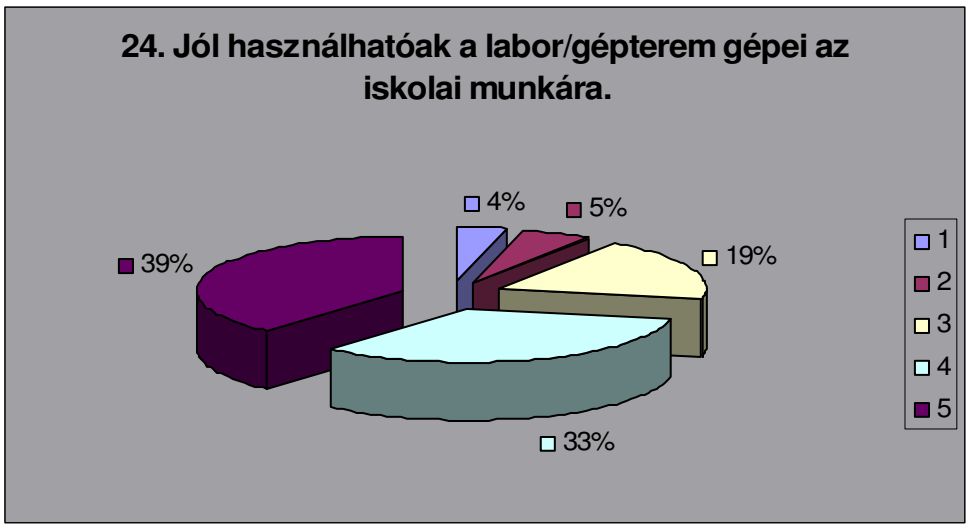

19. Találkoztál már korábban vizuális fejlesztőrendszerrel (pl. Borland Delphi, Microsoft Visual Studio)?
43%
57%
Találkozott vizuális fejlesztőrendszerrel:
Nem találkozott vizuális fejlesztőrendszerrel:

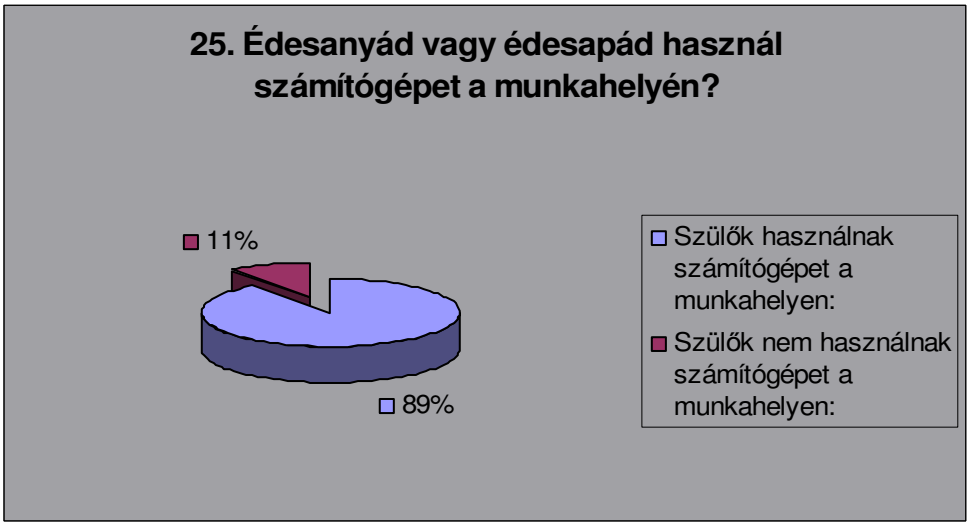

20. Jó (lenne), ha vizuális fejlsztőkörnyezetben tanulok (tanulnék) először programozni.
3%
12%
27%
21%
37%
1
2
3
4
5

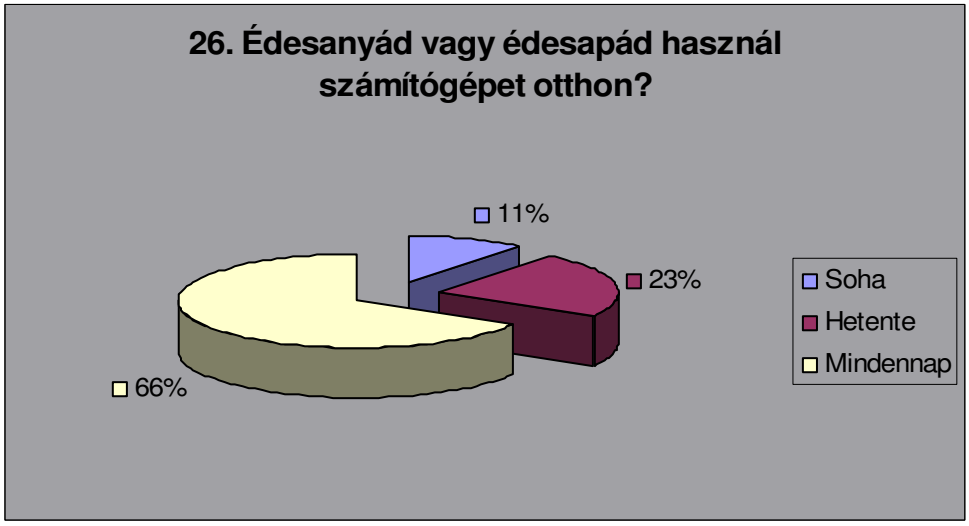

21. Ha választanom lehetne, szívesebben írnék "Windows-os" programokat.
21%
17%
15%
24%
23%
1
2
3
4
5

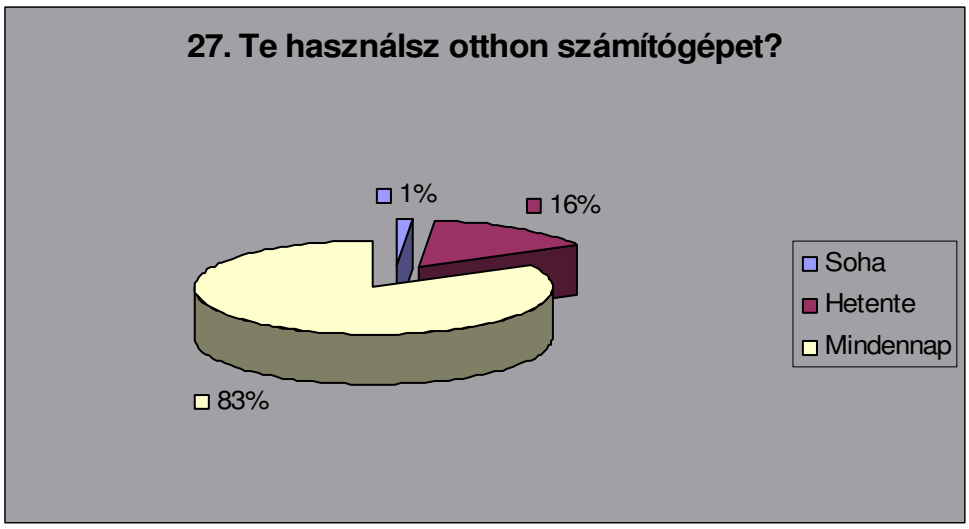

22. Elégedett vagyok az iskola számítógép-ellátottságával.
5%
9%
33%
16%
37%
1
2
3
4
5

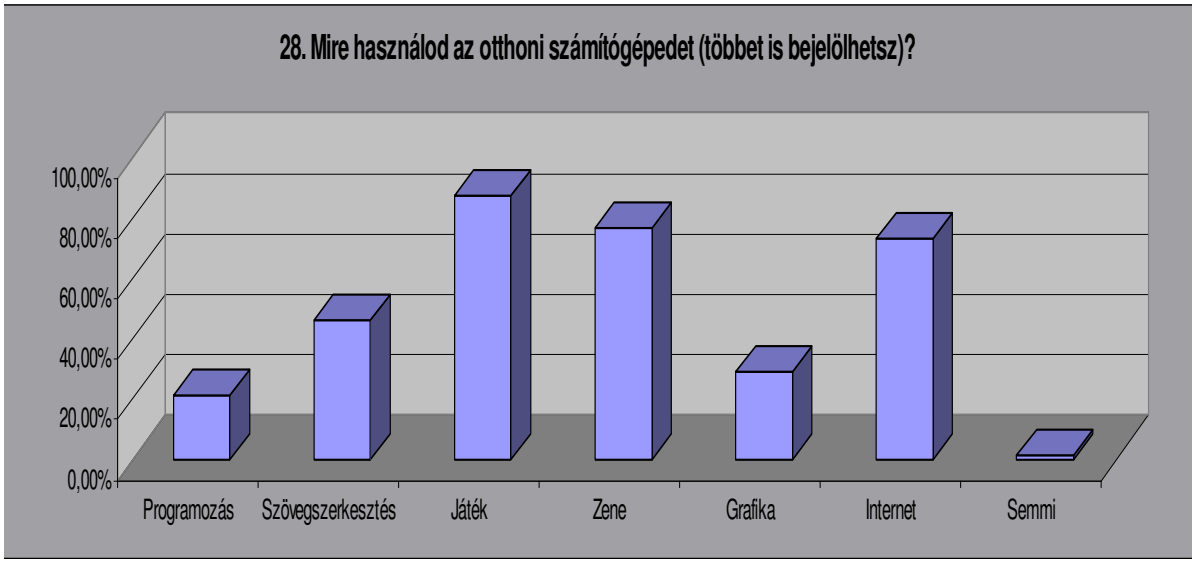

23. Elégedett vagyok az iskola szoftver-ellátottságával.
9%
9%
29%
19%
34%
1
2
3
4
5

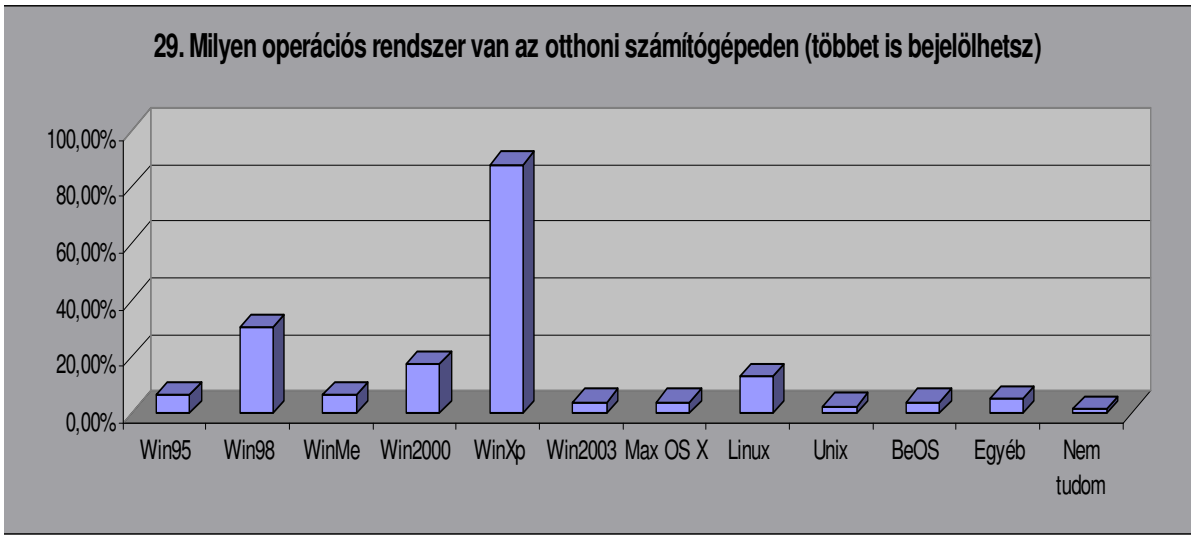

24. Jól használhatóak a labor/gépterem gépei az iskolai munkára.
4%
5%
19%
39%
33%
1
2
3
4
5

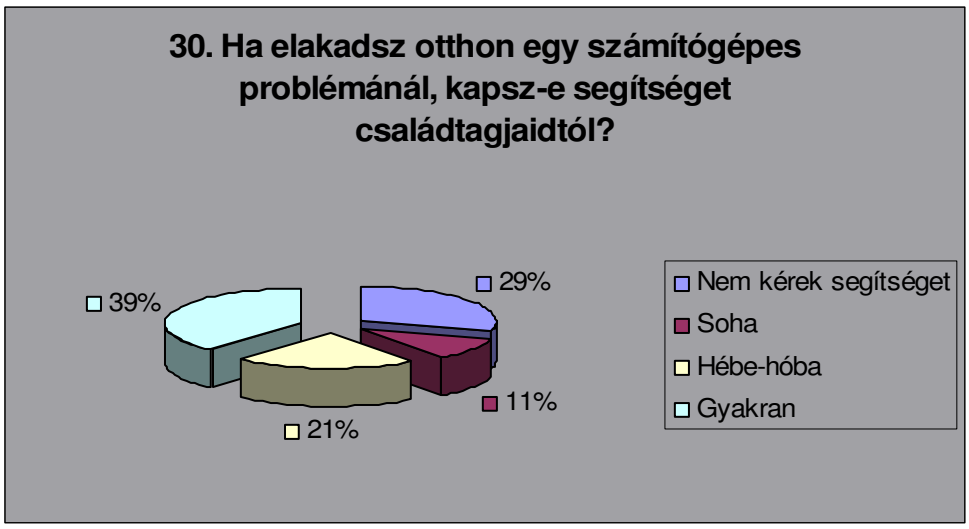

25. Édesanyád vagy édesapád használ számítógépet a munkahelyén?
11%
89%
Szülők használnak számítógépet a munkahelyen:
Szülők nem használnak számítógépet a munkahelyen:

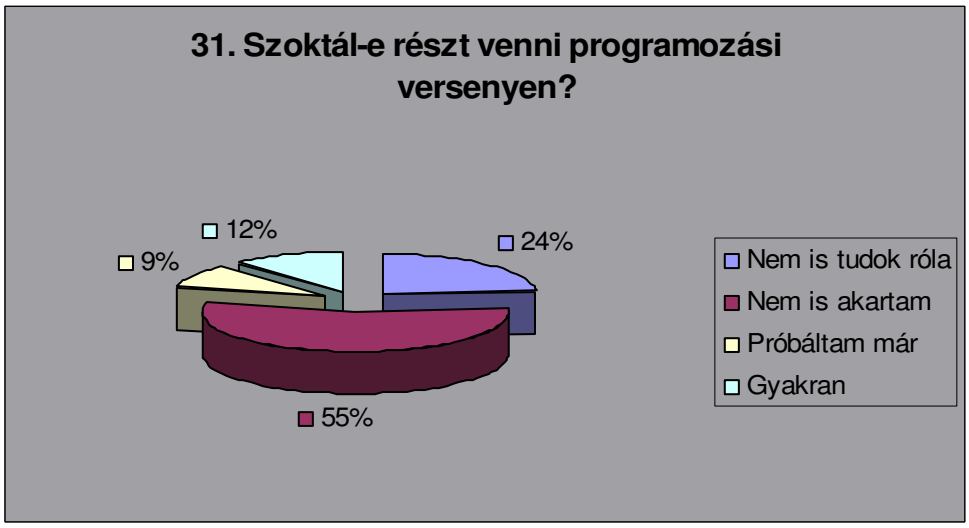

26. Édesanyád vagy édesapád használ számítógépet otthon?
11%
23%
66%
Soha
Hetente
Mindennap

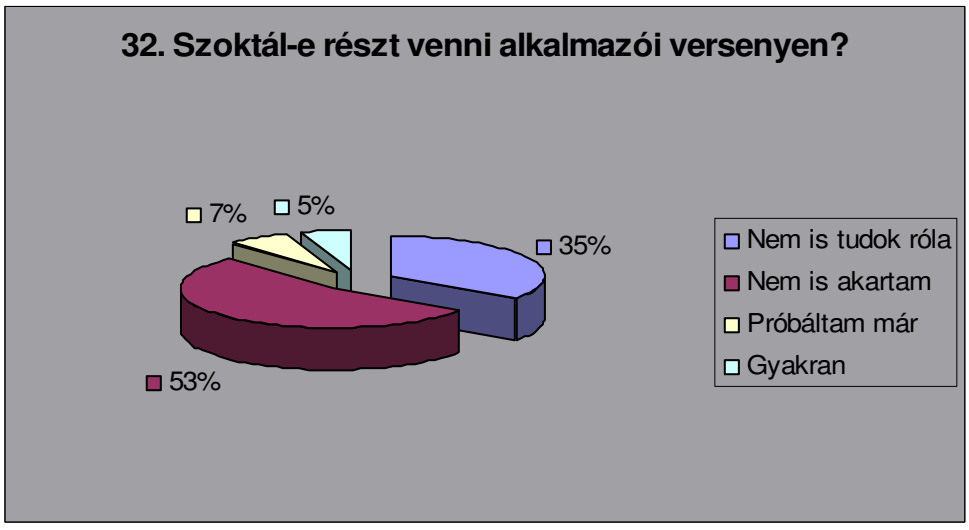

27. Te használsz otthon számítógépet?
1%
16%
83%
Soha
Hetente
Mindennap


28. Mire használod az otthoni számítógépedet (többet is bejelölhetsz)?
100,00%
80,00%
60,00%
40,00%
20,00%
0,00%
Programozás
Szövegszerkesztés
Játék
Zene
Grafika
Internet
Semmi

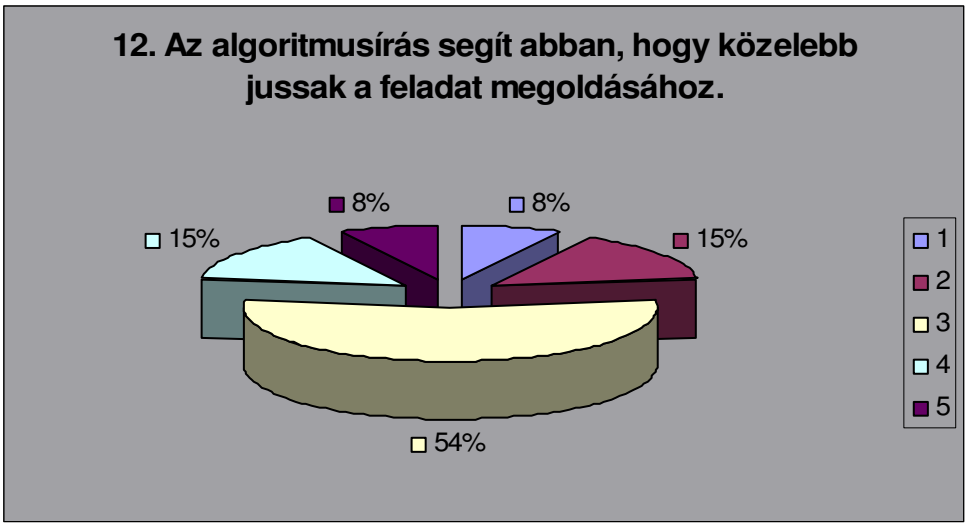

29. Milyen operációs rendszer van az otthoni számítógépeden (többet is bejelölhetsz)
100,00%
80,00%
60,00%
40,00%
20,00%
0,00%
Win95
Win98
WinMe
Win2000
WinXp
Win2003
Max OS X
Linux
Unix
BeOS
Egyéb
Nem tudom


30. Ha elakadsz otthon egy számítógépes problémánál, kapsz-e segítséget családtagjaidtól?
39%
29%
11%
21%
Nem kérek segítséget
Soha
Hébe-hóba
Gyakran

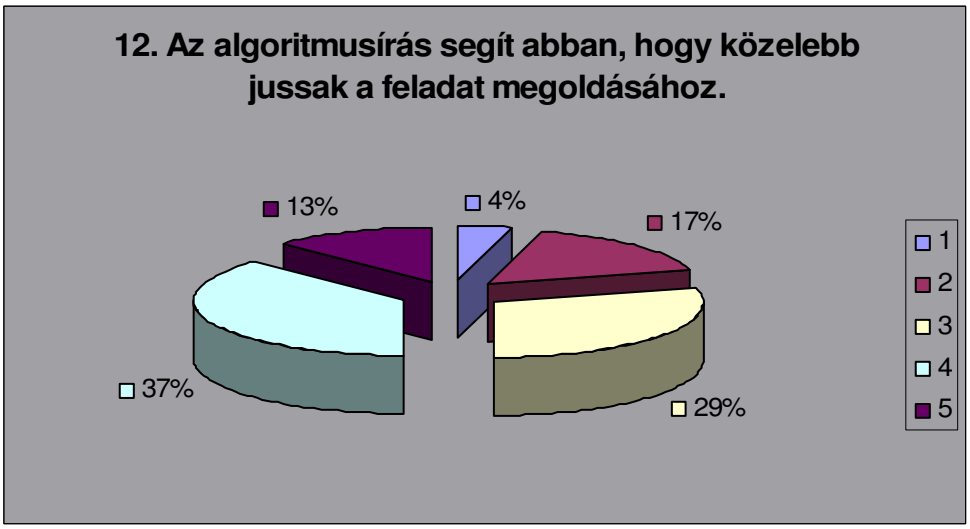

31. Szoktál-e részt venni programozási versenyen?
12%
9%
24%
55%
Nem is tudok róla
Nem is akartam
Próbáltam már
Gyakran


32. Szoktál-e részt venni alkalmazói versenyen?
7%
5%
35%
53%
Nem is tudok róla
Nem is akartam
Próbáltam már
Gyakran

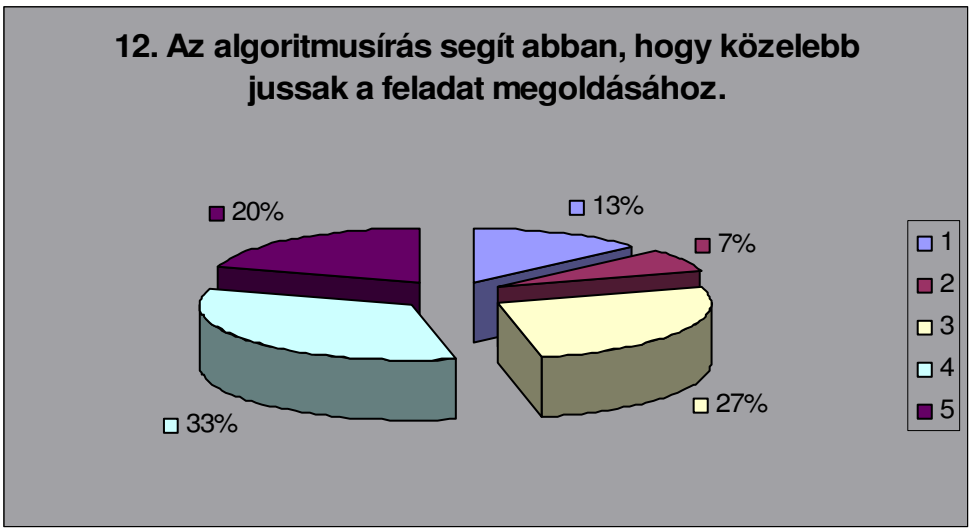
## 1.2 A kérdőív egyes válaszainak eloszlása, évfolyamok szerint



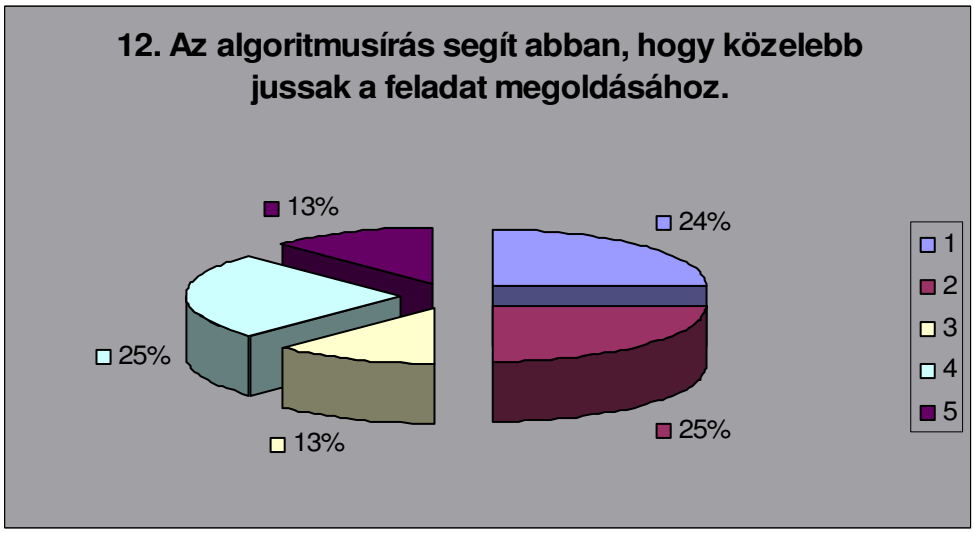

*7. osztály*



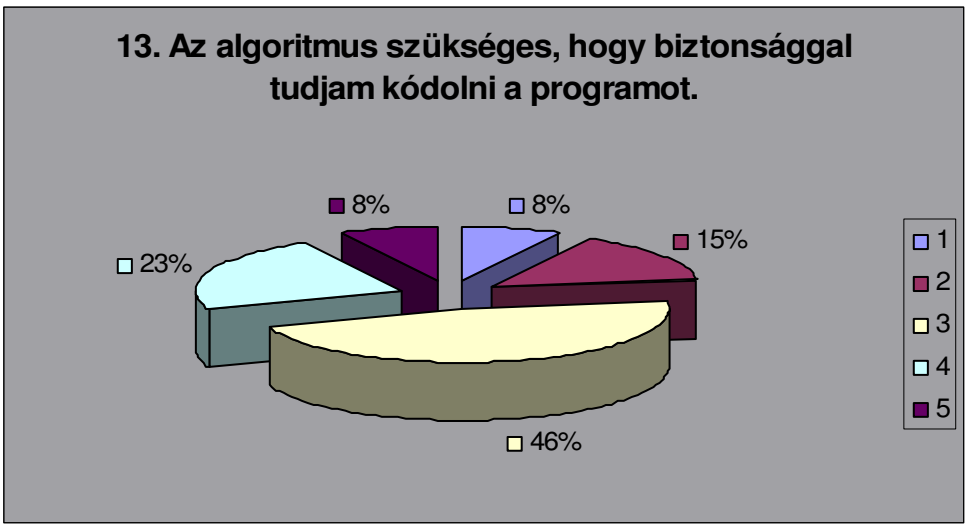

*8. osztály*



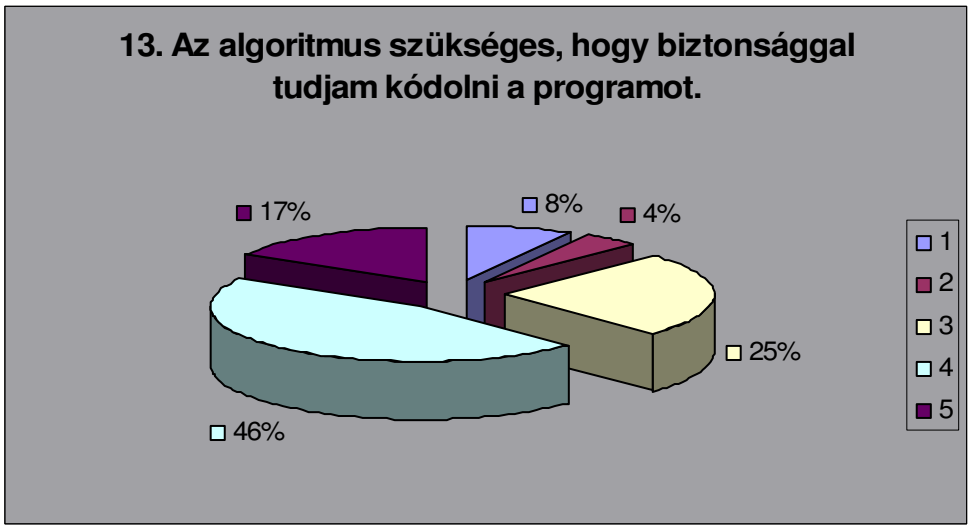

*9. osztály*



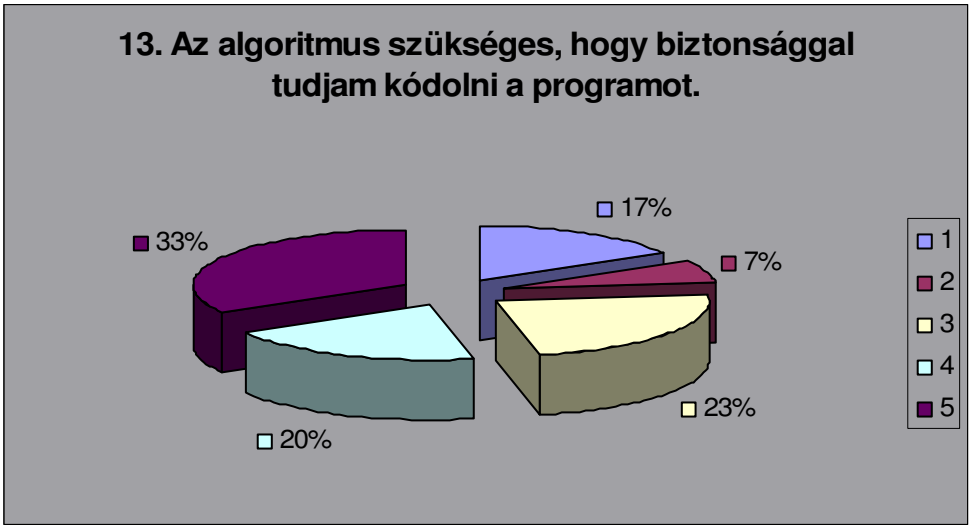

*11. osztály*



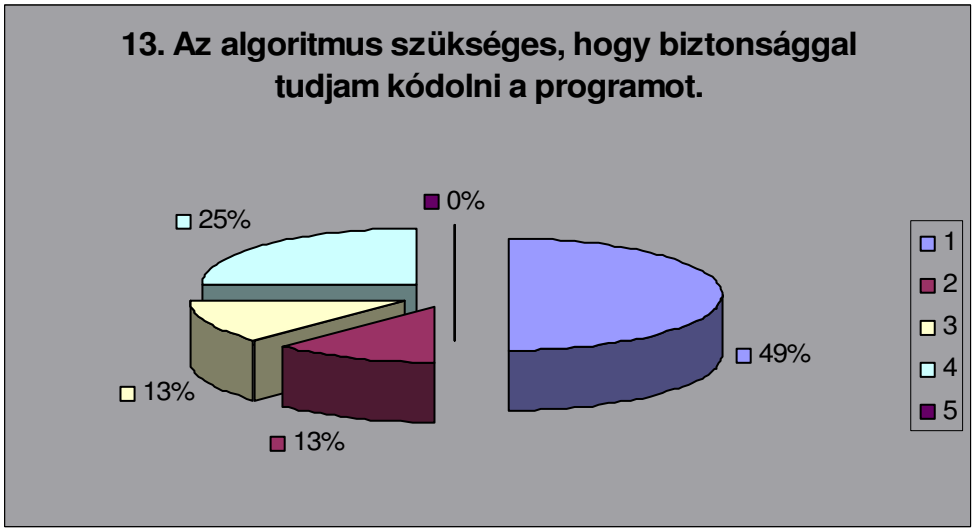

*7. osztály*



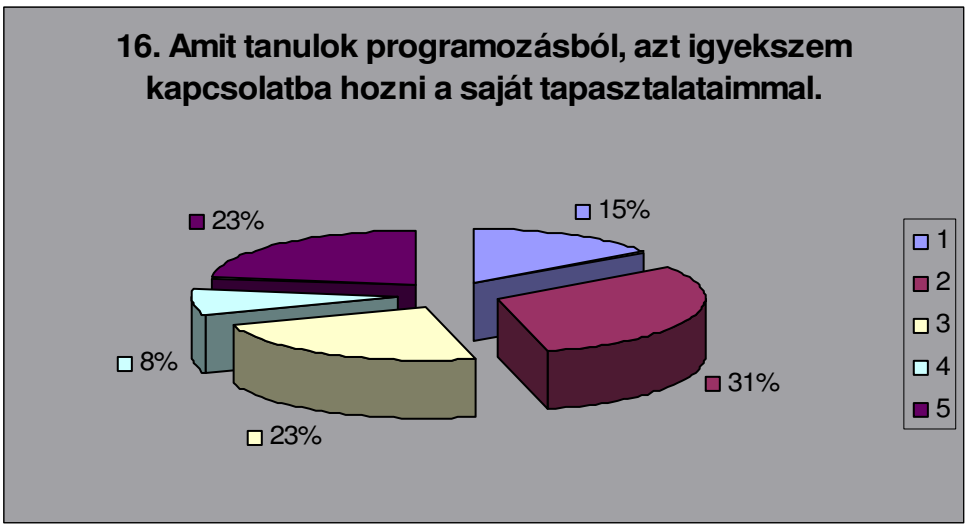

*8. osztály*



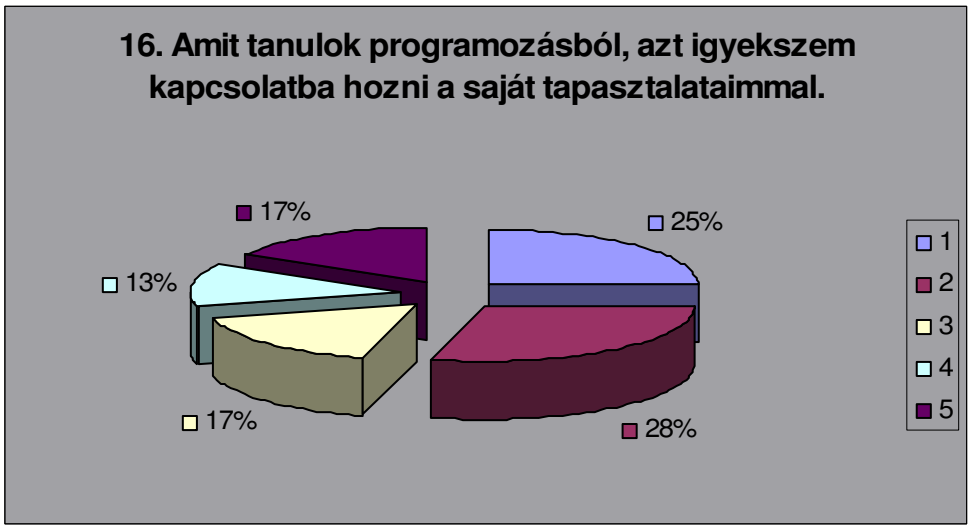

*9. osztály*



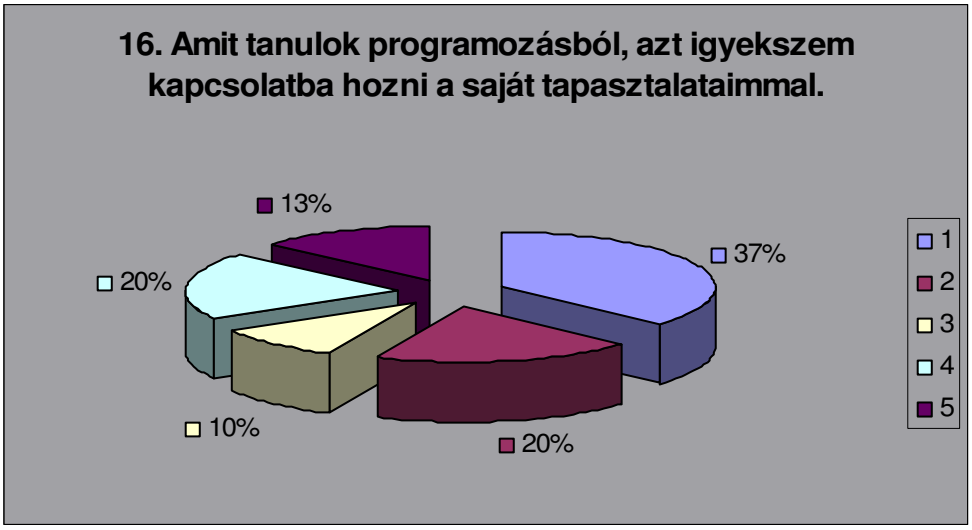

*11. osztály*



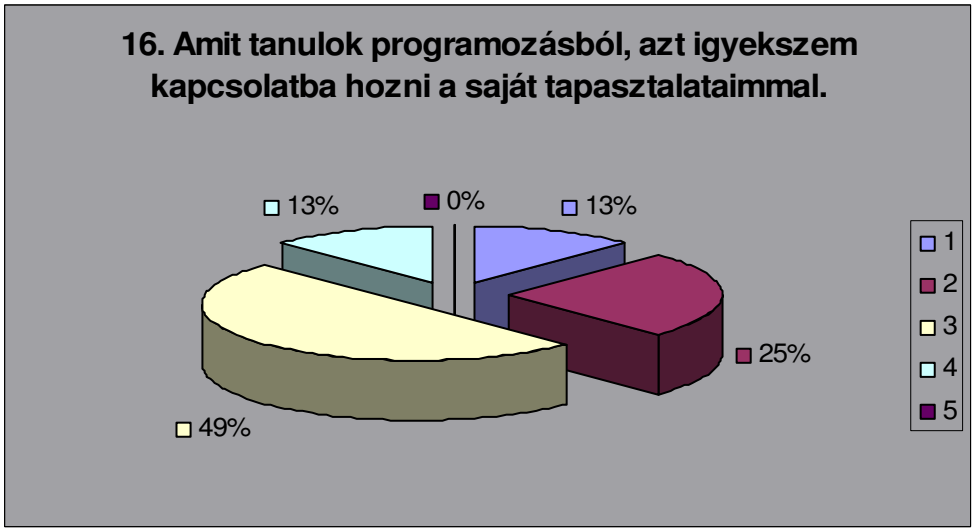

*7. osztály*



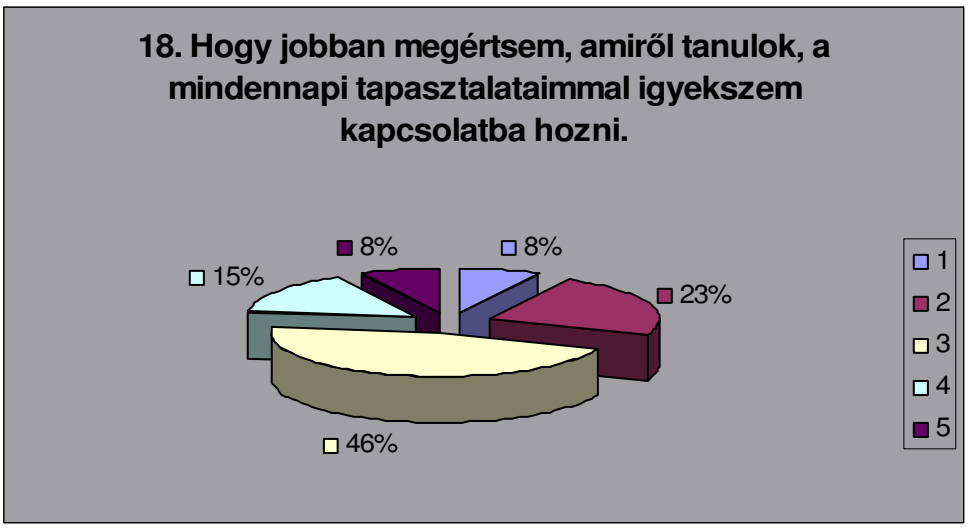

*8. osztály*



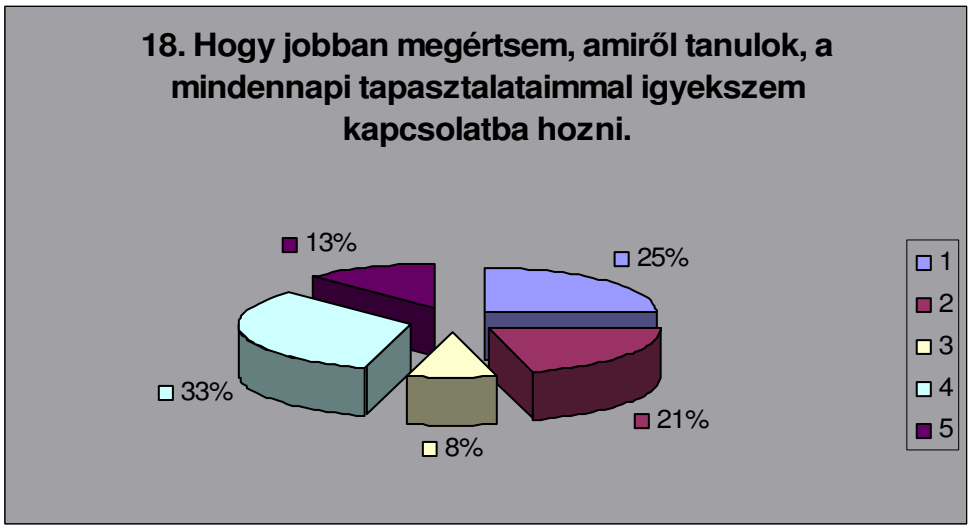

*9. osztály*



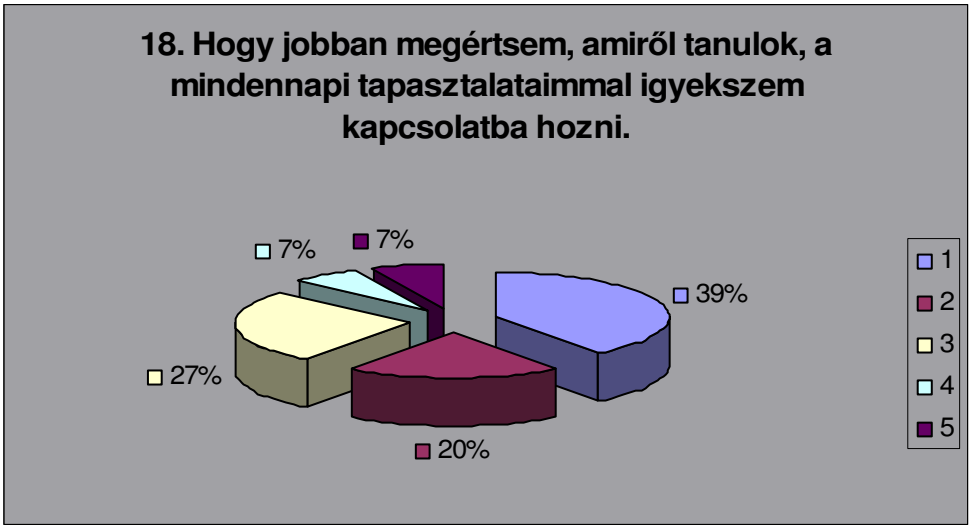

*11. osztály*



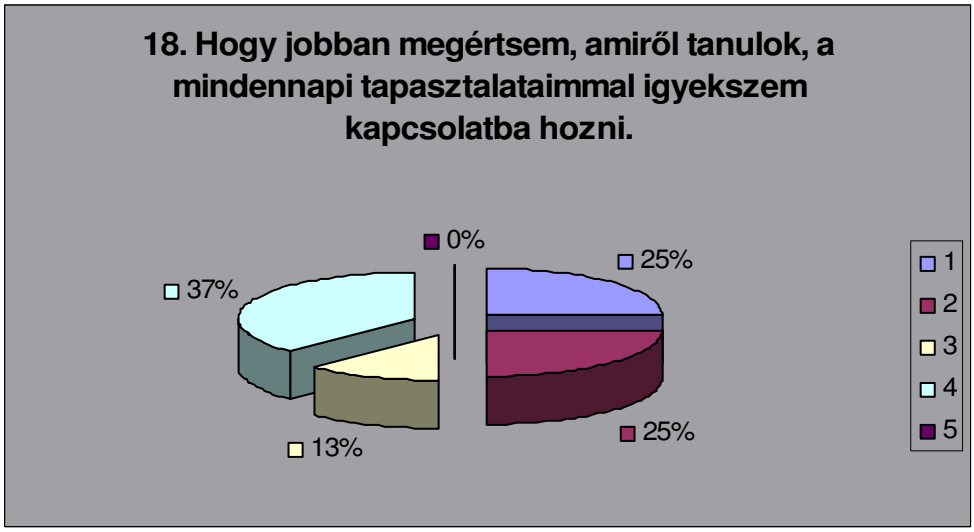

*7. osztály*



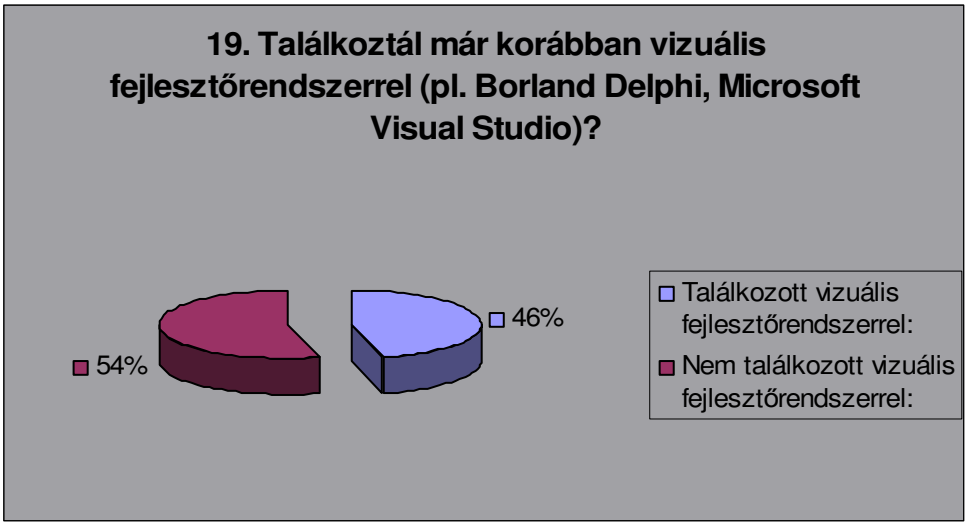

*8. osztály*



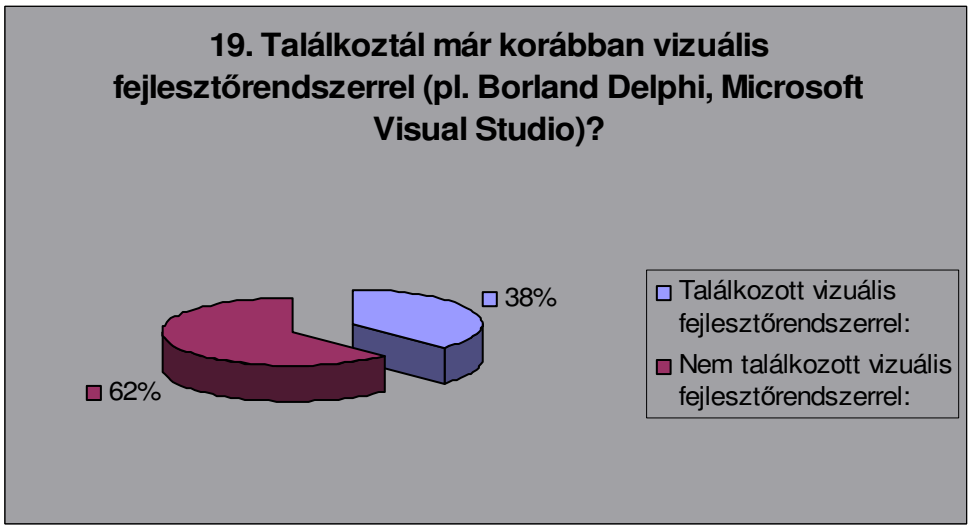

*9. osztály*



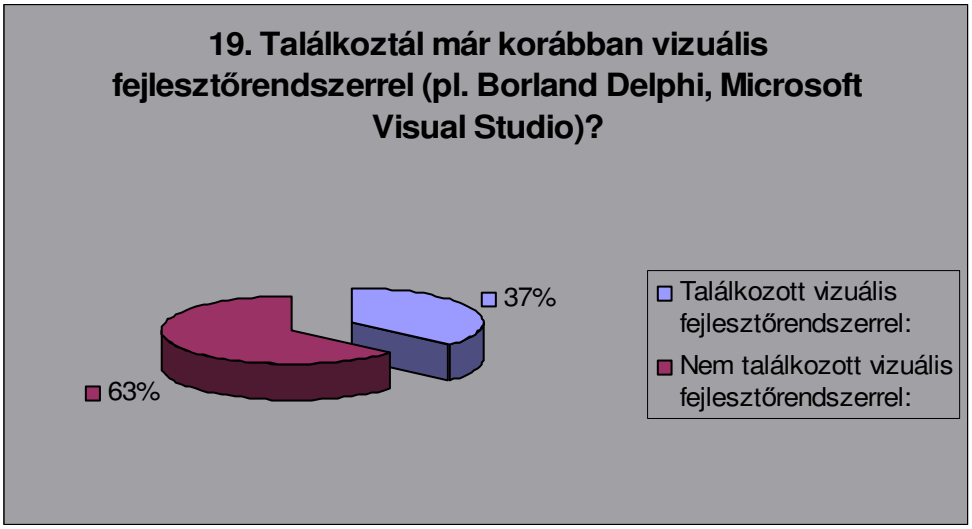

*11. osztály*



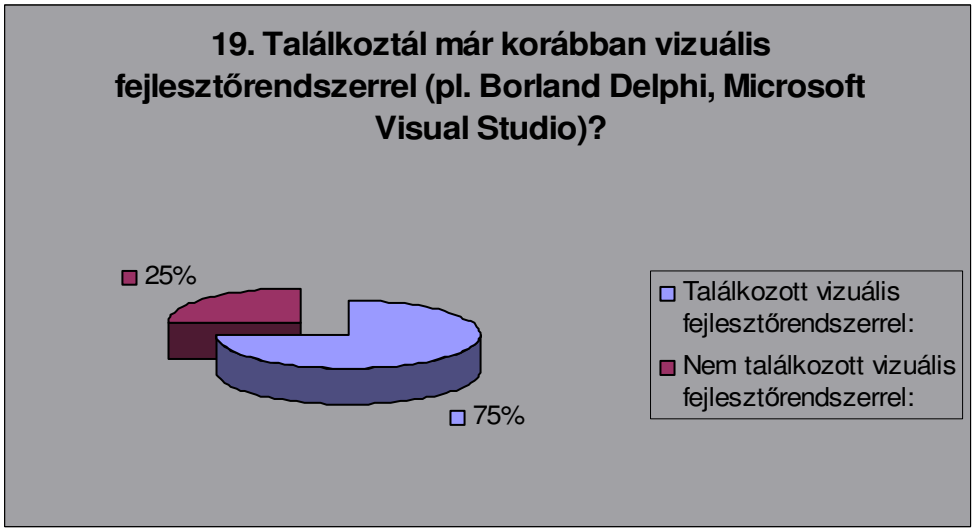

*7. osztály*



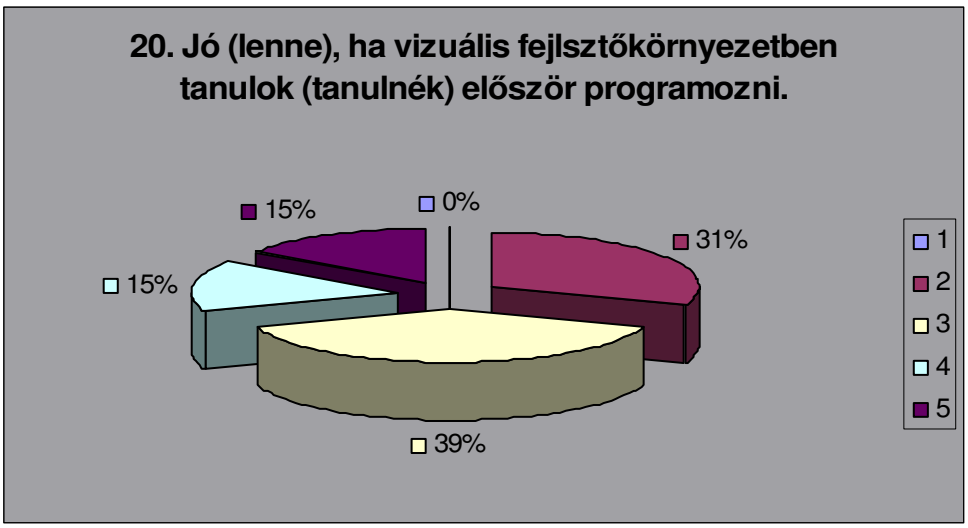

*8. osztály*



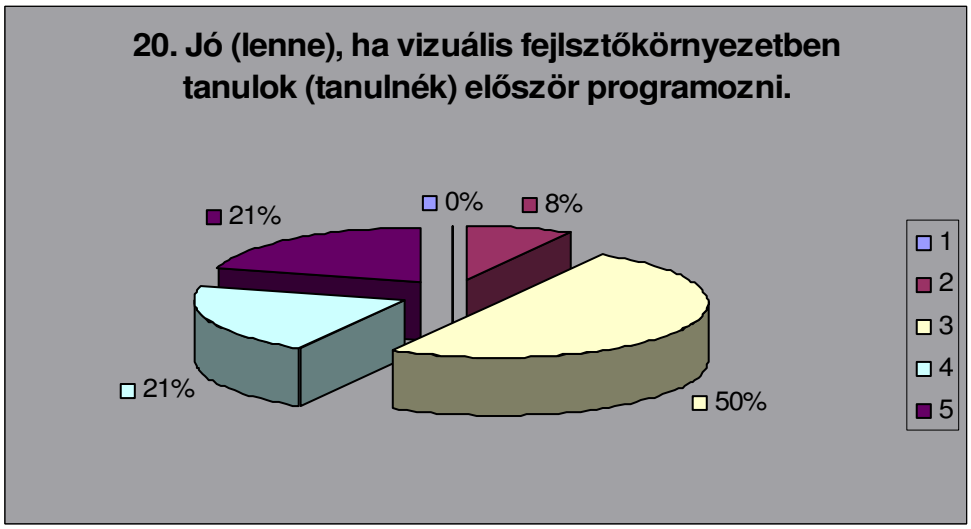

*9. osztály*



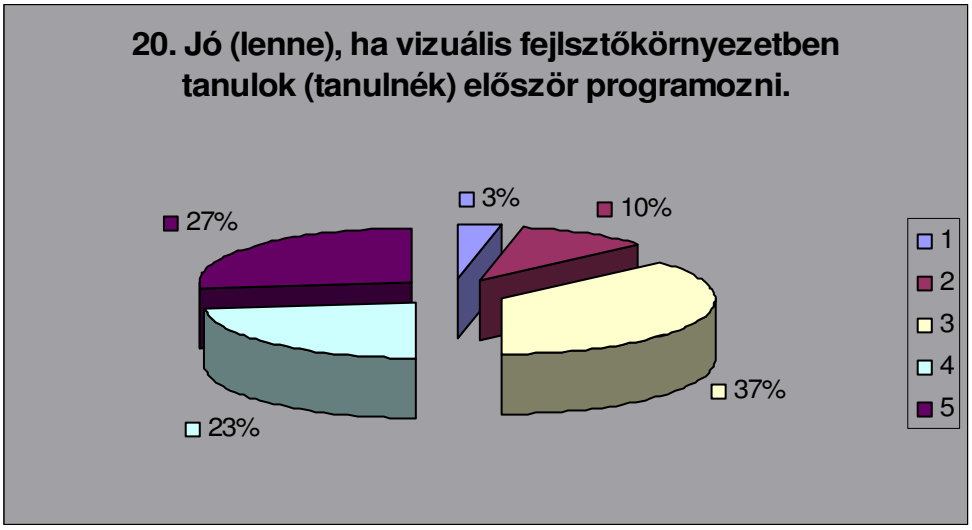

*11. osztály*



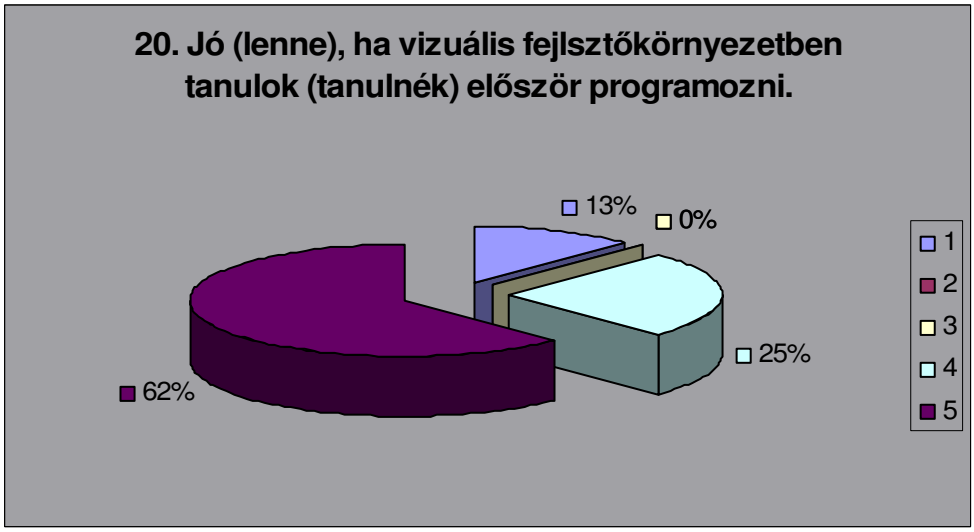

*7. osztály*




*8. osztály*




*9. osztály*




*11. osztály*

# Befejezés

Sok új tapasztalatot hozott a szakdolgozat készítése. Többek között ilyen volt a Turbo Pascal és a Delphi gyakorlatban való összehasonlítása. Számomra a legnagyobb kihívást a kérdőív összeállítása, kiértékelése és eredménye jelentette, ugyanis első alkalommal ástam mélyebbre a pszichológia ilyen téren való alkalmazása és a tesztelmélet terén. Rengeteg új dolgot hozott a dolgozat harmadik része, amelyben nem csak a tankönyvet olvasó diákot hívtam el egy „felfedező útra", hanem saját magamat is.

Ez a dolgozat igazán keveset tudott megmutatni abból a tanítási módszerből, amely naprakésszé tenné a mai középiskolai programozást. Több információra van szükség a „fogadó oldalról", azaz a diákok részéről, így a kérdőív nagyobb mintán való elvégzése pontosabb adatokkal szolgálhat a mostaninál.

A jövőben doktoranduszként szeretnék ezzel a problémakörrel foglalkozni. A programozás-oktatás, mint problémamegoldó gondolkodást fejlesztő „eszköz", minél hatékonyabbá tétele fontos szempont, a közelmúlt PISA vizsgálat – nem éppen pozitív – eredményeinek tükrében.

Szeretnék köszönetet mondani témavezetőmnek, Szlávi Péternek, türelméért és tanácsaiért, különösen a módszertani és kérdőíves részt illetően.

# *Felhasznált irodalom*

## Könyvek, jegyzetek


**[SzZs]** Szlávi,P.–Zsakó,L.: „*Methods of Teaching Programming*”, in Teaching Mathematics and Computer Science, ½, pp. 247-257, 2003

**[FSz]** Farkas Csaba – Szabó Marcell: „*A programozás alapjai Visual Basicben*” – Jedlik Oktatási Stúdió, Budapest, 2004

**[BB]** Bártfai Mária Erika – Bártfai Norbert: *„Fantasztikus Programozás I.”* 26-29. o.– Debreceni Egyetem Egyetemi és Nemzeti Könyvtár 2004.

**[SM]** Sípos Marianna: „*A programozásoktatás megújulása a Visual Studio .NET kínálta lehetőségekkel*”,

**[MK]** Magyar Közlöny 2004/68/11 szám

**[NP]** Nádori (2002) Péter: „*Háló, Kalauz az Internethez*”, HVG 2002. április 27-i számának melléklete

**[SzI]** Dr. Szitó Imre: „*A tanulási stratégiák fejlesztése (Iskolapszichológia 2.)*”, ELTE Eötvös Kiadó, Budapest, 2003.

**[BL]** Dr. Balogh László: „*Tanulási stratégiák és stílusok, a fejlesztés pszichológiai alapjai*”, 7-12. o., KLTE, Debrecen, 1993. Egyetemi jegyzet


## Internetes anyagok


**[SzPI]** Szlávi Péter: „*Szoftverek értékelése iskolai szempontok szerint*” http://digo.inf.elte.hu/~szlavi/InfoOkt/SzoftErt.html

**[SzZsI]** Szlávi Péter – Zsakó László: „*Programozás tanítási módszerek*” http://digo.inf.elte.hu/~szlavi/ProgModsz/SzlaviZsako.pdf

**[TG]** Törley Gábor: „*A teszt*” http://ttkhok.elte.hu/kergezerge/kerdoiv/kerdoiv.html

**[SPSS]** SPSS FAQ: „*What does Cronbach's alpha mean?*” http://www.ats.ucla.edu/stat/spss/faq/alpha.html

**[CM]** L.,J.Cronbach–P.E.Meehl: „*Construct validity in psychological tests*”, 1955 http://psychclassics.yorku.ca/Cronbach/construct.htm