IX. Az informatikaoktatás tartalma

Egy-egy téma oktatásáról cikket írtam [57, 58, 121, 130, 133, 137, 159], más témáknál beszélgetések, tanári konzultációk során mondtam el, hogy mi a különbség a hagyományos informatikatananyag és aközött, amit tanítok. A cikkek ellenére sem mondhatom teljesnek a leírást. Amire most kísérletet teszek, az a szemléletmódnak, az informatika tanításáról való gondolkodásnak a bemutatása – remélhetőleg – jellemző példákon keresztül. Ehhez a NAT 2020 Digitális kultúra tantárgyát veszem alapul, amely a legutolsó megjelent szakmai anyag. A tanterv kidolgozói nemzetközi tapasztalatokat figyelembe véve dolgozták ki az egyes témaköröket. A megjelent dokumentumot kiegészíti számos előadásuk, amely a munka hátterét mutatja be. Ezek egyike az ÉGIG[57] kezdeményezésére szervezett informatika és technika munkaközösségi találkozó, amelynek jegyzőkönyvében [127] leírva is megjelent az egyes témakörök súlyozása:

> *A nemzetközileg elfogadott gyakorlat alapján az informatikaoktatás területei: 1. Az informatikai eszközök használata; 2. Digitális írástudás (szöveges, rajzos, táblázatos dokumentumok, internetes kommunikáció); 3. Problémamegoldás (összetett problémák megoldása, algoritmusok, adatmodellek, adatbázisok, táblázatkezelés programozás); 4. Információs technológiák (robotika, webes és mobiltechnológiák). Az 1. témakör nem önállóan, hanem a másik három témakörben jelenik meg. A három fő témakör aránya a jelenlegi magyar gyakorlatban: 80%–10%–10%, míg a kívánatos az 50%–30%–20% vagy 40%–40%–20% lenne (iskolatípustól függően).*

Az informatika oktatása során tudatában kell lenni annak, hogy az informatikatudomány fejlődik, változik, újraértelmezi a fogalmakat. Az oktatás jellemző célja az adatok értelmezésének és a modellezésnek a tanítása, az informatikai gondolkodás fejlesztése. Ezért jellemzően nem definíciók, hanem kérdések, problémafelvetések sorával jellemezhető az informatikaoktatás. A feltett kérdések, problémák, értelmezési feladatok nem feltétlenül köthetők korosztályhoz, a lehetséges válaszok azonban jellemzők lehetnek egy-egy korosztályra, a válaszoló tudásprofiljára, tapasztalatára.

[57] Élenjáró Gimnáziumok Igazgatóinak Grémiuma

IX/1. Az informatikai eszközök használata (1.)

Ez a téma a NAT 2020-ban a hardverrel és operációsrendszerekkel kapcsolatos gyakorlati ismereteket fedi le. A tematikus egységhez tartozó „fejlesztési területek” és „eredménycélok”:

- *Ismerkedés az informatikai környezettel.*
  - *... megnevez néhány informatikai eszközt, felsorolja fontosabb jellemzőit.*
  - *Megfogalmazza, néhány példával alátámasztja, hogyan könnyíti meg a felhasználó munkáját az adott eszköz alkalmazása.*
  - *Egyszerű feladatokat old meg informatikai eszközökkel. ... összetettebb funkciókat is alkalmaz.*
- *Gyermekeknek készített alkalmazások használata.*
  - *... választ ... néhány applikációt, digitális tananyagot, oktatójátékot, képességfejlesztő digitális alkalmazást.*
  - *... használ néhány, életkorának megfelelő alkalmazást, elsősorban információgyűjtés, gyakorlás, egyéni érdeklődésének kielégítése céljából.*
  - *... feladathoz, problémához digitális eszközt, illetve alkalmazást, applikációt, felhasználói felületet választ. Felsorol néhány érvet választásával kapcsolatosan.*
- *Az informatikai eszközök önálló használata.*
  - *Célszerűen választ a feladat megoldásához használható azonos funkciójú informatikai eszközök közül.*
  - *Az informatikai eszközöket önállóan veszi használatba, a tipikus felhasználói hibákat elkerüli, és elhárítja az egyszerűbb felhasználói szintű hibákat.*
  - *Értelmezi az informatikai eszközöket működtető szoftverek hibajelzéseit, és azokról beszámol.*
- *Az operációs rendszerek önálló használata.*
  - *Önállóan használja az informatikai eszközök operációs rendszereinek felhasználói felületét.*
  - *Önállóan kezeli az operációs rendszer mappáit, fájljait és a felhőszolgáltatásokat.*
  - *Használja a digitális hálózatok alapszolgáltatásait.*
  - *Tapasztalatokkal rendelkezik a digitális jelek minőségével, kódolásával, továbbításával kapcsolatos problémák kezeléséről.*

- *Informatikai eszközök felhasználásának lehetőségei, az informatikai környezet.*
  - *Ismeri és tudja használni a célszerűen választott informatikai eszközök és a működtető szoftvereik felhasználási lehetőségeit.*
  - *Ismeri a digitális eszközök és a számítógépek fő egységeit, ezek fejlődésének főbb állomásait, tendenciáit.*
  - *Képes az informatikai környezetének tudatos alakítására. Ismeri az ergonomikus informatikai környezet jellemzőit, tevékenysége során figyelembe veszi a digitális eszközök egészségkárosító hatásait, óvja saját maga, és társai egészségét.*
  - *Képes önállóan informatikai eszközöket használatba venni és a tipikus felhasználói hibákat elkerülni, egyszerűbb felhasználói hibákat elhárítani.*
- *A mobileszközök, számítógépek, hálózatok operációs rendszerei, felhőszolgáltatások.*
  - *Céljainak megfelelően használja a mobileszközök és a számítógépek operációs rendszereit.*
  - *Igénybe tudja venni a feladataihoz szükséges operációs rendszer és a számítógépes hálózat alapszolgáltatásait.*
  - *Követi a technológiai változásokat az elektronikus könyv, hangoskönyv, oktató-videó és videóblog, az oktatást támogató portálok, alkalmazások használatával.*
- *Segédprogramok, digitális kártevők elleni védekezés, állományok tömörítése.*
  - *Önállóan használja az operációs rendszer segédprogramjait és képes a munkakörnyezet beállításainak elvégzésére.*
  - *Tisztában van a digitális kártevők elleni védekezés lehetőségeivel.*
  - *Használja az állományok tömörítését és a tömörített állományok kibontását.*

Látható, hogy a „fejlesztési területek” az eszközhasználatra vonatkoznak, az „eredménycél” pedig egy gyakorlati tevékenységre való képesség. Ezek a fejlesztési területek és célok nem rosszak, de nem biztosítják sem a jövőbeni önálló fejlődést, sem az informatikai gondolkodás fejlesztését. A megnevezett eredménycélokon túlmutató hosszútávú célok eléréséhez, az informatika tudomány szemléletmódját figyelembe véve a téma tanításakor a főbb fejlesztési területek:

- A használt eszközök, operációsrendszer és egyéb alkalmazások működésének modellezése.

- Az eszközökre jellemző adatok értelmezése.
- Az operációsrendszer és az alkalmazások adatmodelljének jellemzése, objektumok, tulajdonságaik és függvényeik tanulmányozása.
- Az *adat* és a jel (digit) különböző megjelenési, tárolási és továbbítási módjainak megismerése.
- Az eszközök és alkalmazások működési algoritmusainak tanulmányozása, modellezése.
- Informatikai gondolkodás fejlesztése, adott feladatokra és problémákra hatékony megoldások tervezése és kivitelezése, a megoldások verifikációja és validációja.
- Működési problémák beazonosítása a modellek és a jelenség alapján; hibás megoldások, hibalehetőségek felderítése.

Például:

A korábban már tárgyalt gépírás tanulás esetén az eszközhasználattal kapcsolatos fejlesztési terület annak ismerete, hogy a tízujjas vak gépírási gyakorlattal kihasználható egy ergonomikusan optimalizált környezet, begyakorolhatók a megfelelő eljárások a billentyűzet helyes használathoz. Informatikai szempontú fejlesztést jelent, ha a billentyűzet helyett a hasonló beviteli lehetőségek, például a képernyőbillentyűzet és kivetített billentyűzet ergonómiai jellemzése, az ergonómia szempontrendszere is téma.

Az ergonómia mellett, a billentyűzet – karakteres beviteli eszköz – részeként tárgyalandó fogalom az adat, a jel (digit) a karakter kódolása is.

Néhány kérdés:

Mi a szerepe a billentyűn látható karakternek, mi van, ha lekopott? Milyen kapcsolat van a billentyű jele és a képernyőn megjelenő jel között? Miért vannak nemzeti karakterkiosztások? A nyomtatott nagy ’A’ az ’Á’-tól egy jellel, a vesszővel tér el; mi a különbség a leírt karakter jelei, a billentyűleütésekkel létrehozott jelek és a tárolt karakterek között? Az egyes karakterkódolási, írási rendszerekben (pl. morze, braile írás, daktil) mi jelenti a jeleket, hogyan adható meg az ’A’, illetve az ’Á’ karakter? Mi a „bájt”, mi a „bit” és mi a kapcsolat a kettő között?

Példa:

Mi lehet a „MAR” szóból, ha 1 jelet módosítunk? Nyomtatott írásban lehet MÁR, MAP, MÄP. Binárisan kódolva, 1 bit módosításával lehet: LAR OAR, IAR, EAR, M@R, MCR, MER, MIR, MAS, MAP, MAV, MAZ.

A hibakezelés irányából közelítve a témát, néhány példa:

A digitális írástudás birtokosa képes kell legyen legalább egy elmélete arra vonatkozólag, hogy a prezentációjának alsó része miért nem jelenik meg a kivetítőn. Vagy, hogy miért nincs hangja a hangszórónak, miért nem jelenik meg a leütött billentyű karaktere a képernyőn, miért tarthat egy adatművelet hosszú ideig.

A használt hardver-szoftver rendszer működési elveinek, az adattárolás és továbbítás módszereinek ismerete alapján egy hibajelenséget (vagy nem várt jelenséget) tudni kell diagnosztizálni, majd döntést hozni a szükséges lépésekről. Ehhez ismerni kell az eszközök, az operációsrendszer és az alkalmazások működési elvét, készség szinten kell alkalmazni az informatikai gondolkodást.

A konkrét eszközök és alkalmazások az informatikai szempontú megismerésén túl, az elméleti összefüggésekre is ki kell térni, mert a fejlődés új rendszerek önálló megismerésének képességét teszi szükségessé. Ezért fontos a használt fogalmak elemzése, jelentéseinek megismerése.

Például:

Mit jelent a „digitalizáció”? A világ digitális vagy analóg? A billentyűzet hardvereszköz? Mit csinál a CPU? Mi az és hogyan működik az ALU? Hol vannak a fájlok fizikailag és mi a kapcsolat a fizikai és logikai adatstruktúra között? Ha egy gép (szerver vagy munkaállomás) lehet egyetlen fájl, akkor hogyan értelmezhető ennek a gépnek az operációs rendszere és a gépben látszó eszközök? Mi a kapcsolat a valódi gép és a rajta futó virtuális gép között?

Az Informatikai eszközök használata a legkisebb témakör, amely a többi témakörbe integrálva tanítandó. Óraszám nem tartozik hozzá, ezért az oktatás során külön figyelmet kell fordítani az informatikai kérdések felvetésére, az ismeretek elsajátítása során az informatikai gondolkodás fejlesztésére. A NAT eredménycéljaiban olvasható „használ” és „választ” mellett az informatika tudáshoz a „modellez”, „tanulmányoz”, „tervez”, „felderít” cselekvések vezetnek.

6.2.1

IX/2. Információs technológiák (4.)

A legkisebb súlyú, a tanórák 20%-ára tervezett témakör az Információs technológiák téma. Negyedik témaként ide sorolnak mindent, ami az előzőkbe nem fért bele. Főbb területek: célszoftverek használata, kommunikáció, netikett, biztonság, együttműködés eszközei, szocializáció az

információs társadalomban. Jellemző eredménycél, hogy a diák az adott témát „ismeri" és lehetőséget helyesen „használja". Informatikai szempontból az lenne kívánatos, hogy az adott témákról és lehetőségekről rendszerszintű ismeretei legyenek, értse a használat hátterében zajló folyamatokat, a szabályok megsértéséből származó károkat és következményeket.

A témakör rendkívül szerteágazó, a megnevezett fejlesztendő területek gyűjtőfogalmak. Egyetlen fejlesztendő terület eredménycéljait tekintve is látszik a célok megfogalmazásának szempontja:

- *Információkeresés és online kommunikáció*
  - *Ismeri és alkalmazza az információkeresési stratégiákat és technikákat, a találati listát a problémának megfelelően szűri, ellenőrzi azok hitelességét.*
  - *Etikus módon használja fel az információforrásokat, tisztában van a hivatkozás szabályaival.*
  - *Használja a két- vagy többrésztvevős kommunikációs lehetőségeket és alkalmazásokat.*
  - *Ismeri és szükség esetén alkalmazza a hátránnyal élők közötti kommunikáció eszközeit és formáit.*
  - *Az online kommunikáció során alkalmazza a kialakult viselkedési kultúrát és szokásokat, a szerepelvárásokat.*

A fenti célok eléréséhez kapcsolódó informatikai kérdések:

- Az információkeresést segítő eszközöknek hogyan épül, frissül a keresési tartománya?
- A kapott találati lista mennyire teljes?
- A találati lista rangsorolásának mik az elvei?
- Milyen mértékben befolyásolja az ítélőképességet a véleménybuborék?
- Mi hitelesít egy adatot, dokumentumot, mire vonatkozik a hitelesítés?
- Milyen működési, társadalmi, illetve személyes problémák, konfliktusok elkerülését szolgálják az etikai szabályok, viselkedési kultúrák és szokások?

A többi fejlesztendő területre is jellemző, hogy az eredménycélok az éppen aktuális információs technológia alkalmazására irányulnak. A leírt cél az, hogy ismerje a technológia alkalmazásának lehetőségét, az alkalmazás szabályait. Az informatikai tudás ezen felül azt jelenti, hogy az egyén érti a szerepét az információs társadalomban, tisztában van azzal, hogy *rendszerkomponensek*et használ, amivel ő is a rendszer egy komponensévé válik. Érti a használt

technológiák *működési elv*ét, helyesen *értelmezi* a kapott *adatokat*, átlátja, hogy az egyes technológiák használata során milyen adatokat szolgáltat önmagáról, fel tudja mérni, hogy ennek milyen hatása lehet másokra, illetve önmagára nézve.

6.2.2

IX/3. Digitális írástudás (2.)

A legdominánsabb témakör, amely a tervek szerint az informatikaórák 40–50%-át fedi le. Az egyes fejlesztendő területek jellemzően egyes dokumentumtípusokhoz köthetők. Az eredménycélok itt is a felhasználási lehetőségek ismeretét jelentik.

A digitális írástudás – a témakörök alapján olyan készség, amely – a gondolataink, az érzéseink kifejezésére ad lehetőséget. A „digitális írás" során digitálisan tárolt dokumentumot hozunk létre, amelynek olvasója, értelmezője – alapértelmezetten – humán egyed. A dokumentumok eszerint az ember-ember közötti kommunikáció csatornái. A digitális írás a gondolat kódolása. Az egyes dokumentumtípusok használatának ismerete olyan, mint egy adott nyelven a beszéd képessége, amely azonban a nyelvtan és beszédstílus, esetenként a metakommunikáció ismerete és használata nélkül zajos, pontatlan kommunikációt eredményez.

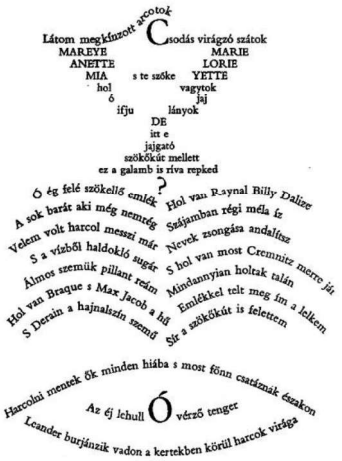
Az ember ötféle érzékelésre képes: látás, hallás, tapintás, ízlelés, szaglás, de a külvilággal több, mint 80%-ban látás útján van kapcsolata [128 p:14] (ha sérült, akkor a hallás és tapintás válik dominánssá). A digitális világgal – illetve azon keresztül más emberekkel – csak látás és hallás útján van kapcsolatunk, ezért a dokumentumokban csak vizuális és audió adatokat tárolunk, de a vizualitás elsődlegessége nagyon jellemző. Ember-ember közötti kommunikáció során az adó a gondolatait, érzéseit, állapotát kifejező adatokat „küld", amelyek – tekintettel a vevő tulajdonságaira – főleg audió-vizuálisan dekódolhatók. Említésre méltó, hogy a nyomtatást követően tapintással és szaglással is kaphatunk információt.

Az ember egyéni és társadalmi evolúciójában jelentős szerepe van a beszéd (artikulált hang), majd az írás, nyomtatás kialakulásának. A digitális írástudásnak is hangsúlyos témája a szöveg rögzítése. Azonban a beszélt nyelv és írás soha nem tudja önmagában pontosan kifejezni a gondolatot, érzést, állapotot. A pontosítást például metakommunikációval, intonációval érhetjük el. A digitális írás során a szöveg karakterek képeként és karaktersorozatként is értelmezhető, amelynek információtartalmát kiegészíti a szöveg digitalizálásának módja, a karakterek elrendezésének módja (például a szöveg tördelése, kapcsolása), a vizuális megjelenést befolyásoló tulajdonságok és a hozzáadott nem szöveges – de jellemzően vizuális – elemek.

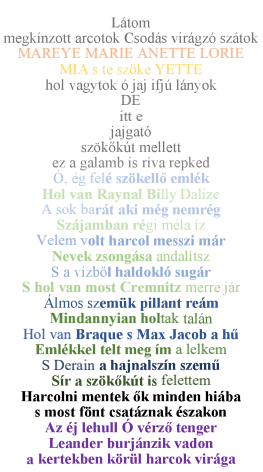
Példaként a digitális írás módjaira és ezek kommunikációs szerepére nézzük Apollinaire versét, amelynek szövegét Radnóti Miklós fordította.

- A szöveg szavalható, ami digitálisan rögzíthető. A tartalom alapján a hangdokumentumhoz háttérzaj (vagy zene) adható.
- A mű eredetileg képvers, digitalizált formája rasztergrafikus kép, a vers szövege is képi elemként jelenik meg. (33. ábra)
- A szöveget karaktersorozatként tárolva, a tördelés és tipográfia elemeit használhatjuk további információ közvetítésre. (34. ábra)
- A szöveget kifejezésekre, szavakra szedve vektorgrafikus algoritmusokat használva szófelhőt készíthetünk belőle. (35. ábra)

5. táblázat: Apollinaire – Radnóti Miklós: A megsebzett galamb és a szökőkút



33. ábra: képvers[58]



34. ábra: formázott szöveg

35. ábra: generált[59] *szófelhő*

Bár a szöveg (pontosabban annak magyar fordítása) adott, az interpretációnak számos eszköze van, amellyel jelentős többletinformációt szolgáltatunk. A vers címe a képi megjelenésre utal, de ez a szöveges tartalomban csak a látvány és a közlő belső világa közötti kapcsolataként jelenik meg.

A digitális írástudás nem pusztán a szöveg bevitele, hanem a többlet információnak a szándéknak megfelelő, minél pontosabb, hatékony hozzáadása. A digitális írás lényege, hogy a dokumentum értelmezése könnyű legyen és az eredmény a közlés szándékának megfeleljen. [129]

A témakör oktatása/tanulása során megvizsgálandó, hogy az adott médiaelemek a képi, szöveges és hang *adatok rögzítés*ére milyen *absztrakció*t használnak, az adott absztrakció milyen *tulajdonság*okkal rendelkezik, mennyire felel meg a közlési szándéknak és hogyan segíti az

[58] A képvers és szöveg forrása: https://hu.wikisource.org/wiki/A_megsebzett_galamb_és_a_szökőkút [2021.10.30]
[59] Készült a szöveg felhasználásával https://wordart.com/ [2021.10.30] használatával.

*értelmezés*t. A NAT tervezet digitális írástudás fejezetének fejlesztendő területei korosztálytól függően csoportosítják a médiákat, az eredménycélok jellemzően az adott dokumentumtípusok alkalmazási ismeretére vonatkoznak, de a terjedelmi korlátozások miatt a területek kifejtése nem nevezhető strukturáltnak, az ezek alá rendelt eredménycélok néha csak utalást tartalmaznak a sokféle ismeret elvárására.

- *Rajzos dokumentumok elektronikus létrehozása*
- *Adatok értelmezése, csoportosítása, tárolása*
- *Dokumentum létrehozása szövegszerkesztő és bemutatókészítő alkalmazással*
- *Különböző típusú dokumentumok iskolai, tanórai, hétköznapi célú felhasználása*
- *Multimédiás elemek készítése*
- *Nagyméretű szöveges dokumentumok szerkesztését elősegítő eszközök*
- *Multimédia és webes dokumentumok szerkesztése és készítése*
- *Grafikus ábrák készítése és képszerkesztés*
- *A probléma megoldásához szükséges módszerek és eszközök kiválasztása*

Több fejlesztendő terület említi a multimédiát, ahol a „multi" a kép, hang, videó kombinációt jelenti. Fentebb láthattuk, hogy a szöveg (mint a gondolat nyelvi kifejezése) digitális megjelenése számos formátumban lehetséges, így a multimédia minden változatában implementálható. Másrészt, az eredménycélok között többször olvasható „szövegszerkesztő" alkalmazást a multimédiás alkalmazásoktól elkülönítve találjuk, de benne a szöveg mellett képi elemek megvalósítását is elvárja. Funkcionálisan a tárgyalt szövegszerkesztő is multimédiaalkalmazás, ugyanakkor célszerű a kép, hang, videó mellett külön médiának vagy médiaelemnek tekinteni a szöveget. A fogalomzavart fokozza, hogy a multimédiás dokumentumok mellett külön említik a webes dokumentumot.

A digitális írástudás témakör látványosan megáll az alkalmazástípusok tanítása szintjén. Ezért kérdéses, hogy miért kellene informatikából egyik vagy másik alkalmazást tanítani, ami az informatikaoktatás devalválódását eredményezi. A digitális írástudás a digitális eszközök – alkalmazások, appok – *hatékony felhasználás*a érdekében szükséges, ehhez *informatikai gondolkodás*ra van szükség. A digitális írástudás programok *célszerű használat*át jelenti. A programok az informatika tudomány termékei, ebből következően a működésüket informatikai szempontok figyelembevételével kell vizsgálni. A digitális írástudás témakörben feltüntetett alkalmazástípusok felhasználói ismeretén túl szükséges az alkalmazások működésének a modellezése és a modellből következő felhasználói elvek ismerete.

Az aktuálisan tanított alkalmazástípusok felsorolásán túllépve, a témakör informatikai tartalmához az egyes médiákat külön kell jellemezni és értelmezni kell a különböző multimédiás dokumentumokon belül a szerepüket. Az informatikai megközelítést az *adatok*kal és *modellek*kel operáló *absztrakció* jelenti.

Médiaelemek

A grafikai, képi elem két fajtáját különböztetjük meg. Az egyik a rasztergrafikus (pixelgrafikus), melynek elemi objektuma a pixel. Ennek tulajdonságai a színkomponensek értéke. A kép pixelek fix méretű táblázata, az adatok időfüggetlen 2D megjelenítésére alkalmas. Előállítása programmal vagy szenzorokkal kapott adatokkal (pl. fényképezés) lehetséges. A másik módja a kép absztrakciójának a vektorgrafikus ábrázolás. Ebben az esetben az elemi objektum az alakzat, amelynek tulajdonsága a határát alkotó pontok és ezeket összekötő vonalak listája, a vonalak és a belső terület szín/mintázat tulajdonsága, mérete…. A vektorgrafikus kép az alakzatok egymáshoz képest rögzített helyzetű összessége. A képek tárolása jellemzően binárisan történik, de van karakteres forma is (svg). A képre jellemző a tároláshoz alkalmazott tömörítés, ami sok formátum esetén veszteséges. Az informatikatudás része kell legyen a tömörítési eljárások vázlatos ismerete, hatása a kép minőségére. Vektorgrafikus képnek tekinthetjük a diagramokat, szófelhőt is, ahol a kép előállításához a megjelenítést befolyásoló adatokból program generálja az alakzatokat.

A 3D elemeknek többféle absztrakciója és megjelenítése létezik. A legegyszerűbb 3D-ábrázolás valójában kétdimenziós, valamilyen perspektivikus ábrázolást használ a megjelenítésre. Két 2D képből színszűrővel és megfelelő eszközzel 3D érzését lehet kelteni. A 3D tervezőprogramok térbeli vektorgrafikus absztrakciót használnak, de a megjelenítés képernyőre vagy nyomtatásra jellemzően kétdimenziós projekció. A 3D nyomtatás ezeket a térbeli testeket képes előállítani, de kérdéses, hogy a létrejövő tárgy média elem-e. Inkább végtermék. (Azt gondolom, hogy egy igazi 3D digitális elem hologramként jelenik meg, körbejárható, és drónhoz hasonló szerkezettel („repülő egérrel”, azaz denevérrel) szerkeszthető.)

Hang digitalizálása során a térbeli tulajdonságokat csatornákra bontással lehet tárolni. A hang absztrakciójában a hanghullám tulajdonságainak időbeli állapotváltozásai dominálnak. Kérdés, hogy a csatornák számossága jelent-e dimenziót. A csatornák a 3D anaglif megjelenítéshez hasonló elven a dekódoló egységek megfelelő elhelyezésével teszik lehetővé a térbeli érzékelést, ezért a sztereó érzékelés inkább több médiaelem egyidejű befogadása, mintsem a hang térbeli absztrakciója.

Szöveg médiaelem legkisebb objektuma a karakter. A karakter az adott nyelvre jellemző legfeljebb 4 bájtos, tipikusan 1 bájtos adat, a szöveget a karakterkészlet egydimenziós sorozataként tároljuk. A nyelvi egységeket (jellemzően szavakat) speciális karakterrel, szóközökkel választjuk el egymástól, a nagyobb gondolatok határolókaraktere a bekezdés vége jel. A beszédben hallható egyéb nyelvtani tagolásokat mondathatároló írásjelekkel, központozással, kötőjelekkel jelöljük. A szöveg megjelenése lehet hang (felolvasó eszközzel) vagy 2D kép, ekkor – jellemzően – a szöveg sorokra tördelve olvasható. A szövegelem megjelenését hozzáadott tipográfiai tulajdonságokkal adhatjuk meg. Ezzel együtt a szöveg nagyobb egységei objektumként értelmezhetők, melyek megjelenéssel kapcsolatos tulajdonságokkal rendelkeznek. A formázott szöveg tárolása lehet bináris (doc), de napjainkban sokkal jellemzőbb a tipográfiai jellemzők jelölőnyelven történő megadása, az adatok és formátum szöveges vagy veszteségmentes tömörítéssel történő tárolása.

Az animáció határesete a multimédiának, hiszen jellemzően képek, grafikák egymás után történő megjelenítéséről van szó. Hasonlóan a felvett videó is a képek (framek) időbeli egymásutániságát megtartó sorozata. Mindkettő tekinthető a 2D – esetleg 3D – ábrázolás időbeli kiterjesztésének, amely azonos médiaelemek sorozata. Azonban az időbeli kiterjedés szinte természetes velejárója, hogy hang is adható a képi médiához, ezért inkább multimédiákhoz sorolható.

Multimédia alkalmazások

A „multimédia” fogalomnak – ahogy az sok más informatikai fogalomra is igaz – több értelmezése volt, van és talán még lesznek újabbak is. Informatikaoktatás szempontjából azonban lényeges, hogy multimédiás tartalom előállításához először az egyes médiaelemek jellemzőit, adatmodelljét célszerű megismerni, ezt követően lehet egy-egy cél érdekében ezen elemekből többfélét kombinálni. A multimédiás alkalmazások célszerű felhasználásához ismerni kell az egyes elemek objektummodelljét és azt is, hogy ezen elemeket a multimédiás alkalmazás hogyan kombinálja. Ezért tartom fontosnak például a Paint mint egyszerű, tisztán rasztergrafikuskép készítő alkalmazás tanítását. Ezen az alkalmazáson keresztül elemeztem, hogy hogyan lehet a menü és az alkalmazás gyakorlati trükkjei mellett a rasztergrafikuskép adatmodelljét is tanítani, az informatikai ismeretek mentén akár a kép programozással történő módosítását is bemutatni. [130] A multimédiás alkalmazások tanítása előtt vagy annak kezdeti szakaszában kell a lehető legtisztább formában megismerkedni a többi felhasználni kívánt médiaelemmel is, így a karakterek kódolásával, nyelvtani és helyesírási szabályokkal is.

Bár a témaköröknél külön említésre kerül, multimédiás alkalmazások a mai fejlett szövegszerkesztők és a weblapkészítő eszközök is. Érdemes megfigyelni az alapértelmezett dokumentumaik néhány hasonlóságát és eltérését. Mindkettőnek az alapja formázott szöveges dokumentum; a főbb objektumok: karakter, bekezdés, szakasz. Mindkettőben van mód az adatok strukturált elrendezésére, használhatunk táblázatot. A szövegszerkesztővel konstans szélességű és magasságú lapokra tervezzük a dokumentumot, míg weblap készítéskor a dokumentum szélessége a felhasználói oldal által meghatározott paraméter és a lapnak nincs előre megadott hossza. A szövegszerkesztővel készített dokumentum a képeket beágyazva tartalmazza, a weblapon csatolással, azaz külső forrásra hivatkozással adjuk meg a többi médiát. Ráadásul a weblapba videó és hang csatolása is értelmes, míg ezek a szövegszerkesztők esetén, nyomtatásra készülve nem relevánsak. Szövegszerkesztőben az egyes formázási stílusoknak van neve, a stílusok egymásból származtathatók, amit módosít a konkrét lokális formázás. Weblapkészítésben a css jellemzően a meglévő elemekhez definiálja a megjelenést, amely a globálistól a lokális felé, valamint olvasási sorrendben kiértékelve felülírva alakítja ki a megjelenő képet. Informatikai szempontból a kétféle alkalmazással lényegében ugyanazt a tartalmat azonos alapelven, de az eltérő cél miatt különböző megoldásokkal kódoljuk a dokumentumokba.

Az inkább képi megjelenésre optimalizált, médiaelemeket mint objektumokat strukturáltan tartalmazó infografika-készítő és a bemutató-készítő alkalmazások között az időbeliség kezelése a legnagyobb eltérés, ugyanakkor filozófiai – és ezért felhasználási területben is – különbség van a diákra (slide-okra) tervezett bemutató-készítő és a Prezi gondolattérkép jellegű megoldásai között. Animációt és videót bemutató-készítő alkalmazásokkal is lehet készíteni, csak a kézi vezérlés helyett időzítést kell alkalmazni.

Az animációkészítés oktatásával előkészítjük a programozást is. Az objektumok megjelenésének vezérlése, mozgatása, átalakítása lényegében deklaratív programozás, eljárások paraméterezett meghívását jelenti. A blokknyelven programozás csak annyiban ad ennél többet, hogy ott az eljárást vezérlési elemekből kell elkészíteni, míg a bemutatókészítő programokban előregyártott algoritmusokból választhatunk. Emiatt az informatikaoktatásban sokkal nagyobb a szerepe a prezentációkészítésnek, mint a későbbi felhasználási területeken. Az értekezleteken vetített bemutatók nem biztos, hogy növelik a hatékonyságot, mert jellemzően az a szerepe, hogy az előadó gondolkodási sorrendjét rákényszerítse a hallgatókra. De a tanítás/tanulás folyamatban fontos szerepe lehet abban, hogy a diák átgondolja egy folyamat szekvenciális és párhuzamos lépéseit. A szöveg megjelenésének animálása is kifejezhet belső (programozói) szándékot,

de ez viszonylag ritka. A prezentációk jelentős részében a teljes dokumentumra jellemző megjelenítési stílust formázzuk az animációval. Az informatikai gondolkodás fejlesztése szempontjából a „történetmesélés", a gondolatmenet kifejtése fontosabb.

Példák a gondolatmenet prezentálására, algoritmizálására:

- Egy-egy kommunikációs jelenetet bemutató prezentáció – a szereplők helyzetváltozásával, a szövegbuborékok megfelelő sorrendű megjelenítésével – felér egy hétköznapi algoritmus leírásával.
- A tanult algoritmusok (például: összeadás, szorzás) lépésenkénti bemutatása, megjelenítése az elkészítés során tanít, bemutatáskor a tanár ellenőrizheti a helyes műveletvégzést.
- Egy szélkerék forgatásában az elemek csoportba foglalt egységként kezelése; egy hétszegmenses kijelzőn visszaszámlálás megjelenítése az objektum részeinek egyedi párhuzamos kezelése megmutatja az „egyben", illetve „egyidőben" végrehajtott művelet közötti különbséget, fejleszti az objektum szemléletet, a párhuzamos és szekvenciális vezérlést. Ebben a két feladatban megjelenik az időzítés, az adott idejű hatás, a végtelenítés és a következő eseményig tartó lejátszás fogalma is.
- Az animációk választása a bemutatott fogalom ismeretét is mutatják. A kislány nem csak feltartja a kezét, hanem integet; a tűz lobog; a háromszög súlyvonala nem bepörög, hanem a csúcsból a szemközti oldal felezőpontja felé megjelenik; a hal beúszik (fejével előrefelé); a hópihe, a falevél a levegőben táncolva halad lefelé…

A programozás előkészítéseként érdemes megfogalmazni, hogy az egyes animációk az *objektum*ok mely *tulajdonság*ait hogyan módosítják, de informatikai és digitális írástudás szempontjából fontosabb, hogy a gondolathoz leginkább illeszkedő kész *eljárás*t ki tudja választani a diák. Ez a gondolat kódolása, az *algoritmizálás*.

Személyes példa

2002-ben, az ELTE-n Informatika a matematikaoktatásban címmel tartottam gyakorlatot. Itt vettem észre, hogy a leendő matematikatanárok a később tanítandó tételek bizonyítását nem jól prezentálják. A megtanult bizonyítások jellemzője, hogy a megfogalmazásban először állít, majd indokol. Például: A két háromszög hasonló, mert két-két szögük egyenlő. Az animációban először kellene megmutatni, hogy a két-két szög egyenlő, majd mozgatással, nagyítással megmutatni a hasonlóságot. Később, a 9–11. évfolyamon, a prezentáció oktatásába mindig bevettem valamely geometria tétel bizonyítását animációval.

Mindig egyéni projektként, tetszőleges forrásfelhasználással. A beadott munkákból egyértelműen látszott, melyek a meg nem értett összefüggések. A legdurvább hibák téves jelölések, rossz helyre berajzolt vonalak formájában láthatók (ez a matematika dolgozatokból is ismert), de alapvető hiányosság jele a hibás megjelenési sorrend vagy hibás animációtípus alkalmazása. Ezekben az esetekben a jól bemagolt, de nem megértett bizonyítás érhető tetten. A legtöbb diák az 1–3 továbbfejlesztési, javítási lehetőség során rendezi logikai sorba a bizonyítás lépéseit. Ez a folyamat nem más, mint a gondolatsor algoritmikus kifejtése. Ezután már nem csak felmondani tudja a bizonyítást, hanem érti is, amit mond, hiszen egy gépnek is el tudta magyarázni.

A multimédiás tartalmat érdemes a vétel (dekódolás) alapján is informatikai szempontok alapján megvizsgálni. Melyek azok, amelyek a lineáris adatsor közvetítésére alkalmasak (pl.: hang), melyek engednek a feldolgozásra több bejárási utat (pl.: weblap) és melyik médiák adnak a feldolgozás, a dekódolás oldalán szabadságot (pl.: kép esetén a szemlélő fókuszál egy-egy részletre). A médiákat az ember dekódolja. Ennek megfelelően már kódoláskor figyelemmel kell lenni a dekódoló képességeire. A vizuális tartalmak dekódolása sokkal gyorsabb, mint az audióé, kivéve, ha gyengénlátó, esetleg vak az illető. A hang egyedi dekódolása (füldugóval) a közvetlen környezet érzékelését gyengíti, a kakofóniából a szükséges hangok kiszűrése fárasztó és nehézkes. A 3D ábrázolás realisztikusabb, de – főleg mozgó változata – az embert érő vizuális és mozgásérzékelési ingerek ellentmondásos jelzései miatt rosszullétet okozhatnak (tengeri betegség).

A multimédiás tartalmak készítésekor a *hatékonyság*ot több szempontból kell vizsgálni: Az elkészítés hatékonysága a közlési szándék és az elkészítési idő függvényében értelmezhető. A dokumentum tárolásának hatékonysága méretbeli és adatbiztonsági kérdéseket vet fel. A felhasználás hatékonysága a közvetített információ megszerzésének lehetősége, ideje, teljessége alapján jellemezhető.

Bár a témakörön belül nincs nevesítve, a táblázatkezelők is multimédiás alkalmazások; az adatok elrendezett megjelenítése, formázottsága és a diagramok tipikus médiaelemek, az infografikának is részei lehetnek. Az adatbáziskezeléshez rendelt felületek, az űrlapok és jelentések szintén multimédiás dokumentumok. E két alkalmazástípust az teszi különlegessé, hogy a megjelenő szöveges és képi elemek hátterében értékként kezelt adatokkal, a megjelenítést befolyásoló programok, függvények, eljárások működnek.

A digitális írástudás témakör eredménycéljai között szerepel a tartalomkezelő alkalmazás használata is, ugyanakkor számos már ma is használt multimédia alkalmazás és médiaelem típus nincs nevesítve. Nem szerepel digitális jegyzettömb (például OneNote) ismerete és használata. Tekinthető a digitális jegyzettömb is tartalomkezelő rendszernek? Egy kicsit más, inkább egy draft verziója a tartalomkezelőnek. Milyen médiaelem az egyenlet? Az egyenletszerkesztők szöveges alkalmazások? Valószínűleg annak tekintendők, de azért elég különlegesek, főképp, ha matematikai értelmező is tartozik hozzá és például diagramot képes generálni egy függvény-hozzárendelésből. Utolsó példaként, mi a helyzet a kiadványszerkesztő eszközökkel? A tudományos kiadványok LaTeX nyelvű szerkesztői szövegszerkesztők lennének? A kiadványkészítő Publisher a szöveget dobozokban helyezi el, ezzel keverednek benne egy bemutató-készítő és egy szövegszerkesztő tulajdonságai.

A digitális írástudás nem csak azt jelenti, hogy ismerünk és célszerűen fel tudunk használni egy tucatnyi alkalmazást gondolataink, adataink közlésére, hanem azt is, hogy ezeknek az alkalmazásoknak ismerjük a *működési elv*ét: az *adat*ait (*objektum*ait) és *algoritmus*ait. Értjük az alkalmazások működése eltéréseinek az okát és ennek ismeretében *választ*juk meg a közlés, tárolás és feldolgozás szempontjából legmegfelelőbb alkalmazást, multimédia elemeket és *megoldási módszerek*et. A digitális írástudás képessé tesz arra, hogy új alkalmazások megjelenésekor annak működési elvét – a korábban megismertekkel összehasonlítva – *önállóan* felderítsük, *hatékony* használatát elsajátítsuk.

6.2.3

IX/4. Problémamegoldás informatikai eszközökkel és módszerekkel (3.)

A téma címe alapján a tantárgyon belül itt oktatunk informatikát. Valójában a témakörön belül az ember-számítógép kommunikáción van a hangsúly. Középpontban a gép vezérlése, a gépen tárolt adatok feldolgozása áll. Informatikatudományra fókuszálva, a programozás különböző módszereinek az alapjai szerepelnek ebben a témában: imperatívan programozás, deklaratívan adatbázis-kezelés, az objektum orientált és esemény-vezérelt programozáshoz leginkább a robotika nyújt több-kevesebb alapot. A táblázatkezelők használata a funkcionális programozás gondolkodásmódját igényli, adatkezelése, megoldásai a programozást több szempontból modellezik. Ezért a programozással kapcsolatos ismeretek a táblázatkezelés oktatása során előkészíthetők. Erre mutattam néhány példát a How to Teach Programming Indirectly – Using Spreadsheet Application [133] cikkemben. Az oktatási gyakorlatban használható példák elméleti rendszerbe foglalásával bemutatható, hogy a táblázatkezelés a programozás általános elő-

készítő alkalmazása lehet, ahogy ezt a Using the Spreadsheet Paradigm to Introduce Fundamental Concepts of Programming to Novices [134] cikkben olvashatjuk. Bár eszerint az elemzés szerint a ciklus megvalósítása nincs meg a táblázatkezelőkben, a képletek másolása egész közel van a ciklusképzéshez. A tömbfüggvények használatával pedig egy képletbe is foglalhatjuk a műveletek ciklikus elvégzését. A SPREGO [135] gondolkodási módszer a táblázatkezelő eszközeit programozási nyelvként használja, így képez kapcsolatot a táblázatkezelés és programozás oktatás között. A láthatóan szoros kapcsolat miatt a programozást előkészítő blokkprogramozás és a robotika mellett az egyik legáltalánosabb irodai alkalmazásnak, a táblázatkezelésnek is helye van a problémamegoldás eszközei között.

Az adatbáziskezelő alkalmazások a táblázatkezelőknél is jobban hasonlítanak a programok fejlesztő környezetéhez. Két nyelv, az adatdefiníciós és a lekérdezőnyelv kódjai futnak a kattintgatásokkal vagy szövegesen kiadott utasítások hátterében. Bár az eredménycél minimumok között nem szerepel az SQL nyelv ismerete, a feladatokat lekérdező-rácson (QBE felületen) kiadva, a blokknyelvekhez hasonlóan programozásról beszélhetünk.

Az alábbiakban a fejlesztendő területek alatt az egyes eredménycélok kulcs kifejezéseit gyűjtöttem ki. A táblázat- és adatbáziskezeléssel kapcsolatos eredménycélok egy része funkcióját tekintve a digitális írástudáshoz tartozik (ezeket kihagytam). Ami ezen felül olvasható, az szoros kapcsolatban van az algoritmizálás, programozás témákkal, sőt, a „programozás” fogalom értelmezésétől függően, tekinthetjük deklaratív nyelven történő programozásnak.

Alsótagozaton

- *A probléma megoldásához szükséges módszerek és eszközök kiválasztása*
  - *…értelmez néhány egyszerű problémát, megoldási lehetőségeket játszik el, fogalmaz meg, valósít meg … információkat keres és használ fel …*
- *Algoritmusok vizsgálata, előállítása*
  - *Felismer, eljátszik, végrehajt … elemi lépésekből álló, adott sorrendben végrehajtandó cselekvést; … algoritmust elemi lépésekre bont, értelmezi a lépések sorrendjét, megfogalmazza az algoritmus várható kimenetelét …*
- *Kódolás, folyamatok irányítása, a robotika alapjai*
  - *Az eszköz mozgását értékeli, hiba esetén módosítja a kódsorozatot … kódsorozatot tervez és hajtat végre, történeteket, meserészleteket jelenít meg az eszközzel, például padlórobottal.*

Felsőtagozaton

- *Egyszerű algoritmusok elemzése, készítése*
  - *Értelmezni képes az algoritmus végrehajtásához szükséges adatok és az eredmények kapcsolatát; megkülönbözteti, kezeli és használja az elemi adatokat.*
- *A kódolás eszközeinek ismerete, a blokkprogramozás építőelemeinek használata*
  - *Probléma megoldásához vezérlési szerkezetet alkalmaz blokkalapú programozási nyelven.*
  - *Szereplőközpontúan mozgásokat vezérel virtuálisan (képernyőn) és valóságban (például robottal).*
- *Adatok kezelése (Táblázatkezelés)*
  - *Cellahivatkozásokat, matematikai tudásának megfelelő képleteket, egyszerű statisztikai függvényeket használ.*
- *Tantárgyi problémák vizsgálata digitális eszközökkel*
  - *... problémákat old meg táblázatkezelő program segítségével. ... hétköznapi jelenségek számítógépes szimulációja ... a szabályozó eszközök hatásai.*

Középiskola

- ***Algoritmizálás**, módszerek, eszközök használata, típusalgoritmusok*
  - *... elemi adattípusok ...: egész, valós szám, karakter, szöveg, logikai; ... egy algoritmus-leíró eszköz ... típusalgoritmus…*
- ***Programozási nyelv** fejlesztői környezete, vezérlési szerkezetek*
  - *... programozási nyelv fejlesztői környezetének alapszolgáltatásai; ... szekvencia, elágazás és ciklus segítségével egyszerű algoritmust létrehozni, és azt egy formális programozási nyelv segítségével megvalósítani; ... feladat megoldásainak helyességét tesztelni.*
- *Adatkezelés **táblázatkezelő** alkalmazással*
  - *... táblázatkezelővel adatelemzést és számításokat végez; ... függvényeket célszerűen használ ... nagy adathalmazokat kezel; az adatokat diagram segítségével szemlélteti.*
- *Adatkezelés **adatbáziskezelő** rendszerrel*

- *... adatbázis-kezelés alapfogalmai; keres, rendez és szűr; adatokat visz be, módosít és töröl; űrlapok..., jelentések...*

– *Számítógépes szimuláció*
  - *... használja a hétköznapi, oktatásához készült szimulációs programokat; ...kezdőértékek változtatásának hatásai...*

Az informatikaoktatás tartalma szempontjából azt érdemes megvizsgálni, hogy a témakörön belül az egyes programozással kapcsolatos fogalmak hogyan jelennek meg, hogyan fejlődnek. A korábban tárgyalt témakörökhöz képest itt sokkal hangsúlyosabban jelenik meg, hogy valamilyen formában mindig programozást, az ehhez szükséges fogalmakat tanítjuk vagy fejlesztjük a programozáshoz szükséges készségeket.

Adat

Az adat absztrakciója az informatika tanulásának kezdetén elég elnagyolt, lényegében médiaelem. Virtuális entitásokat használ a gyerek. Ezek részletesebb megismerése, jellemzése, leírása során alakul ki az adattípus, illetve az objektum fogalma. A robotika alapjainak tanulása során eszközöknek egyes tulajdonságait kell beállítani, egy véges listából választva. A táblázatkezelő kezdetben csak az adatok elrendezését, strukturálását segíti, de már az első pillanatban érdemes tisztázni a szöveg és a szám megjelenésének eltérését, azt, hogy a karaktersorozatként beírt adat átalakulhat, ilyenkor nem lóg túl a cellahatáron, és jobbra igazított lesz. Később tisztázni kell, hogy az ″123″ szöveg, a 2020. február 2. viszont szám jellegű, értékkel rendelkező adat. A táblázatkezelők használatához szükséges ismeret, hogy a számítógép digitálisan működik és az adatokat alkotó jeleket binárisan tárolja. Ismerni kell a kettes és tízes számrendszerek közötti váltást és azt is, hogy a számokat nem tizedestört, hanem kettedes tört alakban értelmezik a gépek. Bőséges tapasztalattal kell rendelkezni arról, hogy a táblázatkezelő adatértelmezője gyengén típusos, az automatikus adatkonverziót megadott formátumokra illeszkedés alapján végzi.

Az első probléma a tizedespont dátumelválasztóként értelmezése szokott lenni, de alapismeret lenne az is, hogy a százalékjel százzal osztja az eléje írt számot, a 'Ft' viszont csak megjelenített mértékegység, nem társul hozzá árfolyam (50% értéke 0,5; a 10 Ft, illetve 20 € tartalmú – és formátumú – cellák összege szoftvertől és a tagok sorrendjétől függően 30 Ft, 30 € vagy 30 is lehet). Táblázatkezelésoktatás részeként értelmezzük a szöveg, a szám (érték), a logikai, a hivatkozás és a tömb adattípust, az adatbáziskezelés programozási nyelvei típusosak, ami a fogalmak további pontosítását és tudatos alkalmazását tennék lehetővé.

A robotika lehetőséget ad arra, hogy az emberi adatbevitel helyett szenzorokról származó inputtal, és az eszközök állapotváltozását eredményező outputtal dolgozzon a tanuló. Ez a fizikai kapcsolat segíti a virtuálisan létrehozott objektum absztrakcióját. A legegyszerűbben érzékelhető inputok (kapcsolók) és vezérelhető outputok (LED-ek) programozásával a processzor működési elvének megértéséhez is közelebb juthat a tanuló. Ezért a robotika témakörben nem csak a számítógéppel (táv)vezérelt robotok működését kell tanulmányozni, hanem a közvetlen, mikrokontrolleres vezérlést, a beágyazott rendszerek működését is, illetve a processzor működésének absztrakciójához a digitális technika alapjait, a logikai kapuk működését is célszerű tanulmányozni.

A programozás tanításakor az ebben és a többi témakörben korábban szerzett tapasztalatokra lehet alapozni a változó, a különböző adattípusok és az osztály, illetve az objektum fogalmát, amelyet a programozási nyelv tanításakor tudunk kreatívan használhatóvá tenni: adatot tudatosan létrehozni, módosítani, megszüntetni. A diáknak az adat több absztrakciós szintet kell megértenie.

Példák az adattal és objektummal kapcsolatos fogalmak kialakításához alkalmas témákra

- Rasztergrafika: pixel, táblázat, kép; a pixel objektum tulajdonságai
- Vektorgrafika: alakzatok, alakzatok tulajdonságai, alakzatok átalakítása
- Szöveg, dokumentum objektumai: karakter, bekezdés, szakasz; teljes dokumentum
- Táblázat, tábla, sor, oszlop, cella, rekord, mező, adat, hivatkozás, kulcs (kapcsolat)
- Szereplő, karakter, profil, sprite, metódus, függvény

Amellett (vagy ahelyett), hogy például a „digitális írástudás" keretei között tanítjuk az alkalmazások használatát, az alkalmazások informatikai vonatkozásait kellene tanítani. Ezen belül az adat, az objektum és hozzá kapcsolódó fogalmakat is vizsgáljuk: aggregáció, kompozíció, asszociáció, tulajdonság, művelet, eljárás függvény; tulajdonságok alapértelmezett értéke (default beállítások); objektum (adat) létrehozása, paraméterezése, törlése, tulajdonságainak módosítása és módosulása, másolása, közvetlen és közvetett elérése.

A múlt században születettek számára az egyszerű adat, a karakter, az egész szám volt a tanulás kiindulási pontja, ebből építette fel a szakember az objektumokat. Az információs társadalom generációi objektumokon keresztül ismerik meg a digitális világot, ezért az oktatás során az objektumok elemző (informatikai) megismerése a kiindulási pont, innen kell – kezdetben szakmai megnevezések nélkül – a különböző elemi és összetett adatok fogalmait megismerniük.

Elemi adatok

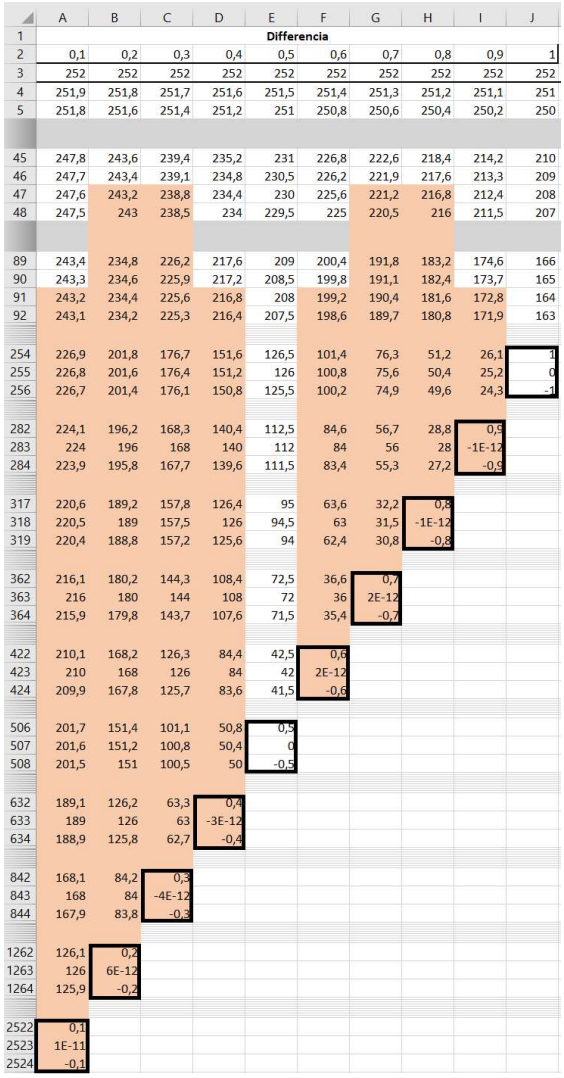
Az informatikatudomány újraértelmező természete itt is megmutatja magát. Az objektumorientált nyelvekben az integer is objektum. Minden objektum. De, hogyan értelmezhető az elemi adat? Az elemi adat tárolása csak egy konkrét érték tárolását jelenti, amelyre a műveletek gépikódban, processzorművelet szintjén definiáltak. A processzor elemi adatokkal dolgozik, az ALU az elemi adatokon tud műveleteket végezni. Ezért az adat fogalmának megértéséhez szükséges a robotika és ezen keresztül a digitális technika alapjainak is az oktatása. Ismeret szintjén meg kell teremteni a kettes számrendszer, a bitműveletek, az aritmetikai és logikai műveletek és a fizikai eszközök (szenzorok, inputok és outputok) absztrakciók kapcsolatát.

Amikor matematikából a diák elkezd a folytonosság és végtelen fogalmával ismerkedni, ideje alaposabban megismernie informatikából a digitalizálás jellemzőit. A matematikai kerekítés szabályaival egyidőben kell megmutatni a mintavételezés és kvantálás tulajdonságait. Matematikaoktatásban a kerekítés oka a végeredmény nagyságrendi megjelenítése, informatikából az adat létrehozásának, vagy mérésnek az eredménye, a valósághoz közelítő érték. Miközben matematikában – formális jelölésekkel – pontos számítások elvégzését tanulja a diák, meg kell mutatni, hogy a digitális világban mely műveletek eredménye pontos, a kvantálás hogyan befolyásolja az adatokkal végzett számítások eredményét. Tisztázni kell a matematikában tanult számhalmazok és az informatikában használt adattípusok közötti kapcsolatokat és (főleg) különbségeket. Példákon keresztül meg kell mutatni a digitális adatkezelés használhatóságának okát: Ismert pontatlanságú, ellenőrzötten értékálló adatok jobban használhatók, mint az elvileg végtelenül pontos, de ellenőrizhetetlen, összehasonlíthatatlan eredmények.

Példák digitális tárolás értelmezésére:

- Számtani sorozat rekurzív számítása során a digitális tárolás kerekítési hibája összeadódik. Alkalmazói programtól, fordítóprogramtól függ, hogy ez a kerekítés mikor lesz kimutatható. A 36. ábra azok a cellák vannak feltételes formázással kiemelve, amelyekben a cellaérték nem egyezik meg az explicit módon megadott értékkel ($a_n = a_0 – n \cdot d$).
- Hasonlóan, a 36. ábra sorozatainál megvizsgálható, hogy hányadik lépésben éri el a sorozat a 0-t. A választott kiindulási értékkel (252), differenciaként minden tizedre a [0,1; 1] tartományból a 0 érték is a sorozat tagja. Látható – programot írva rá tapasztalható –, hogy a lépésenkénti csökkentés nulla helyett sokszor a kerekítési hibát írja ki. A nem egész (azaz lebegőpontos, valós) számábrázolás esetén a tárolt jegyek

száma véges. A kerekítési hiba az eredmény nagyságrendjének csökkenésével valódi értéknek látszik.



	A	B	C	D	E	F	G	H	I	J
1					Differencia					
2	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
3	252	252	252	252	252	252	252	252	252	252
4	251,9	251,8	251,7	251,6	251,5	251,4	251,3	251,2	251,1	251
5	251,8	251,6	251,4	251,2	251	250,8	250,6	250,4	250,2	250
45	247,8	243,6	239,4	235,2	231	226,8	222,6	218,4	214,2	210
46	247,7	243,4	239,1	234,8	230,5	226,2	221,9	217,6	213,3	209
47	247,6	243,2	238,8	234,4	230	225,6	221,2	216,8	212,4	208
48	247,5	243	238,5	234	229,5	225	220,5	216	211,5	207
89	243,4	234,8	226,2	217,6	209	200,4	191,8	183,2	174,6	166
90	243,3	234,6	225,9	217,2	208,5	199,8	191,1	182,4	173,7	165
91	243,2	234,4	225,6	216,8	208	199,2	190,4	181,6	172,8	164
92	243,1	234,2	225,3	216,4	207,5	198,6	189,7	180,8	171,9	163
254	226,9	201,8	176,7	151,6	126,5	101,4	76,3	51,2	26,1	1
255	226,8	201,6	176,4	151,2	126	100,8	75,6	50,4	25,2	0
256	226,7	201,4	176,1	150,8	125,5	100,2	74,9	49,6	24,3	-1
282	224,1	196,2	168,3	140,4	112,5	84,6	56,7	28,8	0,9	
283	224	196	168	140	112	84	56	28	-1E-12	
284	223,9	195,8	167,7	139,6	111,5	83,4	55,3	27,2	-0,9	
317	220,6	189,2	157,8	126,4	95	63,6	32,2	0,8		
318	220,5	189	157,5	126	94,5	63	31,5	-1E-12		
319	220,4	188,8	157,2	125,6	94	62,4	30,8	-0,8		
362	216,1	180,2	144,3	108,4	72,5	36,6	0,7			
363	216	180	144	108	72	36	2E-12			
364	215,9	179,8	143,7	107,6	71,5	35,4	-0,7			
422	210,1	168,2	126,3	84,4	42,5	0,6				
423	210	168	126	84	42	2E-12				
424	209,9	167,8	125,7	83,6	41,5	-0,6				
506	201,7	151,4	101,1	50,8	0,5					
507	201,6	151,2	100,8	50,4	0					
508	201,5	151	100,5	50	-0,5					
632	189,1	126,2	63,3	0,2						
633	189	126	63	-3E-12						
634	188,9	125,8	62,7	-0,4						
842	168,1	84,2	0,3							
843	168	84	-4E-12							
844	167,9	83,8	-0,3							
1262	126,1	0,2								
1263	126	6E-12								
1264	125,9	-0,2								
2522	0,1									
2523	1E-11									
2524	-0,1									

36. ábra:
$a_0 = 252$; $a_n = a_{n-1} - d$; $d = 0{,}1 \cdot e$; $e \in \mathbb{N}$, $e \leq 10$
színes kiemelés: $a_n <> a_0 - n \cdot d$; fekete keret: „zérushely” környéke

- Nem lehet feltétel a sorozatos összeadások hibáinak összegzésekor a tagok egyenlősége. Bármilyen lebegőpontos számok összegzése magában hordozza a kerekítési hibák összeadódását.
- Valós (gyakran racionális) számokkal operáló matematikai azonosságok sem ellenőrizhetők számítógéppel. A 2/GYÖK(2) = GYÖK(2) kifejezés értéke HAMIS épp úgy, mint a SIN(RADIÁN(45)) = GYÖK(2)/2 is. Az egyenlőség igazolása csak fordított irányban történhet: a számítógép által kijelzett egyenlőnek látszó értékekről matematikai úton igazolhatjuk, hogy valóban egyenlőek. Matematikai egyenlőség akkor

is lehetséges, ha a számítások (amelyekben kerekítéseket alkalmaztunk) alapján a két mennyiség nem egyenlő.

- Ha az alkalmazásban a dátum-idő egysége a nap, akkor az időpontokkal történő számolás a fentihez hasonló problémákat jelent. Például 8:00 után 8-szor 0:15 (negyedóra) nem biztos, hogy pontosan 10:00-val lesz egyenlő.

Nem tananyag, de tantárgyi kooperációban érdekes informatikai téma annak vizsgálata, hogy

- a világ digitális-e vagy analóg;
- képesek vagyunk-e analóg mérésre;
- „folytonos" vonallal rajzolt függvényeink minden esetben diszjunkt ponthalmazok;
- egy fénykép nagyításával lehet-e további részleteket megjeleníteni, adatokat kinyerni;
- esetleg: mi a Plank-idő és a Plank-hossz.

Összetett adatok

Visszagondolva tanulmányaimra, a sorozatokat – és ezzel kapcsolatban az indexelést – körülbelül 16 éves koromban tanultam először. Ezután 5 évvel, 21 évesen tanultam programozni BASIC-ben, ahol a számokból vagy karakterekből álló kétdimenziós táblázat volt a legbonyolultabb adatszerkezet. Az első kezelendő összetett adat a szöveg volt, ezt követte az egészek tömbje. 30 évesen, számítástechnikatanári kiegészítő szakon tanultam a struktúrákról és egy icipicit az objektumokról Pascalban. 34 évesen készítettem az első Windows Form alkalmazást, Visual Basicben, újabb 8 év, mire megírtam C#-formon egy önállóan tervezett alkalmazást, de csak 2 évvel később írtam osztálydefiníciót feladatok megoldásában.

Annak, aki lényegében születése pillanatától grafikus képernyőt használ, az első (összetett) adattípus, amivel megismerkedik, az objektum. Ikonokra bök rá, alakzatokat hoz létre, képek tulajdonságait változtatja meg. Ezt használják ki, a padlórobotok és blokknyelvek is. Az objektum fogalmának kibontásával kapjuk, hogy a tulajdonság lehet rész, azaz egy másik objektum, vagy begépelhető adat (számok, betűk). Az autónak van kereke, a keréknek van színe… Az első alkalmazásokban egyedi objektumokat kezel a gyerek, majd néhány objektumot, de egyedileg tartja számon.

30 évestől kb. 42 éves koromig szedtem össze azt a tudást, amit a mai diákoknak 5–12 éves korban kellene tanulniuk az objektumokról. Azóta természetesen változott a környezet is, de a tanítás során figyelembe kell venni, hogy sokkal fiatalabb korban várunk el bizonyos absztrakciókat, a tanítványok másképp tapasztalják meg a körülöttük levő világot, mint a tanáraik.

A tanterv alapján a 12 éves diák – több éves gyakorlattal az objektumok használatában – tanul táblázatkezelést, megtanulja a cellahivatkozást. A pixelgrafikus rajzolással foglalkozva tapasztalatot szerez a koordinátákkal kapcsolatban. Lényegében érti a kétdimenziós táblázat absztrakciót. A sorozatok matematikából 8. évfolyamon tananyag, így 14 évesen már értheti, hogy mit jelent az, hogy $a_7 = 30$. Ez számára a függvényhez kapcsolódik, így csekély köze van ahhoz, hogy a szöveg karaktersorozat, amiben az egyes karaktereket indexükkel érjük el és megvizsgálhatjuk, hogy a ″Hello″[4] == 'o' igaz-e. Az informatikában tömbnek nevezett adott elemszámú, azonos típusú elemekből álló adatsorozat absztrakciójára ezért külön figyelmet kell fordítani. Korábbi ismeretekből, tapasztalatból nem következik, hogy számjegy gombokhoz, kártyapaklihoz vagy akadályok listájához az adott típusú objektumokból álló tömböt először üresen kell létrehozni (mint egy fiókosszekrényt, fióktartó sínekkel), majd ebbe az egyes elemeket – objektumokat is létre kell hozni (el kell készíteni a fiókokat és beletenni) és a megfelelő megjelenéshez az objektumok kezdeti tulajdonságait is meg kell adni (azaz a fiókba bekerülnek az adatok). Könnyebb a tömb absztrakciója, ha a blokknyelven szerzett tapasztalatok helyett a többi tanult alkalmazásból, leginkább táblázatkezelésből veszünk példákat. Azért is fontos, hogy táblázatkezelésben az adattípusokra kitérjünk, mert ott használunk adott méretű, egész számokat tartalmazó tömböt, illetve a szöveget karaktersorozatként, aminek van hossza, meg lehet adni egyes részeit.

A táblázatkezelők alkalmasak a tömb méretezésének bemutatására. Excel 2016 esetén kiszámolható a „teljesen feltöltött" táblázat fájlmérete. Például 256 munkalap minden cellájába 1 karaktert beírva a fájlméret 1 TiB körüli érték lenne. A táblázatkezelők vagy eleve maximalizált sor és oszlopszámot, munkalapszámot és szöveghosszt engednek, vagy a rendelkezésre álló memóriától függ a bővítés engedélyezése. Excelben egy munkalapon a sor és oszlopszám rögzített, a sor beszúrása nem jelent valódi új sort. Google Sheets-ben dinamikusan növelhetjük a sorok számát – egy ideig. A statikusan lefoglalt maximális méret, illetve dinamikusan változó méret előnyei és hátrányai ezeken a példákon jól szemléltethető. Képlethivatkozások másolásával a tömbök alul- és túlindexelésére, az ebből adódó problémákra is szemléletes példákat adhatunk.

Az adatbáziskezelők a tanult alkalmazások közül azzal tűnnek ki, hogy az adatokat felhasználás közben is a háttértáron tárolják, az adatelérés címszámítással történik. Ezzel szemben a tanult alkalmazásokra az jellemző, hogy a dokumentumokat egy folyamként kezelik, ahogy az I/O eszközökről érkező, oda küldött adatokat is. Az IoT és az egyre jellemzőbb online közös

munka szükségessé teszi a fájl fogalmának értelmezését, foglalkozni kell az adatok integritásának problémáival, a párhuzamos hozzáférés kezelésével. Az egyes alkalmazásokban gyakorlati tapasztalatot kell szerezni a többszörös hozzáférés szabályairól.

Példák fájlkezelés, adatkezelés szabályozására

- Jegyzettömb a dokumentumot a memóriába betölti, megnyitáskor minden megnyitott példány írható, mentéskor az aktuális példány felülírja a létezőt.
- Asztali Excel az xlsx fájl első példányát írhatóan nyitja meg, a további megnyitásoknál figyelmeztet és csak olvasható lesz a fájl. Ennek módosítása után más néven lehet menteni. Azonos gépen azonos nevű fájlokat nem lehet egyszerre megnyitni.
- Access adatbázisokat első lépésként el kell menteni, csak ezután lehet táblát definiálni. Ha a tábla tervező nézetben van megnyitva, akkor nem lehet benne adatot módosítani, adatot beírni. Lekérdezésekkel (a nem szerkesztett táblában) egyszerre többen dolgozhatnak, a zárolás rekord szintű.
- Online Excel fájlokat egyszerre többen is írhatják, látszik, hogy melyek az éppen szerkesztett cellák. A cellákban az érvényes adat az utoljára befejezett szerkesztés eredménye (mint a Jegyzettömb esetén).
- Érintőpad és csatlakoztatott egér együttes használata esetén milyen az egérmutató mozgása.
- Képernyőre kiírás (kimenet és log egy helyre vagy többszálú program esetén) lehet védett művelet – a megkezdett kiírás befejezéséig a következő kiírás várakozik – vagy ömlesztett, a két kiírás karakterei keverednek a képernyőn.

Hivatkozás, indirekció

A hardvernél tanult adatbevitel „gépbe” történik, a programozás során egy szoftver fogadja az adatokat azon belül függvények kaphatják meg, változók adhatják át egymásnak. A programozás tanulása során egyre hangsúlyosabb szerepe van az adat megszerzési módjának is: az adat átkerül (mozog), átmásolás útján vagy az eredeti helyére hivatkozással jut az adott eszköz (gép, program, függvény, változó) birtokába. A programozás tanításakor nem csak a programkészítést, alkalmazás készítését tanítjuk, hanem informatikát is, adatabsztrakciókat, adatmodelleket.

Példák a programozáson belül az informatikai tudás fejlesztésére:

- A multimédiás dokumentumokba a médiák beágyazása, illetve csatolása (hivatkozás).
- A táblázatkezelőben az adatok másolatát másolással vagy hivatkozással hozhatjuk létre.

- A függvény paramétereibe meghíváskor argumentumként az eredeti adat másolata vagy az adat címe kerül.
- A képernyőn megjelenő fájlhivatkozáson vagy ikonon keresztül közvetlenül érhető el a fájl, vagy a hivatkozás csak egy kapcsolat a fájl felé.
- Egy .lnk fájlra parancsikon készítésekor létrejövő .lnk fájl az elsőnek másolata lesz, vagy rá mutat? (A pointer másolata vagy a pointerre mutató pointer?) Lehet-e .lnk fájlokkal körbe hivatkozni?

Mindegyik esetben vizsgálandó, hogy a kapott adat módosítása az eredeti adatra milyen hatással van, megszűnhet-e (törlődhet-e) az átadás során az eredeti adat, az eredeti adat módosulása a használat közben kihat-e a kapott adatra, illetve hogyan befolyásolja az adat használata az eredeti adat használhatóságát, létét. Azaz, jelen példákban, a programozás tanítása során meg kell vizsgálni a másolással, az áthelyezéssel és indirekcióval kapcsolatosan a hitelesség, az adatbiztonság, a hatékonyság érvényesülését.

A hivatkozások minden formájánál kiemelt szerepe van az abszolút és relatív fogalmak értelmezésének. A fájl elérési útvonalának abszolút és relatív megadása évtizedekkel ezelőtt csak DOS-os fájlkezelési feladat volt, de napjainkra a multimédiás és online alkalmazások használatának természetes része a médiaelem globális helyének, illetve a hivatkozó objektumhoz viszonyított helyének a megadása. Táblázatkezelés tanulása során új értelmet (és felhasználási környezetet) kap ez a két fogalom, amely nem csak az adott objektum másolásakor vagy áthelyezésekor lesz releváns, hanem szabályok, számítások örökítésekor is, a „csináld ugyanígy” utasítás pontosításaként. Az adatbáziskezelés és programozás során az adatstruktúrákat alkotó adatok kapcsolata, az adatsorozatok, adathalmazok képzése a két fogalomnak újabb absztrakcióját igényli részben a szelektor, részben az indexelés, részben az összetett struktúrák bejárása kapcsán.

Programozáson belül haladó szintűnek számít, de adatbáziskezelésből alapvető ismeret a táblák kapcsolása, az idegenkulcsok használata. Ezzel egy összetett adatra (struktúrára) hivatkozunk egy egyszerű adat bejegyzésével. Ez az egyszerű adat programozásban lehet egy tömb indexe vagy egy objektum címe, esetleg egy pointer. Egyszerű adatok (hivatkozások) használatával relációkat adunk meg, ezek mentén indirekciók sorozata vezet például annak a kérdésnek a megválaszolásához, hogy „milyen a barátom autója üléshuzatának a színe”.

A relációkhoz hasonló „ereje” van a tömbökben az indexnek. Egyfelöl, a logaritmikus keresésnél is gyorsabb, ha a keresett adatnak tudom az indexét, mert címszámítással azonnal elérhető. Egy összetett adat egyik tulajdonsága alapján egy másik tulajdonság megadása (pl. „ki volt a leggyorsabb” kérdés esetén a sebességhez a név kikeresése) leghatékonyabban akkor adható meg, ha a tulajdonság (sebesség értéke) helyett annak indexét ismerjük. Hasonló megfontolásból, matematikában a függvények zérushelyeit, szélsőértékhelyeit és inflexiós pontok helyeit adjuk meg.

Algoritmus, értelmezés, számítás

A hagyományos programozásoktatás az algoritmusokra fókuszált, ezzel együtt a módszeres, procedurális, imperatív programozás jelentette a tananyagot. Az adatok végletekig (amennyire lehet integer típusra) történő absztrakciója, a matematikai problémák túlsúlya miatt az algoritmusok és programozás sokak számára száraz, nehéz tudomány volt. Az OOP, a robotika fejlődése, a grafikus megjelenítés, a webprogramozás és az IoT előretörése érdekesebbé tette a programozást, egyre könnyebb működő programot készíteni, de ez nem igényli algoritmusok kigondolását, a hagyományos értelemben vett módszeres programozást. Egy webes játék elkészítésének didaktikai elemzése [60] megmutatja, milyen „kerülő” utat jelent az algoritmusok oktatásában, ha először el kell készíteni az objektumokat. Hasonló eredményre vezet a padlórobotok programozása és – tapasztalatom alapján – minden OOP alapú játék- vagy programfejlesztésen keresztül a programozásoktatás. Az OOP lényege, hogy a programban szereplő objektumok önmagukon belülről vezérelve változtatják állapotukat. Így például a sorozatszámítás algoritmusának használata szinte hiba: nem „megyünk végig” az objektumokon, hogy kiszámítsunk mindegyikre egy újabb értéket, hanem minden objektum magában hordozza a kiszámítási módot, amit szükség esetén elvégez. Ez a gondolat látszik a VI/2 melléklet Eratosztenészi-szita példájának OOP megoldásán. Nagyon gyakori, hogy az OOP alapokon írt játékprogramokban a „szereplők” száma korlátos. Lényegében a játéktér kezelésével és egy-két szereplővel, avagy egy padlórobottal rengeteg érdekes játékot lehet készíteni úgy, hogy csak egy-egy vezérlési struktúrát használunk egy-egy függvény megírásakor. A játékhoz a mozgást állapotok átmeneteként adhatjuk meg egy óra segítségével, amelynek minden tick eseményére létrejön az előző-állapotból számított új állapot. Az állapotátmeneteket az egyes tulajdonságok újraszámításával lehet megadni, így a „főprogram” vagy egyszerű értékadások, vagy egyszerű feltételek teljesülésétől függő értékadások szekvenciája.

A fentiek némiképp korlátozott funkcionalitással egy animációkészítő (prezentációkészítő) alkalmazással megvalósíthatók. A blokkalapú programozási nyelv lehetőséget ad az állapotátmenetek finomhangolására, közvetlenebb vezérlésére, ami újdonságként jelenik meg a 2020-as tantervben, mint a felsőtagozat egyik eredménycélja. Az előző tantervekhez képest nagy változás, hogy a programozásoktatást OOP alapokon kezdhetjük, ahol az objektum előregyártott elem. Algoritmusokat már alsótagozaton is tanítunk, alsóban a szekvencia tananyag, felsőtagozaton ehhez társul az elágazás és a ciklus. Hagyományosan az algoritmizálás oktatását a „teafőzés" vagy hasonló hétköznapi algoritmussal vezetik be, amiről [121] cikkemben mutattam be, hogy az algoritmusban használt „adat" pontatlan definíciója miatt nehezebb az adat és algoritmus fogalmának, illetve ezek kapcsolatának megértése. A tantervben szereplő padlórobotok és a blokknyelvű programozás, a fizikailag megépíthető (például LEGO) robotok ezt a problémát kiküszöbölik, mert létükkel az objektum (az adat) definíciója is egyértelmű lesz. A hétköznapi algoritmusok tanításához mintaként tekinthetjük egy létező (vagy elkészíthető) robot, illetve objektum tulajdonságait, metódusait.

Az algoritmizálás tanítását a középiskolában a NAT szerint „típusalgoritmusok", illetve „egyszerű algoritmusok" tanítása követi. Ezek az algoritmusok már elemként használják a vezérlési struktúrákat, lényegében az egyszerű (elemi) programozási tételek (algoritmusminták) tanítását jelentik, amelyek sorozatokkal kapcsolatos műveleteket végeznek. A hétköznapi algoritmusok még pontos adatdefiníció esetén is nehezen használhatók ezeknek az algoritmusoknak az oktatásához, mivel az ember a mindennapi gyakorlatban nem úgy oldja meg a problémát, ahogy a gépek. Például egy csoportból a legmagasabb ember kiválasztásához nem kezdjük el elölről, sorban haladva nézni a csoporttagokat. Inkább úgy szemléltetném a kiválasztást, hogy fentről leeresztek egy lapot és akinek először eléri a fejét, az a legmagasabb. Hasonlóan a szókét sem sorban haladva keresnénk, hanem „szkenneléssel", ránézve a csoportra észrevennénk.

OOP-vel nem könnyű olyan érdekes feladatot adni, amelyekhez egyszerű algoritmusok használata is kell, mivel ezek jelentős része statisztikához kapcsolódik. Tovább nehezíti a helyzetet, hogy a tanterv alapján vizuális blokkokról mindeközben át kell térni a karakteres kódolásra. OOP program esetén a programozási tételek alkalmazása előtt rengeteg előkészület kell. Ezt egy vizuális fejlesztőfelületű blokk nyelv esetén kattintásokkal meg lehet oldani, de a keletkező kód az osztálydefiníciók és objektumok létrehozása miatt hosszú; sok, egymásra is hivatkozó részből áll, aminek áttekintése nagyon nehéz. Ezen a ponton a programozási tételek alkalmazása előtt az előregyártott osztályok használata helyett meg kellene tanulni az objektumorientált szoftvertervezés alapjait, ami nem része a közismereti tananyagnak. Ráadásként

az iskolaváltás miatt egy csoporton belül nagy valószínűséggel különböző előismeretekkel rendelkező diákoknak kellene a tanult blokknyelvből karakteres kódolásra átállni.

A gyerekek számára fejlesztett blokknyelven történő programkészítés kódolás irányában történő továbbvitele valószínűleg nem vezet messzire, mert a nyelv a látványra optimális és nem a hatékonyságra. Nagyobb számításigényű programok, például nagy prímszám keresése, rendkívül lassan futnak. Természetesen a blokknyelvek fejlődésével elérhető, hogy a karakteres kódolással azonos hatékonyságú programot készítsünk ezen a felületen is, de ezzel együtt a használható blokkok száma is jelentősen nő, ami miatt a megfelelő blokk kiválasztása a listából nehezebb lesz. Keresés, szűrés hozzáadása viszont épp egy kódrészlet begépelését jelenti, ezért ergonómiai megfontolások miatt elkerülhetetlennek látszik a kódolás.

Egy OOP alapú általános célú nyelv, és ehhez egy grafikus felület kezelését is biztosító fejlesztő környezet megfelelő átmenet lehet a blokknyelv és kódolást támogató nyelvek között, mert ebbe át lehet emelni a blokknyelven tanultak OOP-vel kapcsolatos részét, ugyanakkor a kódolás már karakteres, megtanulható az egyes blokkoknak megfelelő kódnyelvi elem az adott környezetben. A grafikus fejlesztőkörnyezetben használt nyelv azonos lehet azzal, amit a konzolalkalmazásokban használunk. Az átmenetet segíti, ha a blokknyelvnek van kódnézete és az hasonló – a vezérlési struktúrák szintjén azonos – szintaktikájú, mint az új nyelv. Ebben a környezetben néhányórás átvezetéssel át lehet térni a konzol alkalmazások készítésére.

A középiskolai algoritmizálás és programozás témakör az imperatív, procedurális programozást jelenti. Fókuszában az algoritmus, a programozási tételek találhatók. A blokkprogramozás és OOP ismeretekre építés mellett fontos a többi témakörben is e témát előkészíteni. Az OOP programok továbbfejlesztése helyett a gyakorlatban, alkalmazásokban található megoldások megfigyelése, „utánzása” több és célravezetőbb lehetőséget ad a megértésre, begyakorlásra. Bár a kódolás tanítása a középiskolában, jellemzően a táblázatkezelés és adatbáziskezelés ismeretek után célszerű, az algoritmusok tanítását jóval korábban el kell kezdeni, ami jellemzően egybeesik a többi témakörben tárgyalt tartalmak informatikai vonatkozásainak oktatásával. A legfontosabb alapok: csere, keresés, összegzés, szélsőérték-kiválasztás, feltételes összegzés, rendezés, szűrés (kiválogatás).

Csere

Mivel a kezünk viszonylag összetetten működő szerszám, a benne lévő két eszköz cseréjét helyben végezzük. Ezt látja a gyerek és ezt utánozza. Nagyobb tárgyak cseréje esetén külön rá kell szólni, hogy előbb az egyiket tegye le… A csere algoritmusát több, tantervben szereplő alkalmazásban taníthatjuk: felcserélve elnevezett fájlok átnevezése; rasztergrafikában képrészletek

cseréje; szövegben formátumok vagy szövegrészletek cseréje, táblázatban cellák, sorok, oszlopok felcserélése. Emellett oktatóprogramok, számítógépes és kártyajátékok is segítik az algoritmus elsajátítását. A programozást előkészítő robotika – több robot esetén – szintén alkalmas az algoritmus gyakorlására, de a vektorgrafikus képszerkesztés, az animációs programok és a blokknyelven írt játékokban nem jelent problémát, ha két objektum „egy helyen van", azaz takarja egymást. Ezeknél az algoritmus három lépésének elvárása kényszermegoldás lehet.

Keresés, eldöntés, kiválasztás

A keresés algoritmusának tanítása a legnehezebb. Az első nehézséget az jelenti, hogy a keresés többjelentésű kifejezés. Ráadásul a különböző megoldásokat tanulni, gyakorolni kell. Például: keresem a zoknim, keresem az egyenlet megoldását, keresek egy vevőt az autómra, keresek egy idézetet. A keresés algoritmusához először az szükséges, hogy iterálható legyen az a tartomány, amiben keresünk. A keresések közé tartoznak a különböző struktúrák bejárási algoritmusai, például fabejárás, gráfbejárás algoritmusai. A szeriális adatok, sorozatok esetén is négyféle keresési algoritmust alkalmazhatunk: a címszámításos (keressük az 5. elemet), a lineáris keresést, a logaritmikus keresést és – nem gépi algoritmusként – a szkennelést. Más az algoritmus, ha szélsőértéket keresünk, illetve, ha egy adott tulajdonságú elemre van szükségünk. A keresés algoritmusának tanítása során a lehetőségekhez mérten sokféle keresési fajtára kell példát mutatni, meg kell vizsgálni, hogy a rendelkezésre álló információk és a kérdés függvényében melyik keresési eljárás lehet célravezető. Például a diákoknak értenie kell, hogy mely esetekben nem megoldás a Google™ keresőjének a használata.

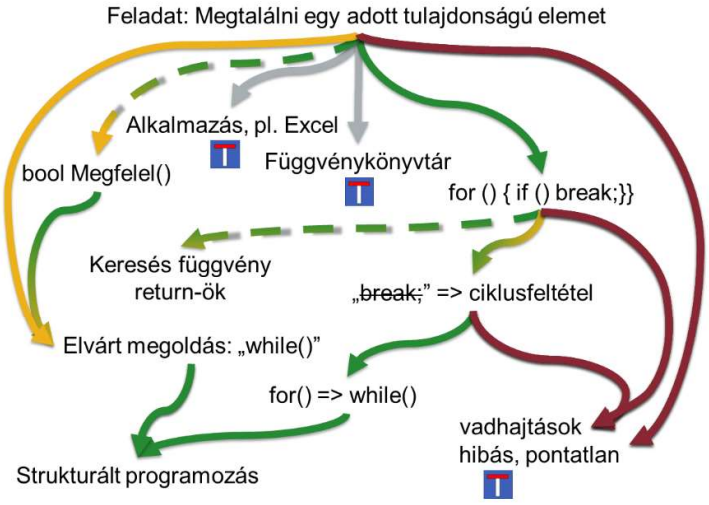
A keresési algoritmusokat érdemes különböző kereső eszközök használatának tanulmányozása során modellezni. Például modellezzük, hogy hogyan keresi meg a fájlkezelő a dolgozat*.* fájlt, hogyan találjuk ki a szükséges cipő méretét, hogyan keressük ki a névjegyek közül egy osztálytárs adatait, hogyan keressünk egy nyomtatott könyvben adatot és ugyanezt hogyan végzi egy számítógép egy elektronikus dokumentumban.

Többféle keresési mód tanulmányozása után célszerű a táblázatkezelésben használt kereső függvények működését modellezni. Ekkor kell tanítani a tartományra vonatkozó kritériumot (egy oszlop, egysor) és a két-, illetve háromféle algoritmust: a lineáris keresést és a növekvő vagy csökkenő rendezettségű adatsoron a logaritmikus keresést. Az Excelben magyarul HOL.VAN()-nak nevezett függvény működésének (különböző paraméterezések mellett a kapott eredménynek) az értelmezése kulcsfontosságú mind az informatika-, mind a programozástanulás szempontjából:

- A függvény használata a probléma deklaratív nyelvű megoldása.

- A keresési tartomány rendezettségétől függ a használható algoritmus.
- A paraméterezéstől jelentős mértékben függ az eredmény.
- A megoldás ellenőrzése, a tesztelés fontossága a tanulás során megtapasztalható. Több példa hozható a hibajelzés értelmezésére, a jónak látszó hibás megoldásra és az aktuális adatokkal jó, de elvi hibás megoldásra.
- Hatékonysági kérdések megfogalmazására ad lehetőséget. Az alapértelmezett beállítás a logaritmikus (tartományban) keresés. Miért? Miért nincs a keresés előtt beépítve a rendezés, hogy mindig logaritmikusan lehessen keresni?

A lineáris keresés algoritmusának tanítása nagymértékben függ a korábban tanult ismeretektől. A lehetséges tanulási utakról több cikkben, előadásban írtam, mivel a tanítási gyakorlatban jellemző probléma a különböző szempontok ütközése. A VI/1 mellékletben írtam arról, hogy a programozók között is tapasztalható a témával kapcsolatosan vita, ami az oktatásban komoly zavart okoz. A lehetséges megoldásokról a [57, 58] cikkekben írtam és több helyen előadtam (37. ábra).



37. ábra: A Lineáris Keresés Buktatói – diakép a megoldási utakról

Az elemzés hasznosnak bizonyult konkrét oktatási gyakorlatban is, mivel a tanulók megértették a saját megoldásuk, a társak megoldása és az elvárt megoldás közötti kapcsolatot, a megoldások mellett és ellen szóló érveket. Mivel ez az elemzés már túlmutat a közoktatás keretén, a megoldási módok közül – egyénre szabottan – a korábbi tanulmányok alapján célszerű választani: azt a megoldást tanítani, amihez a legközelebb áll, ami a tanuló számára „természetesen" jön. Azoknak, akik

1. robotikát tanultak, a végtelen ciklusból break-kel kiugrás lesz a természetes

2. blokknyelven programoztak, azok az OOP-re jellemző sok rövid függvény, eljárás íráshoz szoktak, nekik a keresés is lehet egy külön függvény, ami több helyen térhet vissza, náluk a több return adja a megoldást.
3. a deklaratív nyelvvel ismerkedtek és mögé képzelték az algoritmust, természetes lehet, hogy a keresési feltétel nem teljesülésig futó ciklust, a módszeres programozás megoldást választják.

Az 1. megoldást választók nagyobb, összetettebb feladatok esetén könnyen belezavarodnak a program strukturálásába. A 2. megoldást választók számára, kódolásra áttérve a sok kis programrészlet megtalálása, helyes összeépítése okoz nehézséget. Nekik át kell látni az osztályok rendszerét, számolniuk kell a függvények láthatóságával is. A 3. megoldásban viszont a logikában kell járatosnak lenni. Egy összetettebb keresési feltétel a helyes megoldás megalkotásának kritikus pontja.

Megjegyzés a 2. ponthoz: Ha a blokknyelv olyan, mint a Scratch, hogy a feltételes ciklusban a ciklus elején a ciklusból kilépés feltételét kell megadni, viszont a szöveg-alapú nyelven a hagyományos, ciklusban maradás feltételét váró while() vagy for() ciklus használható, akkor a keresés algoritmusának – sőt, a ciklus vezérlési struktúrának – tanulása is komoly kihívás. Már találkoztam olyan hallgatóval, aki a kilépés feltételét adta meg a for() ciklusban, mert úgy értette, hogy az kell. A Scratch és hozzá ebben hasonló blokknyelvek elterjedésével az átálláskor sokkal több figyelmet, energiát kell fordítani a de Morgan-azonosságok vizsgálatára, tudatos alkalmazására.

A közoktatásban elegendő az egyik megoldás gyakorlati ismerete, hiszen alkalmazni csak egyszerű esetekre kell tudni. Azonban a tanárnak fontos a többi megoldási módról is tájékozottnak lenni, mert ez biztosítja, hogy a többféle megoldási módot alkalmazók együtt tudjanak dolgozni, el tudják ismerni a másik megoldásban jártas társak erősségeit.

Szélsőérték-keresés, kiválasztás eredménye

A szélsőérték-keresés feladatokat táblázatkezelő alkalmazásokkal megoldva két tudásszintre bonthatjuk. Az érték keresése felsőtagozatban, az egyszerű statisztikai függvények használatával történik (MIN(), MAX()), a szélsőértékhez kapcsolódó további információk, azaz a szélsőérték helye, a szélsőértéket birtokló objektum megnevezése, más jellemzőjének meghatározása középiskolás tudás. Nem tudom, létezik-e kutatás, statisztika arról, hogy a számítógépes problémamegoldások körében melyik kérdés gyakoribb: mennyi a szélsőérték, hol van egy szélsőérték, melyek a szélsőértékhelyek, de mind a tanítási, mind feladatkészítési gyakorlatomban és

a mindennapok során is az a tapasztalatom, hogy egy szélsőértékhely meghatározása a gyakoribb, a szélsőérték értékének megadása másodlagos.

Táblázatkezelő alkalmazásban a MIN() és a MAX() függvény használatát követően gyorsan felmerül az igény a keresőfüggvények megismerésére. Például a mennyi a világcsúcs kérdés indukálja a ki a legjobb, ki tartja a világcsúcsot kérdéseket, amelyre a tanterv alapján csak évekkel később tud válaszolni a diák. Ekkor a keresőfüggvények használatával kapunk megoldást, amit sokkal nehezebb megérteni és helyesen használni. Lényegében az INDEX(HOL.VAN(MAX())) függvénykombinációt kell megérteni és alkalmazni tudni. A szélsőértékek mindegyikének megadásához még speciálisabb tudás szükséges: a helyes megoldáshoz speciális szűrőt kell alkalmazni, amelyben a szűrési feltételben függvényérték szerepel, a megoldás azonban eljárás (kigyűjtés) eredménye.

Adatbáziskezelő alkalmazásokban a mezőre számított MAX() és MIN() a táblázatkezelőkben megismert logikával számítható, de az értéket felvevő rekord többi adata kétféle úton adható meg, amely megoldásokat eltérő algoritmusokkal modellezhetünk. A táblázatkezelésben megismert algoritmust követve, egy segéd- vagy allekérdezésben meghatározzuk a szélsőértéket, majd ezt felhasználva megadjuk azokat a rekordokat, amelyeknek az adott mezője ezzel egyenlő. Másik gondolatmenet – és egyúttal másik algoritmus – szerint, az adatok rendezésével és első elem kiválasztásával, egy lekérdezésben megadhatjuk az eredményt. Bármelyik esetet választva, az SQL nyelv tulajdonságaiból következően az eredmény egy lista lesz, ami egyetlen előfordulás esetén egy elemet fog tartalmazni.

A deklaratív programozási nyelvek hátterében mindig egy imperatív végrehajtó motor működik. Vizsgálandó, hogy a „TOP” illetve a „LIMIT” megadása hogyan befolyásolja az eredmény megadását. Érdemes itt elgondolkodni azon, hogy szükséges-e az adathalmaz teljes rendezése, illetve – a rendezés algoritmusának tanításakor – érdemes arra kitérni, hogy az egyes rendezések közül melyek azok, amelyek az első néhány adatra is gyorsan elvégezhetők.

Az algoritmusok tanítása során a táblázatkezelésben használt megoldás algoritmusát vizsgálva látni kell, hogy legjobb esetben is a megoldásban két ciklus szükséges, egyszer végig kell nézni az adatokat a legnagyobb érték meghatározásához, egyszer pedig meg kell keresni az eredmény helyét. Ezután a válasz – az INDEX() függvény algoritmusa – már könnyen megadható tömbös tárolás esetén címszámítással, azonban dinamikusan csatolt sor esetén egy harmadik ciklus is szükséges lehet. Érdemes megvizsgálni a feladatok jellemzőit. A szélsőérték kiválasztása egyszerű adatsor esetén gyakran matematikafeladatot jelent, függvényvizsgálathoz kapcsolódóan szükséges. Itt a szélsőértékhely jellemzi a függvényt. Összetett adat, objektum

esetén a szélsőérték meghatározásához meg kell adni a szempontot, a tulajdonságot, azaz az adatra, az objektumra meg kell határozni a relációt. (Elméleti alapokon ez nem középiskolás tananyag, de a vizsgálat, a feladat elemzése igen.) Ezután a szélsőérték meghatározásában sokkal hatékonyabb az adat indexének vagy az objektum címének (referenciájának) megadása. Ebből a szempontból a szélsőérték keresése sokkal inkább keresés, mint összegzés.

Látható, hogy a szélsőérték-keresés az egyes problémamegoldásra használt környezetekben eltérő hatékonyságú algoritmusokkal történik. Az eredménycélként megadott programozási ismeret – az algoritmus megvalósítása formális programozási nyelven – azt jelenti, hogy az imperatív, procedurális programozás eszközeivel kell megvalósítani az algoritmusokat, ezért olyan programozási (fejlesztő) környezet kell, amely ezt támogatja. A konkrét eredménycél a szélsőérték helyének procedurális, függvénykompozíció nélküli meghatározása. A formális programozási nyelv tanítása nem (csak) a kód írásának elvárása, hanem a vezérlési struktúrák memória és futásiidő szempontjából hatékony alkalmazásának megtanulását célozza meg.

Feltételes összegzés, szűrés

Felső tagozatban elvárt az egyszerű statisztikai függvények használata: a megszámlálás, az összegzés, az átlag- és a szélsőértékfüggvényeket jelenti. Már ekkor, a felsőtagozatban tisztázni kell az összegzés és összeadás közötti különbséget. Mikor jó megoldás néhány cella összegét megadni, melyek azok az esetek, amikor a tartomány celláira (még, ha csak 1 vagy 2 cellára akkor is) végzett összegzés a helyes megoldás.

Informatikai tudást ad, későbbi problémák megoldásának a potenciálja, ha már a legelső gyakorlatok között, a függvények használatával való ismerkedés során tisztázzuk, hogy a függvények milyen adattípussal hogyan számolnak. A statisztikai függvények jellemzően számtípusokkal (ha dátumformátumú akkor is) számolnak, de létezik olyan változatuk is, amelyben a szöveget – 0 értékkel – illetve a logikai értékeket is figyelembe veszi. (A jelenleg használatos Excel verziók kapcsán a szoftverlokalizációs problémákra is érdemes kitérni: miért DARAB2 és ÁTLAGA; mikor MAX2 és mikor MAXA; miért nem tudják egységessé tenni.)

Az eredménycélokban szintén nem szerepel, de informatikai szempontból elvárható a hibakezelés ismerete. Például, a jegyek tantárgyi átlagánál az értékelés nélküli tantárgyakra kapott #ZÉRÓOSZTÓ hibajelzés helyett egy üres szöveg lehetővé teszi a további vizsgálatokat (a példánál maradva: azt, hogy a létező eredmények közül melyik a legjobb). Nem hibajelzés, de érvénytelen lehet az eredmény, ha olyan tömbön keresünk maximumot, amelyben nincs adat. Az ilyen és hasonló problémák kezeléséhez már felső tagozatban is elvárható a HA() függvény, esetleg célzott hibakezelő függvények ismerete.

A hibakezelés az, ami természetessé teszi, hogy egy cellában alternatívan két értéket jelenítsünk meg. Erre lehet építeni a feltételtől függő értékmegjelenítéssel és formázással kapcsolatos problématípusokat. Kezdetben a leggyakoribb alkalmazás a nem megfelelő adat esetén üres szöveg megjelenítése, ezt követheti a megfelelő cellák kiemelése formázással, a kiemelés kiterjesztése sorokra. A hibakezelés gondolatából következik, hogy adatsorokból a feltételnek megfelelő értékek külön oszlopban jelenjenek meg, míg a nem megfelelők sorában üres szöveg legyen – rejtetten ez a kigyűjtés algoritmusa –, így a feltételnek megfelelőkre lehet végezni összegzést. Ezzel a feltételes összegzésre lépésenkénti megoldást tudunk adni. A feladatmegoldásnak ez a formája a feladat szövegének elemzése, a probléma részekre bontása informatikai gondolkodás szempontjából fontos. Az ilyen típusú feladatok a feltételek és az utána alkalmazandó összegzések többféle lehetősége miatt egyértelműen kreativitást is igényelnek. (Ellenpéldaként, a ki_a_leg típusú feladat megoldását sokkal inkább be lehet magolni és utána el is felejtődik.) A feltételes összegzések elemi megoldásában az, hogy először kigyűjtöm a megfelelőket, nagyon helyigényes ráadásul egy idő után unalmas, ami felveti az igényt arra, hogy egy utasítással vagy függvénnyel lehessen megoldani ezeket a feladatokat. Az egy feltételt tartalmazó feltételes összegzésre táblázatkezelésben ötféle megoldás létezik (a magyar Excel elnevezéseket használva: a már bemutatott HA() függvénnyel „szűrés” majd összegzés, SZUMHA(), SZUMHATÖBB(), AB.SZUM(), {SZUM(HA())}). A feladatot adatbáziskezelőben megoldva a lekérdező rács és SQL nyelvű megoldás további két megoldási módot jelent.

A hétféle deklaratív megoldás csak a legegyszerűbb feladatoknál ekvivalens. Az adatmennyiség, a kérdések száma és a feltételek összetettsége alapján egyes megoldási lehetőségek hatékonyak, mások a megkötések miatt használhatatlanok. A megoldási módok taníthatóságáról, a tanítás sorrendjéről és feladattípusok szerinti hatékonyságáról a Guess the code of conditional summation [137] cikkben írtam. A gyakorlatban tapasztalt, különböző sorrendekben felépített tananyag helyett, érdemes szinte egyszerre tanítani mindegyik megoldási módot, mert a diákok egyéni ízlésétől, korábbi tapasztalatától függ, hogy melyik megoldási mód lesz számára a legjobb. Későbbiekben a feladatok nehezítésével, nagy mennyiségével a megoldási módok közül többet is kipróbálnak a diákok, de egyéni, hogy melyik megoldásokat preferálják. Az algoritmikus gondolkodást leginkább a tömbképlettel történő megoldás fejleszti [138], a többi módszerhez a tanítás során kiegészítésként célszerű kapcsolni a „géptől elvárt” algoritmust. Közoktatásban az elemi (egy feltételű) feladatok megoldása elégséges, a több – szűkítő, konjunktív – feltételű feladatok megoldása elvárható. A megoldási módoknak csak egy része alkalmas diszjunktív – „ilyen vagy olyan” jellegű – feltételek megadására, ehhez a megoldási módszerek szinte mindegyikében jártasság kell.

Az informatikaoktatás keretében, a függvények használatának gyakorlásával párhuzamosan kell a műveletvégzés algoritmusát is tárgyalni. Fontos ismeret, hogy az összegzés nem helyiértékenként történik, hanem kumulációval. Az összegzés algoritmusának hasonlósága a szélsőérték-keresés algoritmusával – eredmény kezdőértéke; minden adattal csinálok valamit, ami végeredményhez közelebb visz – később nagy segítség lesz a programozás tanulásánál. Egy-egy algoritmus blokknyelven meg is valósítható, például egy játék eredményeinek összegzésnél. A feltételes összegzések egyetlen függvénnyel történő megoldásában az előzetes kigyűjtés a kumulációval összeépíthető, azaz nem szükséges a kigyűjtött adatok tárolása. A feltételes összegzés algoritmusa hasonlít a szélsőérték-kiválasztás algoritmusához. Megfigyelhető, hogy a tömbfüggvénnyel történő megoldásban az összeépítés nem történik meg. Pont azért tömbfüggvény típusú a megoldás, mert a kigyűjtést külön elvégzi a memóriában létrehozott tömbbe. A beépített függvényeknél memóriaoptimalizálásra ad lehetőséget a ciklusok egyesítése. A megoldási módok elemzése hatékonysági szempontok felvetését teszi lehetővé, ami indokolja a megoldás formális nyelven történő – vezérlési struktúrákból építkező – megoldását.

A feltételes összegzésekkel párhuzamosan lehet tanítani a táblázatkezelésben a szűrést, valamint másolással a kigyűjtést. A SZUMHA() és SZUMHATÖBB() függvények összegzésmentes megfelelője helyben szűréssel megoldható. Az AB.SZUM() feladatok megfelelője az irányított szűrés. Mivel a táblázatkezelésben elsősorban függvényekkel dolgozunk (mert ezek automatikusan újra számítódnak), a szűrést a feltételes összegzés után érdemes tanítani, kiemelve azt, hogy a kigyűjtés függvénnyel mindig sok „nem látható” eredményt ad, ami memóriaigényes, míg a sorok kitakarása és másolás csak a megfelelő adatokat mutatja, kevesebb memóriát igényel, de nem frissül[60]. Adatbáziskezelésben a szűrés, a megfelelő rekordok kiválogatása az elsődleges, ezt „fejeli meg” az összegzés. Itt minden esetben eljáráshívásról van szó. A feltételes összegzés elvégzése adott mezők minden lehetséges értékére, a részösszegek számítása. A részösszegek számítására táblázatkezelőkben is van beépített eljárás, de sokszor függvények sorozatával oldjuk meg, ami sok adat és sok függvény esetén láthatóan memória- és időigényes újraszámítást eredményez.

Problémamegoldás informatikai eszközökkel témakörön belül, a táblázatkezelés és adatbázis-kezelés tárgyalásában be kell mutatni, hogy egy probléma többféle eszközzel is megoldható, de ennél is fontosabb annak sokféle szemléltetése, hogy a megfelelő eszköz kiválasztása nem

[60] Az MS365 Nagyvállalati Excelben létezik SZŰRŐ() és RENDEZ.ALAP.SZERINT() függvény. Emellett az alapértelmezett eredmény nem 1 adat, hanem egy tömb. Ezzel ez a táblázatkezelő még inkább tekinthető egy funkcionális programozási nyelvű fejlesztő környezetnek.

csak a probléma típusától, hanem annak méretétől – adatmennyiségétől, a megoldási mód számításigényétől –, az adathoz és a műveletvégzéshez való hozzáférés módjától, az eredmény aktualitásának fontosságától is függ. Az informatikai tudás része a hatékonyan használható eszköz kiválasztása, az eszköz hatékony (gyors, pontos) alkalmazása.

Példák feltételes összegzéshez eszközválasztásra

- Az adatok változását azonnal követő eredményekre van szükség, például költségtervezés – táblázatkezelő
  - SZUMHA() a legoptimálisabb akkor, ha egyetlen feltétel van és az az összegzendő adatokra vonatkozik. Pl. 100-nál kisebb értékek összege; töredékszavazatok összege. Hasonlóan ehhez, a többi statisztikai függvényre, pl. negatív értékek átlaga.
  - SZUMHATÖBB() a legoptimálisabb akkor, ha egy vagy több, de szűkítő (ÉS) feltétel mellett szükséges az összegzés és ezek a feltételek az egyes adatok értékére vonatkoznak.
  - AB.SZUM() a legoptimálisabb akkor, ha a feltételek egy része alternatív (VAGY) és a számítás egyedi, a feltételtábla elhelyezése nem okoz problémát, ha kell, a másolása könnyen megoldható. Pl. bukott vagy 3,5-nél alacsonyabb tanulmányi átlagot elérő diákok hiányzásának összege/átlaga.
  - {SZUM(HA())} tömbfüggvényes megoldás használata a legoptimálisabb akkor, ha a feltételekben az adatokra vonatkozóan számítás is szerepel. Pl. a hárommal osztható számok száma, összege, átlaga, maximuma.
  - Feltétel külön oszlopban kiszámítása, ebből egyszerű összegzés a legoptimálisabb, ha a feltétel összetett, más adatsoroktól is függ vagy az összegzés több típusa szükséges. Pl. előző naphoz képest melegebb napok száma, a melegedés átlaga, minimuma, maximuma.
- Nagy mennyiségű, esetleg több felhasználó által bővített, módosított adatok kezelése – adatbáziskezelő.
- Több táblázatban szereplő kapcsolódó adatokból statisztika készítése – adatbáziskezelő. Pl. a felvételin egy adott ponthatárt elért lányok első félévi tanulmányi átlaga.

A feltételes összegzés szövegalapú programozási nyelven történő megvalósítása a deklaratív megoldások mögé képzelt algoritmusok imperatív nyelven történő kipróbálását jelenti. Emiatt olyan nyelvet vagy a programozási nyelvnek olyan alrendszerét kell használni, amelyben a vezérlési struktúrákból építkezve, a megfogalmazott algoritmusnak megfelelően lehet megírni a

tanult függvények megfelelőit. Ahhoz, hogy a feladat érdekes legyen, a feltétel valamilyen gyakorlati problémával kapcsolatos legyen, a feladatokhoz tartozó adathalmaz összetett kell, hogy legyen. Táblázatkezelőben sorokba rendezett tulajdonságok táblázatával érdemes foglalkozni, de több esetben az oszlopok és sorok egyenrangúak. Adatbáziskezelőben a rekordok táblákban találhatók. Ennek megfelelően, a programozással megoldott feladatok adatstruktúrája egy- vagy kétdimenziós tömb vagy egydimenziós dinamikusan változó méretű tömb, amelyeknek az elemei adatstruktúrák vagy egyszerű adattagokkal bíró objektumok.

Amennyiben a táblázatkezelést – és ebben a függvények algoritmusainak elemzését – követi a programozás tanulása, akkor az egyes feladatok függvényként megoldása reprodukciós feladat lehet: „írjuk meg az adott feladathoz a SZUMHA() függvényt". A megoldási lehetőségek párhuzamos oktatása és a megoldási lehetőségek algoritmussal majd programmal való modellezése nagyban segíti az informatika oktatási céljának megértését: a számunkra aktuálisan legmegfelelőbb eszköz kiválasztásához ismerni kell az eszköz működésének belső logikáját; az alternatív megoldások a konkrét feladat alapján rangsorolhatók, de nincs sem abszolút sorrend, sem univerzális módszer. Az eszköz kiválasztásának módja, az alternatív megoldások mérlegelése minta az alternatív alkalmazások vagy operációsrendszerek közötti választásra is. Az informatika humán területre való átvetítésével: társakkal való hatékony együttműködéshez nem elég a társak képességeit ismerni, meg kell érteni a belső motivációkat is.

Rendezés

Az adatok rendezése számos alkalmazás beépített eszköze. Fájlkezelőkben a fájlok és mappák tulajdonságai alapján, szövegszerkesztőben bekezdések szövege alapján lehet rendezni. Komolyabban táblázatkezelésben és adatbáziskezelésben tanítjuk, ahol a többkulcsos rendezés, illetve speciális rendezési beállítások miatt kap nagyobb hangsúlyt. Az algoritmizálás és programozás részeként nem jelenik meg tananyagként, azonban több szempont miatt nem szabad kihagyni.

- A kigyűjtött adatokat rendszerint valamilyen rendezett formában szeretnénk megkapni, ezért egy rendkívül gyakori feladattípus.
- A keresés módját jelentősen befolyásolja, hogy rendezett adathalmazban sokkal gyorsabban lehet keresni, kérdés, hogy nem rendezett adathalmaz esetén érdemes-e rendezéssel kezdeni a megoldást.
- Rendezési algoritmusból nagyon sokféle van, ezért akár a csoport minden tagjának önálló gondolata lehet. Az algoritmikus gondolkodást nagyon jól fejleszti a saját rendezési algoritmus kitalálása, elmagyarázása, illetve a társak magyarázatának megértése. (Esetleg: megoldások összehasonlítása, hol térnek el az egyes lépésekben)

A „saját” rendezési algoritmus nagy valószínűséggel egy már ismert algoritmus önálló megalkotása, amelynek helyességét konkrét programban célszerű bemutatni. A sikeres megvalósítás a programozástudás egyik mérföldköve. A „saját” algoritmus és a társak által készített vagy az ismert algoritmusokkal való összehasonlítása tudatosabbá teszi a „saját” algoritmus lépéseit. A saját gondolat összevetése mások gondolatával, az azonosságok és az eltérések elemzése nagyon fontos informatikai készség.

A rendezési algoritmusokhoz hasonló fejlesztő hatása lehet számos rendezéshez hasonló vagy azt használó feladatnak, mint például a hanoi-tornyai, kirakós kártyajátékok, tili-toli.

Logika

A vezérlési utasítások végrehajtása logikai kifejezések kiértékelésének függvényében történik. Logikát (jó esetben) már a bölcsődés korban tanulunk főleg implikációk formájában. (Ez felveti azt a kérdést, hogy a szülői következetesség hogyan hat ki a logikus gondolkodási készség fejlődésére, de ennek részleteivel nem foglalkoztam.)

1987-ben írt első szakdolgozatom [139] – az indirekt bizonyítás oktatásával kapcsolatban – jelentős részben szól a logikai készségek fejlesztéséről.

> *„Az elsőosztályos gyerekek ... tételek kimondása és bizonyítása helyett szabályjátékokat játszanak. ... A középiskola első osztályában találkoznak a tanulók először a logikával, itt tanulják meg, mit nevezünk logikai állításnak, mi az ítélet, a logikai függvény, megtanulják a konjunkció, a diszjunkció és az ekvivalencia fogalmát. ... A középiskolás tananyagból azonban teljesen kimaradt az implikáció, azaz a következtetés mint logikai művelet, annak ellenére, hogy a gyakorlatban – feladatmegoldás, bizonyítás során – sokszor használják.”*

Nagyon régen kutattam ezt a témát, azonban a problémákkal napjainkban is találkozunk, az informatika térhódítása a témát hangsúlyosabbá teszi, oktatási vonatkozásainak alaposabb megismerését teszi szükségessé [140].

1. A köznyelvben használt logikai állítások nem felelnek meg a matematika logika kifejezési módjának. Például: „a belépés gyerek és nyugdíjas számára ingyenes”, „nincs semmi baj”.
2. A matematikában írt egyenletek jellemzően logikai kifejezések, a problémamegoldás, a bizonyítás során a matematikai logika használata jellemző. Informatikában meg kell különböztetni az értékadást (legyen egyenlő) a relációtól (egyenlő-e?); a végrehajtást az elemzésétől.

3. A ciklusfeltétel jellemzően a ciklusmagba belépés feltétele, miközben természetes gondolkodásunkban sokszor a kilépés feltételére gondolunk. Főleg a hátultesztelős ciklusnál feltűnő ez a gondolkodásmódbeli kétféleség.
4. Összetett logikai állítások tagadása szóban (fejben elvégezve) nagyon nehéz. Az előző két pontban szereplő esetekkel is összefüggésben, amikor már a kifejezendő állításban is pontatlanságok vannak, ennek a tagadása anyanyelvi eszközökkel még nehezebb. Például: kinek kell fizetni a belépésért, ha gyerek és nyugdíjas számára, valamint minden hónap első hétvégéjén ingyenes.
5. A logika formalizálásával a kifejezés pontosabbá válik, de könnyen elveszíti köznyelvi értelmét, ezért nehéz az eredmény összevetése a szándékkal. Míg a számítások eredményét nagyságrendileg lehet ellenőrizni, a logikai kifejezések eredményének nincs nagyságrendje, egy bonyolult kifejezés átalakításának eredménye teszteléssel lehetséges.
6. A Ha…, akkor… mondatszerkezet kétértelmű. Az egyik értelmezésben, az implikációban a hárompont helyén premissza és konklúzió szerepel, a másik értelmezésben, a vezérlési szerkezet feltétele (condition) és utasítása (statement) szerepel. Az informatika imperatív gondolkodásmódja a logika segítségével szabályoz, a három vezérlési struktúrából kettőben, az alternációban és a ciklusban a logikai kifejezés „vezérel”, ami más, mint a „megállapító” konklúzió. A [53] cikkben leírtak alapján e két fogalom zavarja egymás értelmezését. Emiatt, ahogy azt [57, 58] cikkekben feltártuk, a lineáris keresés strukturált algoritmusának valódi megértése nehéz.

Példa: Implikáció és vezérlés a lineáris keresésben

A: az i-edik elem létezik, azaz i< N; B: a sorozat i-edik tagja rendelkezik a keresett tulajdonsággal
ciklus amíg $A \wedge \neg B$

kilépett, mert $\neg A \vee B$

- $\neg A \vee B$ egyszerűbben egy implikáció: $A \Rightarrow B$. Jelentése: ha a sorozat létező elemét kaptuk akkor az rendelkezik a keresett tulajdonsággal.
- Mint vezérlésátadás, az „A”-t ellenőrzöm, mert az $\wedge$ rövidzár tulajdonságát így tudom érvényesíteni. Ha(A) akkor talált, különben nem talált.

A lineáris keresés deklaratív megfogalmazása az indirekt bizonyítás gondolatmenetét adja:

- Állítás: $\exists A \Rightarrow B$ (Van olyan elem, amelyik rendelkezik a tulajdonsággal).
- Bizonyítás: Tegyük fel, hogy $\forall A$ esetén $\neg B$ … (Jelentése: azt látjuk, hogy soha nem teljesül az adott tulajdonság.) A konklúzió: $\neg \exists A$ Ellentmondásra jutottunk. (Jelentése: a sorozatban nincs elem.)

7. A matematikai logikát sokféleképpen használja az informatika, de jellemzően módosított értelmezéssel. Például, az alternatíva a „vagy” művelet, ami az elágazás vezérlési struktúra jelzője lesz. A diszjunkció a „kizáró vagy” művelet, de a logikai kifejezések felírásában a konjunktív – „és” műveletekből építkező, szűkítő, metszetképző – forma ellenpárjaként adott diszjunktív forma a „vagy” műveletekből építkező, megoldási uniót képző kifejezési mód, ahol szó sincs az egyetlenséget jelző „kizáró” tulajdonságról. Nem hagyható ki a logikai műveletek értelmezése és használata során a rövidzár elve, amely a legalapvetőbb matematikai tulajdonságokat – kommutativitás, asszociativitás – írja felül és alapvető szerepe van a programok helyes működésében.

Az első két pontban megfogalmazott ismeret oktatása – napi gyakorlatként – a szövegértés és algoritmizálás gyakorlása keretében történik. A harmadik pont szükségessé teszi a logikai kifejezések formális felírásának tanítását a korábbi gyakorlathoz képest sokkal fiatalabb diákoknak. A negyedik és ötödik pontban már a formálisan megfogalmazott kifejezések használatában való megfelelő jártasság jelenti a probléma megoldását. A hatodik és hetedik pontban leírt problémákra a megoldás a közoktatásban a gondolkodási modellek szétválasztása. A bizonyítás, azaz a „belátás” nem vezérlés. A matematika és az informatika különböző modelleket használ problémamegoldásra, a definíciók a modellhez kapcsolódnak, a modellen belül érvényesek. Az informatika tudomány magasszintű művelésének része lehet ezen modellek összehasonlítása. A közoktatásban az azonosnak látszó fogalmak különbözőségére kell helyezni a hangsúlyt: programozásban az „és” nem kommutatív, a Ha…, akkor… vezérlési struktúra, amibe feltételt és utasítást adunk meg.

Programkészítés

A programozástudás egyik lehetséges értelmezése a programkészítés képességének megléte: akkor tudok programozni, ha képes vagyok programot készíteni. Az új tantervek, a fejlesztési tervek és a világtrend alapján a programozást egészen kis korban el lehet kezdeni tanulni, lényegében már az óvodában. Természetesen a korosztálynak megfelelő eszközökkel, ezért szerepel a tantervben a padlórobotok programozása. Ebben az értelemben a programozáshoz az kell, hogy

1. tudjunk utasításokat kiadni, amit a gép tárol, értelmez – programozásinyelv-ismeret;
2. az utasításokat át tudjuk adni a gépnek – fordítás (interpreter, compiler);
3. el tudjuk indítani a végrehajtást – futtatás;
4. ellenőrizzük, hogy a végrehajtás a szándékunknak megfelel-e, tudjuk módosítani az utasításokat – tesztelés, szemantikus ellenőrzés

5. hiba esetén értsük a hibajelzéseket (legalább a legalapvetőbbeket) – fejlesztőkörnyezet ismerete, szintaktikai hiba és eszközhiba felismerése;
6. ismerjük a hiba elhárításának módját – eszközismeret, eszközkezelés.

Padlórobotoktól a programozható építőjátékokon át a robotikáig

A legegyszerűbb programozható (oktató) eszközök a padlórobotok. Jellemző, hogy nincs külön fejlesztőfelület, hanem a roboton levő gombok egymásutáni megnyomása vagy egyéb fizikai eszközök egymásután helyezése jelenti a kódolást. Az első algoritmus, amit meg kell tanulni: bekapcsolás, szerkesztőmódra váltás. kódolás, kód lezárása, futtatás, szerkesztő mód… kikapcsolás. Ennek ismeretében lehet „alkotni" egyéb programokat, megfogalmazni algoritmikusan egy útvonal bejárását. A leggyakoribb hiba az elem/akkumulátor lemerülése (felkészült óvodás tisztában van azzal, hogy különböző játékaiba mennyi, milyen típusú elem kell, hogyan lehet tölteni az akkumulátort). A programozást nehezíti, hogy a „kódolást" nem szabad elrontani. Óvodásoknál még a tervezés is fejben történik. Egy átlagos program beírása gombok lenyomásával sokkal nehezebb, mint az iskolaérettséget vizsgáló sorminta rajzolása, mert nem ciklikus és ráadásul nem is látható, hogy egy adott állapot előtt mi lett kódolva.

Az „irányító panel" – a kódoláshoz használható kártyák, kockák és egyéb eszközök – használatával fizikailag kettéválik a fejlesztő és a futtató eszköz. Általános iskola alsó tagozaton már lehet tömöríteni a kódon, az ismétlődést számmal megadni. Ekkor már igényként jelentkezhet, hogy a programot – például azt, hogy 6 előre 1 jobbra – ne kártyákkal, hanem számítógépes alkalmazással írjuk és a megfelelő számú elem kirakása helyett elegendő legyen egy szám megadása.

Ezen a ponton a programkészítésbe algoritmikusan megjelenik a ciklus, de jelentősebb a fenti hat pontból a 2., az 5. és a 6. programozási részterületen a változás. Már nemcsak a robot merülhet le, hanem a mobil/tablet is. És a memóriája is megtelhet, lefagyhat, felbukkanhat egy nemvárt programablak… A fejlesztőeszközt is alaposan meg kell ismerni. A program áttöltése is okozhat problémákat: az adó adásra, a vevő vételre legyen állítva és stabil legyen a kapcsolat… A programkészítés folyamata összetettebb lett, a lehetséges hibák száma nőtt.

Az útvonalak programozása egy idő után nagyon unalmas, nem szólva az alakállandó robotokról. Az elemekből összeépíthető robotok programkészítés szempontjából nyelvi (1. programozási részterületen) bővítést jelentenek, a programban megjelennek az elemeket kezelő objektumok: motorok, szenzorok; finomabban paraméterezhetők az utasítások: a motor működésére vonatkozó egység, hossz és fok mértékek; a szenzorok adatai feltételek megadását teszik

lehetővé. És tovább bővül a hibalehetőségek listája: elemek kapcsolódásának hibái, sérült eszközök; figyelembe kell venni a mintavételezési és kvantálási tulajdonságokat.

A legismertebb programozható építőjáték a LEGO® Mindstorm, ami ütésálló, „kakaóbiztos", illesztési tulajdonságai nagyon jók, mérési és vezérlési pontossága tűrhető. Legnagyobb előnye azonban nem programozásban, hanem a passzív elemek sokféleségében rejlik. Legnagyobb hátránya pedig az ára. Robotok programozása szempontjából fontos, hogy az építőjátékokban az aktív (programozható, érzékelő vagy vezérelhető) eszközök jól védetten, burokban, modellelemben találhatók, valójában játékok. Az „igazi" aktív eszközök (a különböző mikrokontrollerek, motorok, LED-ek) a használati eszközökben is megjelenő szenzorok, kapcsolók. Ezekből robotok, vagy inkább eszközök készítése típustól függően, egyre komolyabb előismereteket igényel a matematika, a természettudomány, a műszaki és informatikai ismeretek (MTMI/STEM) minden részéből. Az errefelé történő továbblépés a micro:bit eszközkészlet használata, amelyben az aktív elemek modulárisak, könnyű az összekapcsolásuk, átépítésük, de kevésbé védettek és a passzív elemeket egyénileg kell előállítani.

Fejlesztőkörnyezet és nyelv

Amikor a padlórobotok irányítópultja helyett megjelenik a számítógépen futtatható fejlesztőkörnyezet, a hardveres problémák megsokszorozódnak. Informatikatanítás szempontjából ez elég nagy időveszteséget jelent, ezért érdemes a robotok, külső eszközök vezérlése helyett szoftveres problémákra áttérni. A felsőtagozaton megjelenő blokknyelvű programozás alkalmas robotok programozására, de épp ennyire megfelel a számítógép beépített vagy alapértelmezett inputjain érkező jelek vételére, az outputjain keresztül vezérlésére is. A fejlesztő és futtató környezet ugyanazon az eszközön van, így a fordítás (2. programozási részterület) kevesebb hibalehetőséget rejt, az 5. és 6. részterület a használt számítógép és használt fejlesztőkörnyezet hibajelzéseinek ismeretéről szól.

A blokknyelvek használatának divattá válásával az adott korosztályra tervezett és éveken át tanított Logo a hazai oktatásban háttérbe szorul. Míg Logoban a kódolás szöveges, a blokknyelveken a blokkok előregyártott vizuális kódelemek, amelyek kapcsolódása vizuálisan ellenőrizhető illeszkedéssel történik. Ezért a szintaktikai hibák valószínűsége jelentősen kisebb, csak a paraméterek típusára vonatkoznak, ami megkönnyíti a program készítését, ugyanakkor kevesebb jártasságot ad programkészítés problémáinak megoldásában. A Logo kiváltásával együtt járt, hogy algoritmusok tekintetében a rekurzióra (bár lehet, de) nem kell kitérni, cserébe a strukturált programozásra jobban felkészít a blokknyelv.

A középiskolában tanított szövegalapú programozás minden tekintetben nagy váltást jelent a blokknyelvű programozáshoz képest. A blokkokkal programozás és a szöveges programozás között majdnem akkora a különbség, mint a cukrászdában vásárolt tortalap és a lisztből, tojásból, egyéb hozzávalókból sajátkezűleg készített torta között. Mutató eszköz használata helyett gépelni kell. Egy elem beillesztése helyett minden karaktert be kell írni, jó sorrendben, az is számíthat, hogy kicsi vagy nagybetűvel írjuk. A blokknyelven jól programozó diákok számára az áttéréskor zavaró, hogy a program begépelése lassabban megy. A gépeléssel a hibák száma is jelentősen megnő. A szemantikai hibák kiszűrésében segít a fejlesztőkörnyezet, de meg kell tanulni „kommunikálni vele". Az átállást jelentősen kellemesebbé (könnyebbé) teszi, ha a fejlesztőkörnyezet hatékonyan támogatja a kódkiegészítést, snippetekkel segíti a gyors kódolást.

A szöveges programozási nyelveknek sokkal több eleme, alapkifejezése, beépített eljárása van, jelentősen kevesebb az alapértelmezett érték, művelet. Ezért több kódrészletet kell megtanulni, amelyek egy része kezdetben érthetetlen. A mintakódot sokkal könnyebb másolni, ami átsegíthet a kötelező kódsorok gyors reprodukálásán, de károsan hat a tanulásra. A kész kódrészleteket is célszerű gyakran végigírni, mert másképp a részletek megtanulása elmarad.

Példa másolható, de begépelendő kódokra:

- C nyelvben stdio.h, mert nem stbio.h és nem is studio.h; math.h, ami nem mat.h; scanf(″%d″, &a)
- C# nyelven namespace{class Program{}}; static void Main() – más nyelveken másképp

A kód olvasása – megírt kódok értelmezése – általában nehezebb, mert a színezés kevésbé segít: a blokkban lényegében a megjelenített struktúra háttérmintázatának színezése jelenti a segítséget, a szöveges nyelvben viszont a kulcsszavakat, típusokat karakterszínezéssel emelik ki.

A futtatás többféle módját érdemes megismerni:

- fordítással együtt debug módban,
- töréspontoknál megfigyelve a változók állapotát,
- release módban a fejlesztő környezetből, valamint
- parancssoros fordítással és futtatással.

A programkészítés igazi élménye azonban a kész program futtatása. Pontosabban az, amikor a kész programot mások futtatják.

A szövegesen kódolt programkészítést nem csak a blokknyelvű programozással lehet előkészíteni. Informatikai gondolkodás szempontjából is érdemes elemezni, hogy miben különböznek vagy egyeznek meg a blokknyelvek, a fejlesztőkörnyezetnek, a programozási nyelvek, a játékkészítő, illetve a dokumentumkészítő alkalmazások.

A Klik & Play (1994-ben kiadott) játékkészítő alkalmazás „programozásmentes", személyközpontú OOP alkalmazás, amiben azért nem kell programozni, mert az eljárásokat is készen kapjuk, csak ki kell választani. A Game Makerben a GML script nyelven lehet az objektumok eseményeihez kódot írni. A Game Maker játékfejlesztő szoftver, benne a GML egy nagyon kis programozási nyelv. A Unity (valós idejű 3D fejlesztőkörnyezet) ehhez hasonló, de C# nyelven lehet programozni. A MakeCode egy fejlesztőkörnyezet, amiben blokknyelven és JavaScriptben lehet programot írni mikrokontrollerekre, MineCrafthoz. A Scratch a leírások alapján egy programozási nyelv. Valójában fejlesztő környezet, ami a blokkokból JSON projektet készít, amit tömörített kódfájlba ment. A Visual Studio, az Eclipse, a Code::Blocks… fejlesztő környezetek, amelyekben egy vagy (inkább) több nyelven lehet programozni. Ezek a nyelvek jellemzően szövegalapú nyelvek, de létezik grafikus moduljuk is, amit egyes fejlesztőkörnyezetekhez integráltak. Például az állapotgépek működését leíró YAKINDU Statechart Tools az Eclipsbe integrálható, a Nassi-Shneiderman diagram a Code::Blocksban található. Ez utóbbi továbbfejleszthető lehetne blokknyelvre. Mindkét példára jellemző, hogy a grafikusan készített programot szövegalapú nyelvre át lehet alakítani, visszafelé viszont nem alakíthatók. Erről az oldalról nézve, az újabb Unity egy, a Visual Studioba integrált modul.

Visszatérve Klik & Play-re, ez a játékkészítő alkalmazás a bemutatókészítő alkalmazásoktól annyiban különbözik, hogy az objektumok egymásra is hatnak, például ütköznek. A különböző események állapot- vagy mozgásváltozást eredményeznek a PowerPointban is. A bemutató pptx fájlja tömörített formában, xml kódban tárolja a tartalmat, a formát, az animációt. Ebben az értelmezésben a PowerPoint egy fejlesztőeszköz és interpreter, a pptx állomány pedig a kód. Nem blokknyelven, de grafikus környezetben, WYSiWYG nézetben kódoljuk a bemutatónkat. Az animációtól eltekintve hasonlót mondhatunk a többi dokumentumkészítő alkalmazásról is, csak a nyelv más: nem programozási nyelv, hanem dokumentumleíró nyelv: XML, HTML, CSS, TEX vagy binárisan kódolt utasítás a megjelenítendő tartalomra és formára.

Nem informatikáról lenne szó, ha nem kellene azonnal pontosítanom: a HTML5 nem (csak) dokumentumleíró nyelv, webes dokumentumok mellett webes alkalmazások, programok készítésére is alkalmas. A HTML dokumentumleíró nyelvből programozási nyelvvé fejlődött.

Mivel a weblap az internet dokumentuma, sokféle hardveren, operációs rendszeren futó sokféle böngészőnek kell ugyanazt megjelenítenie, ezért jól látható a fejlesztőkörnyezet, az adott nyelven írt kód és a futtató (interpretáló) eszköz és szerepük is. A dokumentum kódja ezért a webes dokumentumok esetén „közismert". A HTML (és CSS) oktatásával a dokumentumleíró nyelvek tanulása közben a programkészítést is – akarva vagy akaratlanul – előkészítjük. A kódolás, a szintaktika ellenőrzése és javítása, a futtatás böngészőben (mint interpreterrel), a tesztelés és a szemantikai hibák javítása a programkészítéshez nagyon hasonló módon történik. A fejlesztő környezetben írjuk a kódot, amit a futtató környezet értelmez. A weblapkészítés esetén a fejlesztőkörnyezet lehet egy egyszerű szövegszerkesztő, egy WYSiWYG weblaptervező eszköz, egy portálba integrált tartalomkezelő – amely saját jelölőnyelvvel is rendelkezhet és lehet menüvezérelt is. Az előállított kód a fejlesztés eszközétől független, a különböző eszközök az emberi munka, az önkifejezés hatékonyságában, a szemantikai és szintaktikai hibák kezelésében adnak eltérő támogatást. Egy WYSiWYG weblapszerkesztőben demonstrálható, hogy a kijelölt szöveg formázásához megnyomott gomb hatása a kódban a megfelelő nyitó- és záró tag beszúrását jelenti. A kijelölés meghatározza a blokkot, nem lehet hibásan elrendezni, nem lehet elírni. Ugyanez történik az összes dokumentumkészítő alkalmazásban, de még a pixelgrafikus képszerkesztőkben is. Mivel általában csak formátum – passzív tulajdonságok, attribútumok – beállításáról van szó egy dokumentumban, ezért megoldható menüből. Amint aktivitása lesz a dokumentumnak, azt a fejlesztőkörnyezet külön szerkesztőjében adjuk meg, külön eszköz kell a hibakezelésre (ami nem WYSiWYG). Például a bemutatókban az animáció és áttűnések szerkesztése, a videószerkesztőkben a timeline, a táblázatkezelőben a képletszerkesztő és a hibakereséshez a képletkiértékelő eszközök. Mindegyiknél megfigyelhető, hogy a kódolást vizuális elemekkel segíti a fejlesztőprogram.

Figyelemre méltó, hogy WYSiWYG weblapszerkesztővel a hatékony munkához a kódnézet is szükséges. Táblázatkezelő programban függvényvarázsló, függvénytündér, azaz függvényszerkesztő panel segíti a szintaktikai hibától mentes függvény létrehozását, de a függvények szöveges beírása a kódkiegészítő támogatással sokkal hatékonyabb az összetett függvények létrehozásakor, szerkesztésekor. Adatbáziskezelés feladatokat táblázatkezelőben is meg lehet oldani, ilyenkor az adatbázis-kezelő alkalmazások lekérdező rácsához hasonló módon használjuk a táblázatkezelőt. Adatbáziskezelőkben lekérdező ráccsal könnyen „összekattintgathatók" az egyszerű lekérdezések, de a generált SQL kódban a megnevezések és a feltételek zárójelezése rendkívül redundáns. A programkészítésben is ugyanez várható. A vizuális fejlesztőeszközök redundanciával segítik a programkészítést. A blokknyelvek – úgy tűnik – nem programozási

nyelvek, hanem a fejlesztőeszköz kódmegjelenítő sablonjai, képi megjelenése a „snippet”-eknek. A blokknyelv szövegalapú kódolás lehetőséggel kiegészítése nem hozzáadott képesség, hanem kikerülő hozzáférési lehetőség ahhoz az implementációhoz, amelyet a blokkok alapján generál a fejlesztő környezet. Ebből következik, hogy egy adott bonyolultság esetén a köztes fordítási lépésekben nehezebb lesz a kód optimalizálása, korlátozott lesz a funkcionalitása.

Az informatikaoktatás során a számítógépes problémamegoldást kétoldalról közelítjük. Egyrészt a számítógépes alkalmazásokkal előállított termékek a számítógép funkcionalitását egyre jobban kihasználják: kezdetben statikusak majd animáltak végül némi intelligenciával is rendelkeznek. Másik irányból a megoldások minőségében az előregyártott intelligens komponensek összeillesztésétől haladunk a komponensek egyedi elkészítése irányába. A két szál találkozása a vezérlési struktúrákból építkező, egyedi, szöveges programírás, a fordítás, a futtatás, a tesztelés és hibajavítás képessége.

Programozás

A programozás jóval több, mint programkészítés. A programozásnak része az adat- vagy objektummodellezés, valamint az algoritmus tervezése is. A programkészítés a programozás fizikai megvalósítása, miközben a programozás jelentősebb részben szellemi tevékenység. Jellemző, hogy a *Programozási alapismeretek* (ELTE IK) tárgyat és a *Programozás alapjai* (BME VIK) tárgyat egymáshoz képest nagyon eltérő, a tantárgy tekintetében lényegében azonos elveken, de többféle nyelven tanítják – C++ és Pascal, illetve C és Python – a fejlesztőeszközre pedig több alternatívát is kínálnak.

A „világ” azt igényli, hogy mindenki tudjon programozni. A NAT eredménycéljai között viszont nem szerepel, hogy a diák tudjon programozni, csak az egyes résztevékenységeket kell tudnia, a programozáshoz szükséges fogalmakat kell értenie, az algoritmusokat kell ismernie. Ez kevés, LAU^{L}1-3 szint. Nem elég érteni és ismerni, a szükséges pillanatban fel kell ismerni, hogy épp melyik ismeretre van szükség és alkotó módon kell tudni használni. Csak az alapokat, de azt LAU^{L}5c szinten kell tudni.

Programkészítési minimum

A programozást a táncművészethez hasonlítva: a közoktatásban meg kellene tanulni járni. A tanulónak képesnek kell lennie egy adott feladat vagy probléma esetén meghatározni a kapott adatok típusát, a megoldáshoz szükséges adat- vagy objektum-struktúrát. Meg kell tudni fogalmaznia a vezérlési szerkezetek szintjén a megoldás algoritmusát és a kimenetnek, az eredménynek formáját. Ezek alapján el kell tudnia készíteni a programot (azt a programot, ami a feladatot

megoldja), tudnia kell a megoldását tesztelni, a hibákat önállóan kijavítani. Tudnia kell a programjához felhasználói dokumentációt készíteni, amelyben leírja a program működésének feltételeit és korlátait, a kezelés módját, továbbá tudnia kell publikálni a kész programot, illetve kezelni a kódoláshoz, a fejlesztéshez szükséges forrásokat.

A programozástudás „tipegő” szintjének jellemzői:

- a feladat megoldásához néhány elemi adat vagy string kezelése szükséges,
- az algoritmus bonyolultsága az egyszerű programozási tételeknek megfelelő
  - for(){if(){}} vagy
  - több ciklus egymás után, vagy
  - ciklusmentesen több feltétel kombinálása;
- a feladat önálló megfogalmazását (pontosítását) követően a megoldás önálló megtervezése;
- kódolás;
- a kódolási hibák önálló javítása, futtatással tesztelése;
- az eredmény értelmes megjelenítése;
- az elkészült program bemutatása, a program publikálása felhasználási tájékoztatással;
- beszámoló az adatmodell az algoritmus és kód bemutatásával a megoldás során felmerülő problémákról és döntésekről.

A programozás „tipegő” szintű teljesítése azt jelenti, hogy a tanuló programozással teljesen önállóan megoldott egy feladatot, ezzel tapasztalatot szerez arra vonatkozóan, hogy képes programozni, programozási problémát megoldani.

6.2
6.2.4
6.3