Teaching Python Considered Harmful? A strukturált programozás Python nyelven tanítása káros

Szalayné Tahy Zsuzsanna

sztzs@inf.elte.hu
ELTE IK

Absztrakt. A Python programozási nyelv alapjai a programozásban teljesen járatlanok számára könnyen tanulható, a bőségesen rendelkezésre álló nyelvi modulok sok területen jól használhatók. Ezek és hasonló érvek teszik a programozás alapjainak oktatásához divatossá a Python nyelvet. A 2020-as NAT kötelező tananyaga az algoritmizálás és programozás, így ez a téma is külön fejezetként szerepel az egyetlen állami tankönyvben, amely – követve a divatot – a Python nyelvet használja. A dokumentumok elemzése során megvizsgáljuk, hogy a pythonic kódolási stílus egyedisége mennyire segíti vagy gátolja az algoritmizáláshoz, strukturált programozáshoz szükséges fogalmak elsajátítását, a gondolkodási és kódolási kompetenciák fejlesztését.

Kulcsszavak: programozás oktatása, programozási nyelv választása, Python, strukturált programozás

1. Bevezetés

1968-ban „*Go-to statement considered harmful"* címen jelent meg Dijkstra levele [1], amelyben a strukturált programozást javasol a gépikódnál magasabb szintű programok írásához. A levél eredeti címe „*A Case against the GO TO statement"* arra utal és a levél tartalma azt fejti ki, hogy spagettikód egy összetett, nagyobb program esetén káros. A publikált, frappáns cím a gépikódnál magasabb szintű nyelvekre érvényes strukturált programozás [2] leggyakrabban idézett mondatává vált [3].

Az akkori, gépikódra optimalizált `goto` használatához hasonlóan, napjainkban az emberi kommunikációra optimalizált Python nyelv tanulása divatos. A divatot meghatározó előnye, hogy elsődlegesen objektumorientált; „mindenre" van kész modul; ingyenes. Ezek az előnyök érvényesülnek a mesterséges intelligencia programozása területén, az IoT fejlesztések során, a szoftver prototípusok készítésekor, más tudományágak informatikai támogatása során. Python nyelven a gyakran használt nyelvi eszközök használata könnyű (szinte angol nyelven írt egyszerű mondat); ezért a nyelv tanulásának első lépései sikerélményt adnak, motiválnak.

2. A Python nyelv szerepe az iparban és a felsőoktatásban

Az előnyöket érvényesítendő, a BME VIK ipari szereplőkkel közösen szervezett – duális – üzemmérnök-informatikus képzésén, a programozási alapismeretek első szemeszteres tantárgyban[1] (2018–2022 években) a Python nyelvet használják. Az oktatási szakértőkkel beszélgetve azt halljuk, hogy „az ipar a Python ismeretét várja el, ezért mindenhol Pythont tanítanak", de az állítás ellenőrzése során nem

[1] BME-VIK Bprof képzés „A programozás alapja" tantárgy adatlapja: https://portal.vik.bme.hu/kepzes/targyak/VIEEBA01/ [Megtekintés: 2022.10.31]

egészen ezt tapasztaljuk: A 2019-es Craft Conference[2] kiállítóit kérdezve, a Python „kezdetnek jó az is" nyelv, a cégek termékei jellemzően C++, C#, Scala nyelven készültek.

Másik végletként figyelemre méltó az ELTE IK-n zajló képzés. Az elmúlt években volt, hogy a felsőfokú szakképzésen Python nyelven tanították a Programozási alapismereteket, néhány évig az *Imperatív programozás* tantárgy keretében a C nyelv pointere tárgyalása helyett Pythont tanultak a hallgatók, de a 2022-es tanévben a bevezető, *Programozás* tantárgyban a strukturált programozás alapjait C# nyelven[3], az *Imperatív programozás*t C nyelven tanítják[4]. Ezzel szemben a korábban IK-n is tanuló matematikusok ma már a TTK keretei között csak Python nyelvet használnak[5].

Az ELTE TTK-n tanított „*Bevezetés a tudományos programozásba gyakorlat*" tantárgy tematikájában feltűnő, hogy az objektumorientált paradigma mellett még utalás szintjén sem fordul elő sem a strukturált programozás, sem a programozás típusalgoritmusai (programozási tételek). A tantárgy fókuszában a Python matematikai moduljai állnak: numpy, matplotlib, pandas.

Kibővítve a vizsgálódást, az egyetemi képzésben a Python szerepéről online tájékozódhatunk. Magyarországi egyetemek körében:

- BME VIK BSc mérnökinformatikus képzésén C[6] nyelvvel kezdenek.
- Az Óbudai Egyetem NIK-en „Szoftverfejlesztés és -tervezés I" tantárgy[7] C# nyelvet használ.
- A Pázmány Péter Katolikus Egyetemen saját PLang[8] nyelven tanítják a strukturált programozást.
- A Szegedi Tudományegyetemen C nyelven implementálják a programozás alapjait.[9]
- A Corvinus Gazdaságinformatikus[10] képzésén a C# nyelvet preferálják.
- Semmelweis Egyetemen orvostanhallgatóknak az általános informatika képzés részeként Pythont tanítanak.[11]

  Nézzük néhány ismert külföldi egyetem képzési tájékoztatóját is:

- A Columbia[12] egyetemen hét hetes, heti kétórás tárgy keretében lehet Pythont tanulni.

---

[2] Craft Conference 2019 (szervező: Prezi és IBM Budapest Lab): https://craft-conf.com/2019 [Megtekintés 2022.10.31.]

[3] ELTE IK: *Programozás*. Programtervező informatikus nappali és esti tagozat részére: https://progalap.elte.hu/ Programtervező informatikus felsőoktatási szakképzés: https://progalapfsz.elte.hu/ [Megtekintés 2022.10.31.]

[4] ELTE IK: *Imperatív programozás*. http://gsd.web.elte.hu/imper/ [Megtekintés 2022.10.31.]

[5] ELTE TTK Matematika Intézet: *Bevezetés a tudományos programozásba gyakorlat.* https://www.math.elte.hu/kepzesek/bsc-alapkepzes/bsc-matematika-tantervi-halo-tervezo/#verboseHook [Megtekintés 2022.10.31.]

[6] BME VIK EET: *Programozás alapjai 1*. https://infoc.eet.bme.hu/utemterv/ [Megtekintés 2022.10.31.]

[7] ÓE NIK: *Szoftverfejelsztés és -tervezés I.* OE-NIK-AII, 2017 https://docplayer.hu/126600453-Szoftvertervezes-es-fejlesztes-i.html [Megtekintés 2022.10.31.]

[8] Pázmány Péter Katolikus Egyetem ITK: *Bevezetés a programozásba 1*. https://docplayer.hu/6664391-Ppke-itk-bevezetes-a-programozasba-1.html [Megtekintés 2022.10.31.]

[9] SzTE TTIK Informatikai Intézet: *Programozás alapjai.* http://www.inf.u-szeged.hu/kurzusleirasok/IB103.xml [Megtekintés 2022.10.31.]

[10] Budapesti Corvinus Egyetem Információrendszerek Tanszék: *Szoftvertechnológia I.* http://sirius.uni-corvinus.hu/root/web/web.nsf/do?open&lang=hu&page=oktatas [Megtekintés 2022.10.31.]

[11] Szemmelweis Egyetem Egészségügyi Közszolgálati Kar: Bevezetés az információs technológiákba (2) – Programozás https://semmelweis.hu/dei/files/2020/09/Programozas_20220607.pdf [Megtekintés 2022.10.31.]

[12] Columbia University Computer Science Department: *COMS W3101-1 - Programming Languages. Python.* http://www.cs.columbia.edu/~bauer/cs3101-1/ [Megtekintés 2022.10.31.]

- Az ETH Zürich tájékoztatójában[13] jellemzően a biológiával kapcsolatos képzéseken (42,5%) Pythont tanítanak, a képzések további 37,5%-án C++, 20%-ban Java a kezdő nyelv.
- A Harvardon David Malan nem programozóknak szóló, online is tanulható kurzusán[14] az első héten bemutatott nyelv a Scratch, majd C követi. Egy előadásban van szó a Python nyelvről.
- A MIT[15]-en azoknak tanítanak Pythont, akiknek nincs programozási előismerete.
- Cambridge egyetemén a felvételizőknek ajánlják[16], hogy valamilyen nyelvet tanuljanak, ami lehet a divatos Python.

Konklúzió:

Az ELTE TTK matematikus szakján nem tanítják a strukturált programozást, de vélhetően használják. Ahol a Python nyelvet használják a programozás tanítására, ott a programozás kvázi alkalmazói ismeret; az algoritmizálás absztrakciója, tipizálása nem tananyag. Az informatikus (Computer Science) képzésben kivétel nélkül a szakma alapjainak elsajátítása a strukturált programozás oktatását helyezi középpontba, a Python nyelv nem szakmai, hanem a holisztikus tanulási/tanítási módszer eszközeként kerül a tananyagba.

3. Programozás, mint az általános műveltség része

Főként a külföldi példákon érzékelhető, hogy a programozás egyetemi oktatása felzárkóztató jellegű, aminek során egy, ma már mindenki számára szükséges kompetencia megszerzését biztosítják. Ezeket a kompetenciákat találjuk meg a DigComp 2.2: The Digital Competence Framework for Citizens [4] elvárásai 3.4-es pontjaként. A témához képest tömör megfogalmazásokhoz készített tanulmány – Programming for All: Understanding the Nature of Programs [5] – a programozással kapcsolatos kompetenciákat nyelvfüggetlenül írja le.

A korábban ECDL, újabban Európán túl is elismert bizonyítványt adó ICDL vizsgák fejlesztése a DigComp kompetenciákhoz igazodik. Egyik új (2021-es) modul a *Kódolási ismeretek (Python)*. A modul tananyaga, a syllabus 1.0 [6], csak magyar nyelven található meg, bár ausztrál, román és szlovák tematikában is szerepel. A tananyag néhány részlete:

3.2.6 Összetett adattípus használata egy programban, például: tömb, lista, tuple (tároló).

4.2.2 Ciklus típusok: elöl tesztelő, hátul tesztelő és számláló (FOR, WHILE, REPEAT).

4.2.3 Ciklus használata egy programban.

4.4.4 Függvény írása és elnevezése egy programban.

4.5.1 Az esemény ismerete. Az esemény célja egy programban.

4.5.2 Eseménykezelők használata, pl.: egérkattintás, billentyű lenyomása, gombkattintás, időzítő.

[13] ETH Zürich: *Programming for beginners.* https://ethz.ch/en/studies/bachelor/beginning-your-studies/subject-related-preparation/programming-beginners.html [Megtekintés 2022.10.31.]

[14] Harvard University: *CS50: Introduction to Computer Science.* https://online-learning.harvard.edu/course/cs50-introduction-computer-science [Megtekintés 2022.10.31.]

[15] Massachusetts Institute of Technology *Programming & Software Engineering.* http://catalog.mit.edu/subjects/6/ [Megtekintés 2022.10.31.]

[16] University of Cambridge, Department of Computer Science and Technology: *Admission FAQ.* https://www.cst.cam.ac.uk/admissions/undergraduate/faqs [Megtekintés 2022.10.31.]

A tananyag jelentős része a strukturált programozásról szól, amihez Pythont használ. A nyelv választásának oka, a népszerűsége. A holisztikus tanuláshoz – könnyen tanulható – a Python az elsőnek ajánlott nyelv. A cél és eszköz egymásmellé helyezését megfigyelhetjük a syllabusban:

- A folyamatábra és pszeudokód készítéséhez a tömb és rekord (adatstruktúra) fogalma szükséges, de Pythonban ezek nem léteznek. A 3.2.6 pont alapján a tömböt helyettesíti a lista, a rekord helyett lenne a tuple, ami alkalmas vegyes típusú adatok tárolására, de ez az adattípus Pythonban nem módosítható, ráadásul bár több nyelvben létezik ez az adattároló, nagyon eltérő a szerepük, tulajdonságaik. Kérdéses, hogy a „tároló" kiegészítés miért szükséges és valójában mi lesz a kapcsolat a pszeudokód és a Python között.
- Ugyancsak kérdéses, hogy ha az összetett adattípusok a lista és a tuple (3.2.6), akkor milyen adat lesz az, amire értelmezhető az egérkattintás (4.5.2). A tananyagból az OOP ismeretének részletezése hiányzik, pusztán az eseménykezelés utal arra, hogy a rajzon vázolt algoritmus és konzolos alkalmazások készítésén túl grafikus felületű alkalmazást is tudni kell készíteni.
- A 4.2.2 pontban megjelölt ciklus típusok a strukturált programozás vezérlési elemei, az eredeti szövegben csupa nagybetűvel kiemelt vezérlési kulcsszavak a Pascalra emlékeztetnek. A három típusból csak kettő adott a Python nyelvben. Vajon milyen működő programmal lehet kipróbálni a hátul tesztelő ciklust? Vajon a számláló ciklus absztrakciójában a ciklusváltozó növekedni fog vagy végig lépked egy tartomány elemein, az egymást követő számokon?

Az egymásmellé rendelt tartalmak alapján úgy tűnik, hogy a kódolási ismeretek megszerzésének két része van. Az egyikhez felfedező, kísérletező tanulás társul, aminek során Python nyelven kódolt alkalmazások készülnek. A másik egy elméleti tananyag, ahol rajzolni vagy írni kell számítógépen, de ennek kevés köze lesz a működő programhoz. Így az is kérdéses, hogy miért szükséges és az is, hogy mi ezekben (az algoritmus leíró módszerekben) a motiváló.

További kérdés, hogy mi az a tudáscsomag, aminek ismeretét a Kódolási ismeretek (Python) vizsgamodul méri, mi teremti meg az elemi adattípusok és az eseménykezelés között a kapcsolatot. A vizsgamodul esetleg mérheti egy robotnak, annak korlátozott számú részegységének a programozásában való jártasságot. Ebben az esetben a programozás alapfogalmait: a kódolást, az adattípusokat, a vezérlési struktúrákat egy robotra, egy eszköz vezérlésére vonatkoztatva kell ismerni.

Kétségtelen, hogy napjainkban nagyon fontos a robotok (és mesterséges intelligencia) programozásának ismerete. Valószínű, hogy az orvosi egyetem és az ETH Zürich biológia szakjain is többnyire az automatizált és mesterséges intelligenciával bíró eszközök kezeléséhez szükséges a programozás, az első évben, ezért ennek alapjait tanítják. Ugyanakkor a DigComp-ban leírt elvárások és egyes egyetemi képzések azt mutatják, hogy a digitalizáció a robotok (eszközök) programozásánál nem áll meg, ezért a problémamegoldási készség fejlesztését nem szabad leszűkíteni egy-egy robot vezérlésére. A digitális kompetenciáknak része a nagy rendszerek működésének értelmezése és nagy adathalmazokban tájékozódás is.

Egy gondolat erejéig visszautalva a bevezetésre: Dijkstra a `goto` programozásban való alkalmatlanságát akkor vetette fel, amikor már nagyobb, összetettebb problémákat kellett megoldani. [1][3] Ma is használják a vezérlésátadó utasításokat gépikódban és használják magasabb szintű nyelvekben is. A strukturált programozás elvei a tervezés során tiltják a vezérlés átadását, az átláthatatlan spagettikódok írását. A Python nyelv készítőinek is célja az átlátható, könnyen írható és olvasható kód, amit nem a strukturált programozás szabványos lépéseivel, hanem az összetett problémák modulokra bontásával ér el. A robotika OOP alapú programjai jellemzően eseményhez rendelt egyszerű eljárások, az összetett rendszerek kicsi moduljai. A címkére ugrások helyett események vezérlik a program működését: egérrel kattintás, gomblenyomás, erős hang, időzítő `tick` eseménye…

Gonosz kérdés: A program érthetőségét mennyiben változtatja meg, ha sok `goto` helyett ugyanannyi függvénydefiníció van? A válasz kitérő: az egyes függvényeket jó programozók írták, tesztelték,

ezért nem kell hibát keresni benne, jól működnek. Előregyártott megoldások bőséges kínálata megkönnyíti munkát [7] anélkül, hogy a részletekkel, a „nehéz matematikai absztrakcióval" foglalkoznunk kellene [8]. Ebből a megfontolásból következik, hogy nem fogunk belezavarodni a spagetti kódba, de nem is fogjuk tudni, hogy valójában hogyan működik a programunk.

Tényleg meg lehet tanulni úgy gondolkodni, mint egy „Computer Scientist" informatikus azzal, hogy Python nyelven programokat írunk?

Esetelemzés:

Egy csoportban C# nyelven tanítottam programozást, de az egyik diák inkább a barátjával – egyénileg – Python nyelven akart megtanulni programozni. Hagytam… Azt látták, hogy rengeteg forrás van mindenféle érdekes programok megírásának tanulásához. Egyik tanórán egy dátumok összehasonlítását is tartalmazó feladatot kellett megoldani. Felírtuk a megoldáshoz szükséges ciklust, elágazást és logikai formulát. A diák pedig hozta a sokkal rövidebb megoldását, amiben a Python három kész függvényét használta: „minek erőlködnek a többiek a megoldással, ez így érthetőbb és működik." Az ő kódja mindig rövidebb, frappánsabb volt, sokszor kombinálnia sem kellett, csak megkeresni a kész eszközt. Amikor a csoport haladó diákjai, a „menő programozók", a bmp fájl bináris olvasására és módosítására írtak programot, megváltozott a véleménye. A csoporttársai korábban tanult eszközöket (elágazás, ciklus, elemi adatok, tömb, osztály…) kombinálva módosítottak képeket, neki új eszköz megismerésére lett volna szüksége. A következő félévben már C#-ot tanult. A váltásnak lényegében az volt az oka, hogy hiába írt sok tucatnyi programot, nem tanult meg programozni, nem tanult meg úgy gondolkodni és kreatívan alkotni, mint a társai.

A korábban bemutatott példák alapján, azokon az egyetemi képzési helyeken, ahol a programozás általános problémamegoldási eszköz, a Python nyelv ismerete nem elvárt. Nem a Python nyelv az, amin a programozás alapjait, gondolkodásmódját megismerik, de konkrét területeken a nyelvet önállóan megtanulva, használják.

Konklúzió:

Úgy tűnik, a `goto`-hoz hasonlóan, a Python sok esetben jó, de az alapokat, a strukturált programozás általános elveinek elsajátítását nem támogatja. A megszerzett tudás nem transzferábilis.

4. Programozás oktatása a középiskolában

A nem informatikai egyetemi szakokon a bevezető programozás oktatás sokszor hiánypótló, felzárkóztató, kivételnek tűnik a fenti magyar példák közül az ELTE TTK programozás tantárgya. A DigComp, az ICDL minősítés is elsősorban a már felnőtt állampolgárok számára adnak iránymutatást, itt is fő cél a technológiai fejlődés követése, a lemaradók támogatása. A közoktatás, ezen belül a középiskola feladata azonban nem a jelenre, hanem a jövő szükségleteire való felkészítés. A Nemzeti Alaptanterv egy 12 éves képzési program, aminek a végére a végzéskor szükséges kompetenciákat kell biztosítani. A társadalmi-gazdasági problémát is okozó lemaradás hatására a 2020-as Nemzeti Alaptantervben a korábbinál hangsúlyosabban jelenik meg a programozás.

4.1. Hozott ismeretek

A 2020-ban bevezetett Nemzeti Alaptantervben a programozás, az algoritmizálással és robotikával ismerkedés már az óvodákban elkezdődik.

A Robot Turtles – magyar nyelvű kiadásban Robot Teknősök – társasjáték hagyományos, kartontáblás játék, nem kell hozzá elem sem. A játék során a gyerek tanul számlálni, a vezérlési struktúrák közül a szekvenciát és a modul kiemelését (több kártya helyettesítése eggyel) gyakorolja. Négyéves kortól, már óvodás korban ajánlott.

Az elemmel működő játékok között számos programozható játékot találunk, amelyeket kártyák helyett gombokkal, színes kockákkal, színes vonalakkal lehet vezérelni, így fejlesztik a programozási készségeket.

Robot Teknősök társasjáték[17]

Alsó tagozaton – a tantervkészítők elképzelése szerint – az egyre többet megértő játék robotok programozása során a gyerekek megismerkednek néhány fejlettebb vezérlési eszközzel.

Ötödik osztálytól a tankönyvi tananyagba [7] rejtve „Robotika, algoritmizálás, programozás" címen megjelenik a programozás. Ezen belül a pszeudokód, algoritmus, szekvencia, elágazás és ciklus fogalma. Programozási nyelvként a Scratch a leginkább ajánlott, de van utalás a magyar oktatásban nagy hagyományú Imagine Logo programra is. A programozás jellemzően a robotikával kapcsolatos feladatok megoldását jelenti, ami, ahol erre van mód akár 5. osztálytól kezdve micro:bit programozása is lehet. A tankönyv online „okostankönyv" verziójában [10] megoldási útmutató és működő program is megtalálható ehhez. A motivációt az elkészült játék adja, amelyet mintakövetéssel (megoldási útmutató) készít el a tanuló.

A felső tagozatos kerettanterv [11], a tankönyvi tananyagok (5.[7], 6.[12] és 7.[13] osztályra), ezek online okostankönyv feldolgozásai (5.[10], 6.[14] és 7.[15] osztályra) és feladatgyűjtemény[16] alapján felsőtagozat végére algoritmus, robotika és programozás témakörökben a diákoknak gyakorlati tapasztalatuk lesz az ICDL Kódolási ismeretek (Python) vizsgájának legtöbb témájában. A témakörök elnevezése is mutatja a hangsúlyáthelyezés irányát: 5. osztályban a „robotika" a cím első, később az utolsó eleme. Kiemelendő, hogy a programok nem karakteres – és így nem is Python –, hanem blokk alapú programozási nyelven készülnek. Az egyetemi bevezető/felzárkóztató programozás tantárgyak – az alapján, ahol elérhető a tananyag – az általános iskola végére elvárt ismeretektől csak a programozási nyelvben térnek el. Az ICDL modulhoz képest ezen túlmenően hiányoznak a tananyagból az elméleti definíciók.

A Python nyelv kezdő, motiváló nyelvként történő tanítását a felső tagozaton általában nem lehet kihasználni, mert a tanulók nem tudják kezelni a billentyűzetet. Ezért a kezdő nyelv egy blokk nyelv. A vizuális környezet, a vezérlési struktúrák és adatok értelmezést segítő jellegzetes alakú és színű blokkok a 4–14 évesek körében motiválók. Ebben a korban az elemek pakolása kevesebb frusztrációval jár, mint a billentyűk keresése, gépelési hibák javítása. (Frusztrációt okoz például a gyakori fókuszváltás, az, hogy a billentyűzetre le kell nézni.) Azonban haladóbb csoportokban a blokk-programozásról át lehet térni karakteres kódolásra. Erre alkalmas a Python nyelv, mivel Logo teknőc, micro:bit és Lego robot programozásához is van Python modul, így a már elkészített programok alapján elkészíthető a Blokk–Python szótár. A billentyűzet kezelés nehézségei is elviselhetőbbek, ha egy programon belül lehet váltogatni a programozási nyelveket, a blokk és kód nézetet.

Mivel a tantervet felmenő rendszerben vezetik be és a tankönyveket sem adják ki a bevezetés előtt, a 8. évfolyam programozás témaköre még nem ismert. Az alacsony (heti 1) óraszámra és a táblázatkezelés témakör súlyára tekintettel, elképzelhető, hogy robotikára, programozásra egyáltalán nem lesz

[17] RobotTurtles társasjáték kép forrása: kép forrása: https://www.thinkfun.com/products/robot-turtles/ [Megteintés: 2022.11.01]

idő 8. évfolyamon, a gondolkodási készségeket adatok kezelésén és függvények használatán keresztül fejlesztik. Ezt támasztja alá az is, hogy a tanterv 7-8. évfolyamos követelményeiben programozás témakörben javasolt tevékenység: „Típusalgoritmusok – összegzés, másolás, eldöntés, maximumkiválasztás – használatát igénylő programozási feladatok megoldása" inkább az algoritmus elvi vagy eljátszással történő megvalósítását sugallja, a blokk-programozás ilyen típusú feladatok megoldására nem optimális. Ehelyett motivációt jelenthet, hogy táblázatkezelésből a tananyagban megtaláljuk a típusalgoritmusokat megvalósító statisztikai függvények is.

4.2. Programozás oktatása a középiskolában, a tantervek alapján

A gimnáziumokra érvényes Digitális kultúra tantárgy kerettanterv [17] alkalmazandó a szakgimnáziumi képzésre is [18] A szakképző iskolákra külön, összesen 32 tanórában tanított Digitális kultúra tanterv [19] van. Ebben a programozásra 6 tanóra jut. Figyelemre méltó, hogy előzetes ismeretként elvárja egy fejlesztői környezet használatát, valamint, az is, hogy programot írni az összetett adattípusok és egyszerű algoritmusok értelmezése alapján kell; az alkotás, a kitalálás nem követelmény.

A hátrányos helyzetű középiskolás korú tanulók segítését célozza meg az Arany János Tehetséggondozó Program (AJTP). A program része a tanulók ECDL alapvizsgára történő felkészítése [20]. A programban heti kétórás tárgy az alap modulok vizsgáira készít fel [21], a három- [22] és négyórás [23] modul emellett az általános iskolai digitális kultúra tananyagnak felel meg, így a robotika, algoritmizálás és blokk programozás tanulására is van lehetőség.

Az új NAT bevezetése 2020-ban az 1. 5. és 9. évfolyamokon indult, jelenleg a középiskolákban digitális kultúra tantárgyat tanuló diákok nem tanultak digitális kultúrát az általános iskolában, így pillanatnyilag nem számolhatunk az általános iskolai digitális kultúra ismeretek meglétével. A 2012-es tantervben az Informatika óraszáma a 2020-as tanterv szerinti digitális kultúra óraszámának a fele volt.

A középiskolai tanterv erősen épít az általános iskolai követelményekre: „*A 8. évfolyam végére a tanulók a digitális írástudás alapjainak elsajátítását lezárták.*" [17 p. 4.] A tanterv bevezetője kiemelten foglalkozik a programozással: „*A programozás és algoritmizálás témaköreiben a tanulók új kihívással találkoznak. Míg korábban a blokkprogramozás segítségével gyakran közvetlenül vezéreltek eszközöket, most magasabb szintű absztrakciót igénylő feladatokat oldanak meg hagyományosnak nevezhető, azaz a programkód közvetlen beírását elváró fejlesztői környezetben.*" A robotika középiskolában nem digitális kultúra tananyag, valószínűleg a többi tantárgyban a robotika a tanulás egyik eszköze kellene legyen. Az algoritmizálás és programozás területén magasabb szintű absztrakció célul való kitűzése viszont azt jelenti, hogy az „alacsonyabb szintű absztrakció" meglétét elvárja. Tovább építkezik, ezért elmondhatjuk, hogy a 2020–2023-ban induló 9. évfolyam hátrányos helyzetű, számukra az AJTP-hoz hasonló kiegészítő képzés lenne szükséges a tantervi követelmények teljesítéséhez.

A felső tagozatos [11] és gimnáziumi tanterveket [17] összehasonlítva nagyon szoros egymásra épülés, a spirális (1-2 évente visszatérő ismétlő-fejlesztő témakörök) tantervi struktúra látható. Ezzel szemben feltűnő, hogy

- a 7–8. osztályos tanterv [11 p. 13] javasolt tevékenységei között szerepel a strukturált programozásra utaló típusalgoritmusok használata:

  „– *Vezérlőszerkezetek tudatos választását igénylő blokkprogramozási feladatok megoldása. Típusalgoritmusok – összegzés, másolás, eldöntés, maximumkiválasztás – használatát igénylő programozási feladatok megoldása*"

- de a 9–10. osztályos tanterv [17 p. 5] ezt elkerüli.

  „– *Típusok, változók és vezérlőszerkezetek (szekvencia, elágazás, ciklus) tudatos választását igénylő feladatok önálló megoldása, a választás indoklása*

  …

  – *Feladat megoldása során a fejlesztői környezet lehetőségeinek használata (pl. tesztelés)*

  – *Feladatmegoldás strukturálatlan algoritmussal és függvények, eljárások használatával.*"

Első olvasásra talán fel sem tűnik a „**strukturálatlan**” kifejezés. Lehetne elírás is, ha nem tudnánk, hogy informatikus körökben milyen viták folynak a `goto` a `break` és egyéb vezérlésátadási utasításokhasználatával kapcsolatban. A strukturált programozásban az algoritmus vezérlő szerkezetekből épül fel, ezek ismeretét és tudatos választását írja elő a 7–8. osztályos (és az 5–8. osztályos [11 p. 5]) tanterv. A micro:bit programozáshoz javasolt makecode[18] fejlesztőkörnyezetben a kimondottan strukturált programozást támogató blokkok között, a ciklusok csoportban megjelenik a „folytat” (angolul „continue”) és a „kitörés” (angolul „break”) blokk, de csak kontrolláltan, az adott ciklusra vonatkozva használható, így a teljes algoritmus nem lesz strukturálatlan, a szigorúan strukturált megoldás formális átírással (blokkok átszervezésével) előállítható. Informatikai, programozásmódszertani szempontból a strukturálatlan algoritmus alkalmazása káros, csak akkor fogadható el, ha a vezérlési struktúrák (elágazás, ciklus) nem állnak rendelkezésre.

Felmerülhet, hogy oktatási szempontból lehet pozitív hozadéka a strukturálatlanul kódolt megoldásoknak. Kérdéses, hogy a tananyagban hogyan jelenik meg a strukturálatlan algoritmus készítése, de az már most elmondható, hogy az általános iskolai tananyagra alapozva nincs létjogosultsága, az adott korosztálynak további javasolt tevékenységeihez nem kapcsolódik, a fejlesztési célok elérését nem támogatja. Elképzelhető, hogy egy diák strukturálatlan algoritmust készít, de ez legfeljebb része lehet a *Fogalmak* részben elvárt „tervezési folyamat”-nak, vagy tanulságos példája a tesztelésnek. A digitális kultúra tantárgyi követelmények teljesítése szempontjából egy strukturálatlan algoritmus programozásból annyit ér, mint szövegszerkesztés témakörben a szóközökkel középre igazított cím.

4.3. Programozás témakör megjelenése a Digitális kultúra tankönyvekben

A digitális kultúra tantárgyban az oktatásirányítás szándéka szerint az informatika oktatása (is?) új megközelítéssel szerepel. A tantárgy 3–11. évfolyamokon átívelő tantervét 2020-ban a 9. évfolyamon úgy vezették be, hogy a megelőző 6 év tananyagának áttekintésére, a tantervi követelmények fokozatos bevezetésére semmilyen központi terv nem volt. Tovább rontotta a helyzetet, hogy a tankönyvek is csak a már aktív évekre készültek el, így 2020-ban az 5. és 9. évfolyamra készült tankönyv. Ennek ellenére több témakörben a 9. évfolyam tananyaga épít a korábbi években tanultakra. Ez részben azért megalapozott, mert az adott témakör tanítását elvárták az Informatika tantárgyban is, másrészt azért lehetséges, mert az Informatikából elvárt minimum követelmények nagyjából megfelelnek a Digitális kultúra tantárgyban elvárt minimumnak. Praktikus megfogalmazással: lehet, hogy Informatikából a jeles Digitális kultúrából csak közepes, de aki Informatikából elérte az elégségest, annak ez a tudás Digitális kultúrából is elégséges.

Az Algoritmizálás, formális programozási nyelv használata [17] tantervi téma előzménye – hasonlóan a többi témakörhöz – az előző NAT felső tagozatos Informatika tantárgyában is megtalálható, az új tantervben ennek blokk nyelvvel kibővítése jelenik meg. Azonban a 2012-es kerettantervben az oktatásához nem volt óraszám, így jellemző, hogy egyáltalán nem tanulták a diákok, az átmeneti időszakban a Digitális kultúra tantárgyban 9. évfolyamon nem lehet az általános iskolai ismeretek meglétében bízni. Sajnos, a problémára nem egy átmeneti tananyag elkészítése lett a megoldás, hanem a középiskolai tananyag függetlenítése az általános iskolában tanultaktól.

Amennyiben a következő évben megjelenő 8. évfolyamos digitális kultúra tankönyvben van utalás a blokk nyelvről Python nyelvre áttérésre és a gimnáziumi tananyag építene az általános iskolában tanultakra, akkor ésszerű folytatás lehetne programozási nyelvként – legalábbis kezdetben – a Python nyelv választása. Az 5–8. osztályos Digitális kultúra feladatgyűjteményben – sem a programozás fejezetben [16], sem máshol – nem szerepel a Python szó. Ezért a megelőző tanulmányok nem indokolják, hogy a gimnáziumban a Python nyelv legyen a programozás nyelve. A Python nyelv választásának másik lehetséges magyarázata: „a Python nyelv jó a kezdőknek, motiváló, hogy könnyen lehet benne

[18] makecode: https://makecode.microbit.org/

programot írni" szintén nem állja meg a helyét, hiszen az első 4 évfolyam kivételével nem lesznek teljesen kezdők a diákok.

Az Algoritmizálás és (formális) programozási nyelv használata fejezetek 9. évfolyamon pdf [24] és okostankönyv formában [25] elvileg teljesen kezdőknek szól, a folytatás a 10. évfolyamon [26] [27] majd 11. évfolyam végére a [28] [29] tananyaggal lefedi a tantervi követelményeket.

Az okostankönyvek [25] [27] [29] statikus része megegyezik a pdf formátumban megjelent tankönyvek tartalmával (az elírások is), de az okostankönyvekben minden oldalon van 2-3 interaktív feladat is. Ezek sokszor a tananyagban szereplő új fogalmak gyakorlását szolgálják, máskor kiegészítik azt. A plusz feladatok és a szintén online megjelent feladatgyűjtemények [30][31] is nagyban segítik a tananyag elsajátítását, de egyben rámutatnak a tankönyv hiányosságaira is.

Ilyen kiegészítés a 9.-es okostankönyv [25] elő leckéjének (Mi az a programozás?) végén, a lecke utolsó tartalmi egysége (Kérdések) után található kipróbálható példa, ami a micro:bit programozásának Python nyelvű változatát mutatja be. Ha lenne visszautalás a korábbi évekre, akkor ezen a példán értelmezni lehetne a változó, a 10. évfolyamon tárgyalt eljárás és a 11. évfolyam végén érintett eseménykezelést is, alátámasztva, hogy jó döntés lehet a Python nyelvet választani az általános iskolai tanulmányok után is. Ha lenne…, de nincs.

Az okos(feladat)gyűjtemény két programozás fejezete pedig azt mutatja, hogy a téma feldolgozásának több módszere is lehet. Egyrészt a Python mellett megtalálható a C# nyelvű mintamegoldás is, másrészt az „Általános irány" [30] az adatok, vezérlési szerkezetek, típusalgoritmusok (programozási tételek) mentén csoportosítva jeleníti meg a feladatokat, míg a „Tehetséggondozó irány" [31] problémakörökre bontva, az egyes feladatokon belül – akár hosszú távú projektként – egyre több ismeretet igénylő részfeladatokat tartalmaz.

Konklúzió:

A rendelkezésre álló forrásokból látható, hogy a tankönyv csak egy példa a témakör feldolgozására, azonban ennek a feldolgozásnak biztosítania kell a tantervi követelmények teljesítését (az Oktatási Hivatal által akkreditált tankönyv). Az oktatási cél az algoritmizálás, a strukturált programozás alapjainak megtanítása, az ehhez választott implementálási eszköz a Python nyelv.

5. Python nyelven tanított algoritmizálás

A Digitális kultúra tantárgyhoz írt 9–11. évfolyamosoknak szóló tankönyvek és okostankönyvek Algoritmizálás és (formális) programozás fejezetei rögzített tanítási/tanulási útvonalat, tanítási tervet adnak. Mindegyik tankönyvhöz tartozik tanmenet, ami a tankönyvbéli fejezeteket adja meg egy-egy tanóra címének. A tankönyv megjelenésének formájából következően a témakör tárgyalására kötött volt a terjedelem, ezért több helyen rövidíteni kellett rajta. Ez eredményezte azt, hogy a tankönyv sokkal többet foglalkozik a Python nyelvvel, a programozás eszközének ismertetésével, mint az elméleti alapokkal, magyarázatokkal. A keletkezés körülményeitől függetlenül, a tanárok és diákok csak a végeredményt látják, ez alapján kell tanítaniuk, tanulniuk.

Ebben a fejezetben azt tekintjük át, hogy a tankönyv hogyan segíti a kerettantervben [17] előírt tevékenységek megvalósítását és a kompetenciafejlesztést.

5.1. Adattípusok

5.1.1. A karakter nem szöveg

Tanterv: „*ismeri a következő elemi adattípusok közötti különbségeket: egész, valós szám, karakter, szöveg, logikai;*" [17 p. 4]

A Python nyelv nem ismeri a karakter adattípust, pontosabban a szöveg adattípus karakterlánc, de a lánc eleme nem jelent külön típust, az is „lánc”. Ez némiképp ellentmond a matematikában már tanult halmazelméleti fogalmaknak, ott egy egyszerű halmaz eleme nem halmaz.

A karakter a 11-es tananyagban kap némi értelmet [28 p. 171]: „*Most, hogy már úgy írunk és olvasunk fájlokat, mintha mindig is remekül ment volna, ki kell térnünk egy fontos részletre. A számítógép a karaktereket számkóddal tárolja, mind a memóriában, mind a fájlokban.*” Úgy tűnik, hogy a bájt, a bit, a bináris adattárolás fogalmát misztifikálni kell. Mivel minden adatot bitekkel tárolunk és a bitek véges mennyiségével kell leírnunk, ezért ezek a bináris jelek (binary digits) 1 és 0 értékekkel értelmezve kettes számrendszerben „számkód”-ot eredményeznek. Valójában egy jelsorozat nem eredményez semmiféle „számkód”-ot, a felhasználás módjától függ, hogy karakternek vagy számként, ezen belül előjeles vagy előjel nélkülinek értelmezi a program.

A karakter „számkód” értelmezésére azért tér ki a tankönyv, mert a Windows és a Python között alapértelmezésben nincs meg az összhang a karakterkódolások értelmezésében. A probléma a tankönyv szerint a Windows hibája, a háttérben meghúzódó adatmodellezési kérdéseket nem tekinti át, inkább elkerüli az ékezetes karakterek használatát a fájlokban. Nemhogy 10. osztály végére [17 p. 4], de a tananyag befejezésekor se lesz tiszta, hogy mi a különbség a szöveg és a karakter között, például, hogy a szöveg – mint karaktersorozat – lehet üres, de a karakter nem. A szöveges fájl beolvasása során épp csak említés szintjén megjelenik az 1 karakter beolvasása (read(1))[28 p. 172], ami sikertelen olvasás esetén lehet üres szöveg is. Sehol nem jelenik meg, hogy az adatoknak mérete lenne, hogy a memóriában tárolása valahány bájt lefoglalásával jár.

A karakter adattípus a kódtáblákon keresztül természetes kapcsolatot jelent az adatmennyiség fogalmának értelmezéséhez. Megmutatja a rögzített, nyolcbites tárolási mód korlátait, a programoknak a számítógép-architektúrával (pl. 64 bites processzor), az elektronikával, robotikával való kapcsolatát. A Python nyelv ennek megtapasztalásától elzárja a használóját, jelen esetben a tanulót.

5.1.2 Az adattípus fogalom kialakítása

A Python nyelv dinamikusan típusos. A változók típusát az értéke határozza meg. „*Legegyszerűbb, ha a változókra olyan dobozként gondolunk, amibe bármit tehetünk*” A változónak ez a fajta értelmezése elfedi a digitális világ alaptéziseit. Két dolgot nagyon pontosan kellene látnia a tanulónak: a „bármi”-nek fizikai mérete van, méghozzá nyolcszor annyi elemi tárolócellát foglal le, mint ahány bájtos, továbbá, minden tárolóhely véges, így a doboz mérete is véges. A dinamikus típusosság esetén vagy végtelen nagy méretű doboz kell, vagy nagyon korlátos a „bármi”, vagy a változónak nem csak az értéke változik, hanem a doboza is. A változó típusának módosulását már az első bemutatott példa [24 p. 104] is kihasználja:

```
”
IDEI_ÉV = 2021
felhasználó_kora = input('Hány éves vagy? ')
print('Te most ', felhasználó_kora, ' éves vagy.')
felhasználó_kora = int(felhasználó_kora)
születési_év = IDEI_ÉV - felhasználó_kora
print('Ekkor születtél: ', születési_év, '.', sep='')
```

…Logikusnak tűnik egy `ilyen=input('És milyen', felhasználó_kora, 'évesnek lenni?')` megoldás… A hibaüzenet ismét int-ről és str-ről beszél, és ekkor már gyanítjuk, hogy az lehet a baj, hogy a felhasználó_kora a 4. sor óta egész szám,…”

Mennyiben segíti a változó fogalmának megértését az, hogy ugyanazt a változónevet használjuk különböző típusú adatokra? Másik adattípus, más memóriaméret. Hogyan segíti a programozás megértését, hogy az adattárolás és adatra hivatkozás pontos ismerete, a fogalom pontos megértése előtt „varázsolunk”?

„*A programozónak fontos a jó változónév, hogy ha holnapután előveszi a programját, még mindig el tudjon igazodni rajta. A változónevek választásával a program értelmezését segítjük.*" [24 p. 104] olvashatjuk a változónevek megadásának szabályai után. Vajon a beszédes névhez miért nem tartozik hozzá az is, hogy nem adom ugyanazt a nevet különböző típusú adatoknak?

A Python nyelv a dinamikus típusosságával támogatja, hogy kevesebbet kelljen gondolkodni a változónév kiválasztásán, de

- a hibás adattípus használata csak futtatáskor derül ki;
- a kód későbbi értelmezését megnehezíti a típus változékonysága;
- az adat, adattípus és változó fogalmak megértését felületessé teszi.

Ezért a dinamikus típusosság kihasználása a tanulás során káros, nem felel meg a tantervi céloknak, pusztán a Python nyelven beszélők – „mi így szeretjük" – közösségtudatának erősítését szolgálja.

Egy adattípust két dolog határoz meg: az adat tárolásának mérete és a rajta végezhető műveletek. Az adattípusnak (például karakternek) Pythonban – a dinamikusságot biztosító indirekció miatt – nincs értelmezhető mérete, ezért alkalmatlan az adattípus fogalmának tanítására.

5.1.3. Adatsorozatok

Tantervben 9-10. évfolyam: „*Az elemi adatok és sorozatok megkülönböztetése, kezelése és használata …Fogalmak: … sorozat …*" [17 p. 5]. 11. évfolyam: „*Az elemi és összetett adatok megkülönböztetése, kezelése és használata, Fogalmak: vektor*" [17 p. 16]

A sorozat és vektor két kifejezés, feltételezhetjük, hogy két különböző adatsorozatot jelent. A sorozat (sequence, range) összefügg a sorban léttel, egymásra következéssel, míg a vektor (array) a dimenzióval, aminek kétféle értelmezését használják: a vektor ahány dimenziós annyi adat, illetve annyi kiterjedtség. Számítógépes megvalósításra is többféle mód létezik. Kérdés, hogy melyeket célszerű tanítani, melyek azok, amelyeknek az ismerete hozzájárul a természeti és társadalmi környezet megértéséhez. A tankönyvben a Python nyelvben preferált lista lett az egyik adatsorozat: „*Ilyen például a Python lista adattípusa.*" Majd rögtön az egynemű adatok láncolásának bemutatása után egy specialitást is megemlít: „*Nem kell egyforma típusú adatokat tenni egy listába, azaz teljesen jó a csata = ['Isaszeg', 1849, 'április', 6, 'magyarok']*" [24 p. 120].

Láthatjuk azt is, hogy a lista mindenre jó csodaeszköz. Egyszerre valósítja meg a struktúrát és a láncolt listát. „*Akiben felmerül a kérdés, hogy „És hogyan tárolnám a tavaszi hadjárat összes csatáját egyetlen listában?", azt megnyugtatjuk, hogy lehet egy lista eleme egy másiknak. Aki ettől megijed, azt is megnyugtatjuk: ebben a könyvben nem foglalkozunk ilyen listákkal.*" [24 p. 120] Valóban, „ebben a könyvben" nem szerepel ilyen lista, de később annál inkább. Egy évvel később megjelenik a kétdimenziós adatszerkezet: „*A kétdimenziós adatszerkezetek használata nagyon gyakori – pont úgy, ahogy a táblázatoké a valóságban.*" Azonban a táblázat megvalósítása „listában listával" történik, így a fogalomzavar még egy elemmel bővül: a lista egyszerre láncolt lista, struktúraszerű és véges méretű tömbszerű (táblázat).

Valószínűleg eddig tart a „Pythonban könnyű megérteni a programozást" motiváló hatása. Az egyetemi programozásoktatás során a csak Python-t tanult hallgatókra jellemző, hogy nem értik, mi a tömb, mi a struktúra. Az ismereteik elégtelenségének felfedezése elkeseredést vagy lázadást vált ki.

Pythonban a lista „kézreálló", a használata könnyű. Bizonyos típusú feladatokra. A tankönyvből megtudjuk, hogy mire nem praktikus a lista: „*Kihívást jelentő feladatok, ahol nem segít rajtunk a* `count()`*, és mindenképp be kell járnunk a listát: a. Hány helyen előzi meg a hatos dobást ötös dobás? b. Hány helyen van egymás után két hatos?*" [24 p. 126]

A „kihívás", a feladattípus nehézségének az oka, hogy a listában az adatoknak nincs pozíciója, indexe. Nem mindegy, hogy az adatokat tároló „dobozok" egymáshoz illeszthetőek, mint az irattartó papusok vagy egy kartotéktároló szekrény fiókjaiként képzeljük el. Mi jelenti a kihívást: a másmilyen struktúra vagy a használt nyelvi eszközzel a struktúra implementálására?

Azok a feladatok jelentenek kihívást, ahol nem az adatsorozatban lévő adat (objektum) tulajdonságáról, állapotáról szól a kérdés, hanem a többi objektumhoz képest a helyéről. Megszemélyesítve: a „helyem a világban" megadható úgy is, hogy az én tulajdonságom a „gps-koordináta" értékem és úgy is, hogy a föld egy adott gps koordinátájú pontjára azt mondom, hogy „itt a helyem" Ez utóbbihoz szükséges egy térkép, ami a helyeket tárolja. Ennek digitalizált megfelelője a tömb (vektor, array, mátrix). A kétféle megközelítési módot a diákok számára is érthetően szemlélteti a pixelgrafikus és vektorgrafikus képmegjelenítés, mivel mindkét absztrakciós módszert tanulták már 10. évfolyamos korukra.

A láncolt lista és tömb összemosása kritikussá válik a kétdimenziós adatstruktúrák tárgyalása során: „*Mik azok a kétdimenziós adatszerkezetek? Az egyszerű listák egydimenziós adatszerkezetek – azaz csak hosszuk van, mint egy szakasznak a geometriában. Azonban a listákban elhelyezhetünk olyan elemeket is, amelyek saját maguk is listák. Így lesz az adatszerkezet kétdimenziós: van „szélessége" és „magassága."… A kétdimenziós adatszerkezetek használata nagyon gyakori – pont úgy, ahogy a táblázatoké a valóságban.*" [26 p. 81]

Itt a valós dimenzió fogalmát vetítjük a digitális adattárolásra. Ha a szakaszról van szó, azt digitálisan az egy dimenziós tömb (vektor) valósítja meg. A lista ebben az értelemben félegyenesnek, esetleg egyenesnek felel meg. Kétdimenzió esetén a téglalap alakú terület a tömb, míg a síknegyed a listában lista megfelelője. Aminek van kerete, – ez adja a szélességét és a magasságát, – az tömb jellegű. Aminek nincs határa, bővíthető, az a megfelelő dimenziók jellege alapján lehet akár listában lista is, de a tömb és lista keveréke is. A 10. évfolyamos diák számára ismert tömb a rögzített méretű táblázat. Például:

- A megjelenített kép egy 2D pixel tömb. Minden vektorgrafikusan tárolt képet végeredményben egy adott szélességű és magasságú pixel tömbben látunk.
- A táblázatkezelő munkalapja nagy, de véges méretű 2D tömb.
- A sakk, sudoku, keresztrejtvény… játékok.
- Ha tanult általános iskolában is digitális kultúrát, akkor ismeri a micro:bit 5×5-ös LED-mátrixát, de ha nem, akkor is számtalan jól érzékelhetően táblázatos elrendezésű LED-mátrixot lát a környezetében, például tömegközlekedési eszközök tájékoztató tábláit, reklám táblákat.

Ezekre az adatstruktúrákra könnyen tud kérdéseket és feladatokat megfogalmazni. Érti, hogy mit jelent a felette, az alatta, a jobbra, a balra, a „lóugrás", adott esetben a 90°-os elforgatás, tükrözés. A „listában lista" megvalósítása során a belső listák különbözőek lehetnek (Pythonban még az adattípusok is), így szemléletben a fel-le lépés és a jobbra-balra lépés közül csak az egyik értelmes. Míg tömb esetén a „sorfolytonos tárolás" logikus implementáció, a listában listára ez, és ezzel együtt a 2D táblázat értelmezése 1D vektorként is értelmetlen. A lista absztrakció a „dolgokat egymás után teszünk", sorozatképzési feladathoz köthető. A „listában lista" gyakorlati előfordulása például egy feladatlista, aminek minden tétele mellé odaírjuk, hogy milyen eszközök szükségesek hozzá. Egy ilyen struktúra megalkotása, konkrét eset elképzelése (fejben megalkotása) nehéz. Ezzel szemben a 2D tömb absztrakciója lényegében a létünk része: egy szem a világról 2D tömb képet érzékel. A szemidegek valószínűleg nem négyzetrácsban helyezkednek el, de nincs az információfeldolgozásban irány szerinti külső-belső rendelés. Másik oldalról közelítve: az egymás mellettiség és egymásfölöttiség is kapcsolat, míg a todo listában – ha egymás alá helyezem a „kis listákat", akkor ezek között nincs az egymás fölöttiek között kapcsolat.

A tömb listában listaként történő értelmezése ezek szerint egy programozói trükk, aminek megtanulását elvárjuk egy programozásban kezdőtől. Ezzel ellentmondunk a Python fejlesztők „hitvallásának" [32], azaz olyan elvek mentén tanítjuk a programozást, amit az éppen tanított nyelv fejlesztői károsnak tartanak:

(2.) „*Explicit is better than implicit.*" A kifejtett jobb, mint a hozzágondolt.

(3.) „*Simple is better than complex.*" Az egyszerű jobb, mint a bonyolult.

(5.) „*Flat is better than nested.*" A sima jobb, mint a beágyazott.

(7.) „*Readability counts.*" Az olvashatóság számít.

(8.) „*Special cases aren't special enough to break the rules.*" Nincs annyira speciális eset, hogy felülírja a szabályokat.

(17.) „*If the implementation is hard to explain, it's a bad idea.*" Rossz az ötlet, ha a megvalósítást nehéz elmagyarázni.

Bár a fenti kijelentések szakemberek számára irányadóak, oktatási és nevelési szempontból nem fogadható el, hogy a felhasználó és esetleg jövendő szakember képzésében más elveket közvetítsünk. Nem taníthatjuk diákjainknak, hogy a Föld lapos, ugyanígy, a programozáshoz hozzá tartozik a kétdimenziós intervallum digitális absztrakciója, mert amit a digitális világból a szemünkkel érzékelünk, az többnyire egy korlátos kétdimenziós tömbre van leképezve.

A fenti elveket figyelembe véve, a kétdimenziós tömb nem helyettesíthető „listában lista" struktúrával az oktatás során. Ugyanakkor, mivel alkalmazásával folyamatosan szembesül a diák, a 2D tömb megismerése a körülöttünk lévő világ értelmezéséhez szükséges. Ebből következően, vagy a Python nyelvben is meg kell valósítani a tömb adattípust, vagy más programozási nyelvet kell választani.

A programozók pontosan tudják, hogy a programozási nyelvek adott feladatok megoldására íródnak. A Python bizonyos feladatokra nagyon könnyen használható nyelv, amely elrejti a részleteket, ezzel segít a probléma megoldására koncentrálni. Pontosan ez, az általában hasznos a tulajdonsága teszi alkalmatlanná arra, hogy a diákok megismerjék a géphez egy absztrakciós szinttel közelebb álló adatstruktúrákat.

5.1.4. Rekord és osztály

A tantervben a rekord (vagy struct) adattípus nem szerepel. Adatbáziskezeléshez kapcsolódva találjuk meg a strukturált adattárolás ismeretét. [17 p. 12-13, 15-16; 21] Az osztály (class) ismerete sincs nevesítve, csupán az objektum fogalmán keresztül: fejlesztési feladat 9. évfolyamon: „*Az objektumorientált szemlélet megalapozása*" [17 p. 5] majd 11. évfolyamon szintén, valamint a fogalmak felsorolásánál: „*Objektumorientált szemlélet … eljárás, függvény, kódolás, objektumorientáltság, típusfeladatok, tesztelés…*" [17 p. 16].

A tanterv alapján nem szükséges a rekord, struktúra adatszerkezet ismerete, másrészt érteni kell az OOP alapvető megjelenési formáit, az objektum fogalmát, létrehozás, a tulajdonság és metódus fogalmakat. A fogalmak sorrendjéből következően vélhetően elvárt a típusfeladatok megoldása objektumok sorozatára is, de nem tananyag az öröklés. Ehhez képest vizsgáljuk a tankönyv releváns tartalmait.

A tankönyvekben több fejezetben szerepel az objektum kifejezés. Objektumokkal dolgozunk szövegszerkesztés és grafikák készítése során. A Python objektumorientált paradigmát követ, ezt hangsúlyozza a 9. évfolyamos tankönyvben a lista-objektum és a range-objektum kifejezés [2428 p. 123–124]. A range-objektum külön érdekessége, hogy ez lehetne egy vektor, de nem hivatkozunk rá változónévvel. Emellett a lista általában inkább adatsorozat. Úgy tűnik, adatsorozat-objektum. Ezzel a szemléletmóddal érünk az „Objektumok adatai kétdimenziós listában" fejezet példafeladatához: „*Egy osztály tanulóinak adatait tároljuk egy 2D-listában. Egy-egy kis lista egy-egy objektumról, azaz egy-egy diákról szól. A kis lista elemei, azaz az objektumainkról tárolt tulajdonságok: név, nem, kor és e-mail-cím*" [26 p. 84]

Egyrészt, a lista egy programozásban használt, bejárható objektum, aminek a tulajdonságairól egy szó sem esik. A 2D-lista vélhetően „objektumban objektum", a „kis lista" is objektum, ami egy objektumról, a diákról szól. A lista-objektum elemei a diák-objektum tulajdonságai – vagy a cím alapján az objektum adatai.

Innen egyenes út vezetne az osztály fogalmához, a példányosításhoz, azonban csak egy évvel később, lényegében kiegészítő ismeretként kerül elő a téma: „*De mi az az objektum?*" majd a meghatározás: „*Az objektumok egy-egy létező, azaz egy-egy személy, tárgy, fogalom számítógépes programban létrehozott reprezentációi, megvalósulásai. A létező lehet bármi: diák…*" [28 p. 202]

Egészen a 11. évfolyamos tananyag végéig az objektum a programozáson kívüli „létező bármi", értelmű, csak ekkor vált át arra, hogy az objektum egy programbéli reprezentáció. Tekintettel arra, hogy a tantervben nem hadászati témakörben, hanem az adatok és programozási nyelv használata témakörben szerepel az objektum szó, az objektumorientált szemlélet nem a célszemély tárgyiasításáról, hanem az objektum tulajdonságainak számítógépes reprezentációjáról kellene, hogy szóljon.

A Python nyelv elrejti az adatot a felhasználó elől, a tankönyvi feldolgozás ezt továbbviszi, amennyire lehet: az objektum belső struktúráját is elrejti. Ahelyett, hogy az egyes tulajdonságokhoz megnevezéseket (és adattípust) társítana a változó-érték fogalompár felhasználásával, kulcs–érték párokkal, objektum–objektum modellt épít csak azért, hogy könnyebben olvasható legyen a kód: „*Előrebocsátjuk, hogy a szótár adattípus nem szükséges abban az értelemben, hogy minden, amire a szótár típus használható, megoldható listák használatával is, de a szótárak használata olyan sok esetben teszi kényelmesebbé a munkánkat, hogy kifejezetten érdemes megismerkednünk vele.*" [26 p. 85]

A szótárra természetes példa egy angol-magyar szópár lista, de pont ez a példa alkalmatlan a szótár specifikálására, hiszen mindkét nyelven számos szinonima teszi pontossá egy fogalom megfelelő megnevezését e két nyelven, ami a kulcs ismétlődését jelentené. De az alfejezet címéből tudjuk, hogy vagy a fogalom, vagy a magyar és angol szó egy objektum: „*Objektumok szótárban*" [26 p. 86].

Ezt követi az „*Osztálytagok a szótárban*" [26 p. 87] fejezet. A címnek többféle értelmezése is lehetséges, de a folytatás nem tisztázza, hanem érthetetlenné teszi a fogalmakat.

„*A szótárakat nagyon gyakran használjuk arra, hogy egy-egy objektum tulajdonságait tároljuk egy szótárban. Az előző lecke utolsó feladatában, ahol az osztálytagok adatait – tulajdonságait – tároltuk, a tárolást listával oldottuk meg.*" [26 p. 87]

Ez alapján azt várnánk, hogy van egy feladat, amiben egy osztály tagjait, azaz a diákokat tároltuk listában, most diákok szótára lesz. A végére az lesz, de előbb diák tulajdonságaiból képez szótárat. Minden diáknak lesz 'nev', 'nem' és 'kor' kulcsú szótárbejegyzése a megfelelő értékekkel. Ebben a megközelítésben a 'név', 'nem' és 'kor' objektum, hiszen a kulcs és az érték is objektum. Ugyanakkor a class adattag fogalmához hasonlóan adja meg a diák tulajdonságait:

```
{'név': 'Noémi', 'nem': 'l', 'kor': 15}
```

Minden diákra elkészítve a szótárt, az osztály a diákok, azaz az osztálytagok listája, adatszerkezete szótárbejegyzésekből épített szótárak listája.

Úgy tűnik a 10. évfolyam (heti 1 órája) a bátorságról szól. Az egy évvel korábban ijesztő listában lista mellé jön a szótár, de nem eredeti értelmében, hanem struktúra (vagy class) helyett, ráadásul rögtön listában alkalmazva. De ez sem elég: „*Legyünk bátrabbak…*" és már jön is a szótárban szótár.

```
osztály = {'Noémi': {'nem': 'l', 'kor': 15},
           'Dezső': {'nem': 'f', 'kor': 17},…
```

Miért kell a szótár? Miért kell összekeverni az objektum, az objektum számítógépes reprezentációja, a tulajdonság, változó, adat, kulcs, érték fogalmakat? „*Ezért küzdöttünk!*" [26 p. 88]

```
print(osztály['Dezső']['kor'])
```

Sok bátorság, fogalmak zagyvasága árán eljutunk egy jól olvasható leíráshoz. Az eszköz – a szótár – nem tananyag, csak egy újabb Python objektum. Az objektum mibenlétéről nem tudunk meg semmit. Valójában érteni sem kell. Ezen a ponton a tanár és a diák is vagy a nyelvi eszköz megtanulását választja, vagy marad a listában lista.

Nem csak a programozás alapjainak megértését nehezíti (vagy gátolja) meg a szótár bevezetése, de a tantárgy más témáival való kapcsolódás szempontjából is káros a struktúra (vagy class) ilyen formán történő megkerülése. 10. évfolyamon – a tankönyvben a programozás után, de valójában valamikor a tanévben – az adatbázis-kezelés alapjainak megismerése is tananyag.

„*Képzeljünk el egy osztályt, a 10.g-t! Ott van a barna hajú Molnár Ricsi, aki 160 cm magas és 59 kg. Padszomszédja… Ezeket … nevezzük adatoknak. Egy-egy tanuló adatainak összességét… rekordnak hívjuk, a rekordok egyes adatait mezőnek, az adatok megnevezését pedig mezőnévnek.*" [26 p. 90].

A 10. évfolyamos tankönyv (84–88.) öt oldalán egy bevallottan kihagyható levezetéssel eljutunk odáig, hogy jól olvasható egy osztály tagjainak néhány tulajdonságára történő hivatkozás. Majd kétoldal gyakorlófeladatot követően, ugyanarra a feladatra (diákok tulajdonságai) egy, az eddig bemutatott gondolatmenethez, fogalom-rendszerhez és leírási módhoz semmilyen módon nem kapcsolódó fogalmak, definíciók sorakoznak. A digitális világ kettéhasadt: van a táblázatkezelés és adatbáziskezelés az adatokkal, táblákkal, és van a Python nyelven programozás, listában listával, szótárban szótárral. A kettő között nem látszik fogalmi kapcsolat. Ezzel a Python nyelven programozás és fogalomhálója elszigetelődik a többi témakörtől, ezért a tanultak a dolgozat után elfelejthetőek. A programozás ismeretekből annyi marad, hogy Pythonban írt kódot a diák.

5.1.5 OOP és osztály

A tanterv a 11. évfolyamra előírja a továbblépést a grafikus felületen futó programok készítése irányába, javasolt tevékenységként elvárja bizonyos objektumok létrehozását és tulajdonságainak módosítását:

„– *Az alapvető vezérlők használata: címke, nyomógomb, szövegmező, jelölőnégyzet, rádiógomb a felhasználói felület programozásában alkalmazói jellegű feladatok során (pl. megrendelés beviteli felülete)*

– *Alapvető grafikus vezérlőelemek létrehozása és használata a felhasználó felület programozásában*"

A „saját adatszerkezet", azaz a class definiálásához ezúttal is egy osztály (mostantól csoport) egyes adatai jelentik a példát, a példa megoldását először listában szótár struktúrával láthatjuk, de választható lenne a listában listák is. [28 p. 197–202.] Ezt követi az osztállyal kapcsolatos fogalmak tisztázása. Továbbra is a Python nyelv specialitásai elsőbbséget élveznek az általánosan használt szakmai terminológiával szemben: „*Úgy mondjuk, hogy ilyenkor történik meg az objektum inicializálása (magyarra előkészítésnek fordíthatnánk, de nem szokás fordítani). Néhány más programozási nyelvben az osztályok, objektumok hasonló szerepű függvényét konstruktornak nevezik, és ez az elnevezés olykor átszivárog a Python világába is.*" [28 p. 203]. – Ez az egyetlen említése a konstruktor kifejezésnek.

A konstruktorhoz hasonló sorsra jut a tulajdonság és adattag fogalom is: „*Az objektumokat ebben a leckében még csak arra használjuk, hogy a tárolt létezők bennünket érdeklő tulajdonságait tároljuk bennük. … Az ilyen tulajdonságokat ebben a könyvben jellemzőnek nevezzük (szokás még többek között a mező, az adattag és a változó elnevezést használni).*" A „jellemző" valószínűleg az „adattag" megnevezést helyettesíti. Nem esik szó arról, hogy az OOP egyik alapelve az egységbe zárás, de felhívja a figyelmet arra, hogy ha az objektumnak nincs saját függvénye, akkor a szótár és lista elég.

Bár a tanterv csak a grafikus képernyőn tipikusan használt objektumok programozását várja el, a tankönyv alapján ehhez gyakorlatot kell szerezni tagfüggvények írásában, meg kell ismerkedni a kulcsszó-paraméteres függvényekkel és a modulkészítéssel is. Mindezt lényegében egy-egy mintán bemutatva, majd szintén példaként, azok, akik még értik, hogy miről van szó és eléggé érdeklődőek, lemásolhatják két grafikus alkalmazás kódját. A tantervben szereplő javasolt tevékenységek így legfeljebb kiegészítésként, a legérdeklődőbb, kitartó diákok számára lesz megvalósítható. [28 p. 219–230.] A grafikus felületen futó alkalmazások programozását tipikusan kész sablonokból, a programkód módosításával szokták tanítani. Itt ehelyett egy ennél jóval nehezebb, megtanulásra alkalmatlan módszert látunk, két grafikus felület elkészítését. A diák nem arra lát példát, hogy hogyan tudja más környezetben alkalmazni eddigi tudását, hanem arra, hogy hogyan lehet elkészíteni egy másik alkalmazási felületet.

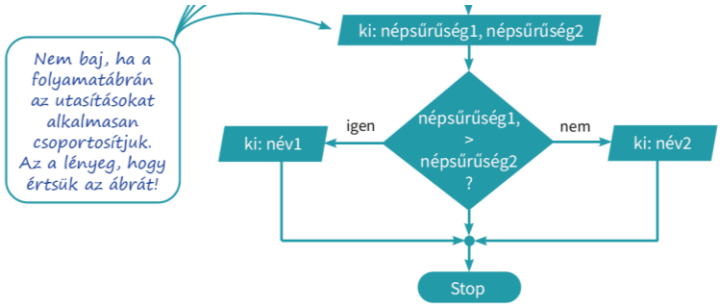
5.2- Vezérlési szerkezetek

Ahogy az adattípusok, úgy a vezérlés alapelemei és a típusalgoritmusok tárgyalása is a Python nyelvhez igazítva jelenik meg, előtérbe helyezve a nyelvi specialitásokat a tantervben elvárt ismeretekhez képest.

5.2.1. Elágazás

A Python nyelv specialitásai elsődlegesnek tekintése más programozási nyelvekhez, technikákhoz képest mutatkozik meg az elágazásban használható 'elif' összevonás kapcsán. Az egybeírt módot a mondatszerű leírás (pszeudokód) nyelvére is visszavetíti: „*Elsőként az algoritmus mondatszerű leírását módosítjuk. Az új, „különbenha" ágat megvalósító Python-utasítás az elif.*" [24 p. 110].

A kettőnél több ágú elágazás egybevonása folyamatábrán is megjelenik. Az egyik feladatban a folyamatábra alapján kell elkészíteni a kódot. Ennek részlete az alábbi ábra: [26 p. 47]



Az elágazás – a mintamegoldás alapján – nem hibás, hanem azt akarja szuggerálni, hogy legyen kiírás az egyenlőség esetére is.

5.2.2. Ciklus

A Python nyelv kétféle ciklust ismer, az elöltesztelős feltételes (while) ciklust és a bejáró (foreach) ciklust. A típusalgoritmusok tanításához kétféle ciklust használnak, az elöltesztelős feltételes (while) ciklust, és a számlálós (for) ciklust. A spagetti kód strukturált programmá „kibogozása" során célszerű volt a hátul tesztelő ciklust (do-while) is bevezetni, a belépéskor tesztelő és a tesztelés nélkül, automatikusan léptető ismétlés mellé. Kérdés, hogy a rendelkezésre álló nyelvi eszközök alkalmasak-e az elvárt kompetenciák elsajátítására. A tanterv alapján a ciklusok ismeretére épülnek a típusalgoritmusok, illetve a felhasználó által megadott adatok ellenőrzése is.

5.2.2.1 Léptető ciklusok

A bejáró és számlálós ciklus használata során az adatsorozat elemének elérésében van jelentős eltérés: a bejáró ciklus az adatsorozat objektum elemein iterál, míg a számlálós ciklus egy, az adatsorozattól akár független változót, a ciklusszámlálót módosítja. A típusalgoritmusokban ezt a ciklusszámlálót használjuk a tömb elemeinek elérésére mutatóként. A Python lista (illetve a range és később a set) adatsorozat objektum, amely egyes adattagjainak az elérése objektumon belül történik, ehhez igazodik a bejáró ciklus. A számlálós ciklust a range objektum helyettesíti. Azonban e kettő együtt sem biztosítja az adatsorozat elemeinek egyszerű, természetes elérését kívülről „rámutatással", hivatkozással. Ezért lesz kihívás azoknak a feladatoknak a megoldása, amelyekben kívülről több sorozatelemre kellene hivatkozni: egy részsorozat első és utolsó eleme, szomszédos elem vagy többdimenzióban a transzponált.

A tankönyv minden évben mutat olyan feladatot, ami az indexek elérésének körülményessége miatt kihívást jelent.

- 9. évfolyamon egymás után két 6-os dobás megszámlálása nehéz feladat [24 p. 126],
- 10. évfolyamon a visszafelé számlálás, utolsó jó érték keresése [26 p. 78]: „*A bejárandó indexeket előállító range() függvényt így paraméterezzük: range(len(bevételek)-1, -1, -1). A len()…ezért az első –1 … … hogy … A második –1 … azaz … A harmadik –1 … Ez a korrekt megoldás. Sok más nyelven ezt egy visszafelé haladó számlálós ciklussal valósítanánk meg – de Pythonban nincs ilyen.*".

- 11. évfolyamon a típusalgoritmusokkal kapcsolatban olvashatjuk: „*Ha nem az ott szereplő elemmel, hanem a belőle kiszámítható, meghatározható értékkel, vagy az elemek közül csak azokkal, amelyek megfelelnek valamilyen feltételnek, tehát nem mindegyikkel kell dolgoznunk, az egyszerűbb, függvényes formák mit sem érnek.*" [28 p. 160] Az egyszerűbb, függvényes formákban tisztán lehet használni a bejáró ciklust, ezért is készítették el a függvényt rá.

Vajon helyes választás a Python a típusalgoritmusok tanításához, ha az alapesetekre beépített függvény használatát ajánlja az algoritmus helyett, a módosított algoritmus kódolása viszont nehezebb, mint „sok más nyelvben"?

5.2.2.2. Feltételes ciklusok

Az elöltesztelős feltételes ciklus a Pythonban és a pszeudokódban azonos módon használható. Talán érdemes megemlíteni, hogy a feltétel mindkét esetben a ciklusba belépésre vonatkozik, míg vannak nyelvek, ahol az ismétlendő átugrására szól (például Scratch).

Hátultesztelős ciklus Pythonban nincs. Ciklus belsejében kialakuló feltételek ellenőrzésére a „biztosan belépjen" kezdőállapotot használja: „*A Python az ilyen esetekre tartogatja a nagybetűs semmit, azaz a None értéket. Ha a harmadik sort így írjuk át: kacat = None"* [24 p. 122]. Ez a megoldás filozófiájában más, mint a pszeudokódban és sok más nyelvben meglévő do-while, hátul tesztelő ciklus. A gondolkodásmód – és ebből következően a megoldási mód – nehezen általánosítható, mert feltételezi a minden más adattól különböző `None` adatot. Mivel Python nyelven nem gondoljuk egy változónév „dobozról", hogy fizikai megfelelője, azaz lefoglalt, biteket tartalmazó memóriaterülete van, ezért nem gond, hogy üres. Más nyelveken azonban problémát jelent az „éppen nem szükséges speciális érték" használata és mindenképpen más gondolkodást igényel ez a megoldás és a ciklusmag végén történő tesztelés. Emiatt a hátultesztelős ciklus működésének megértéséhez és használatának begyakorlásához nem alkalmas ez a kerülő megoldás.

A Python nyelv lehetőséget ad „belül tesztelő" ciklus szervezésre. „*A Python nyelv úgy segíti elő a break utasítás használatát, azaz a ciklusokból való idő előtti kilépést, hogy a ciklusoknak létezik else-záradékuk – ez meglehetősen ritka dolog a programozási nyelvek körében. Az else-záradék csak akkor fut le, ha a ciklus végigfutott, azaz nem léptünk ki belőle break-kel.*" A megoldás lényege egy, a szigorúan strukturált programozásban nem értelmezett utasítás: a vezérlés megszakítása. Ugyanakkor, ha az `if` és ezen belül a `break` közvetlenül a ciklus vége előtt van, akkor a működése megfeleltethető hátul tesztelő ciklusnak. Kérdéses, hogy így egyszerűbb-e, ahogy az is kérdéses, hogy a strukturált programozás céljának, a biztosan helyes program tervezésének megfelel-e, hogy a programozó a `break` után bármit beírhat.

Végül, a ciklus lefutását adatbevitel ellenőrzésekor kívülről is lehet tesztelni megszakítás kezelővel, a try-catch utasítás párral. Ekkor nem `break` utasítással szakad meg a ciklus, hanem kivétellel (exception), de a hatás lényegében hasonló.

Míg a strukturált programozás elől, illetve hátul tesztelő ciklusa a stabil programvezérlést segíti, a „belső" és „külső" tesztelés a programozási nyelv specialitásait használja ki. Bármelyik gondolkodásmód lehet helyes. Kérdés, hogy könnyen megtanítható-e egy speciális megoldásra alapozva a biztonságos megoldás, vagy előtérbe kellene helyezni a biztonságos megoldást, és ebből specializálni a nyelvben megtalálható megoldási lehetőségeket.

5.3. Típusalgoritmusok, programozási tételek

Az Algoritmizálás és programozási nyelv használata témakör algoritmizálás részében központi szerepe van a típusalgoritmusoknak. A típusalgoritmusok egyes feladattípusokra kidolgozott gondolkodási sémák, amelyeket programozási nyelven kódolva a számítógép elvégzi a megadott feladatot. A digitális kultúra Algoritmizálás és (formális) programozási nyelvek használata témakörének tárgyalásában az egyik legnagyobb hiba, hogy felajánlja a típusalgoritmusok (gondolkodási sémák) megtanulásának elkerülését. Minden típusalgoritmus magyarázata és a mintafeladatok megoldása is úgy kezdődik, hogy

kell-e vagy nem kell a típusalgoritmust használni. Ezzel azt is sugallja, hogy a tantárgy teljesítéséhez nem szükséges az algoritmizálási készség. Hasonlattal élve: mintha a főző tanfolyamon azt tanítanák, hogy főzésnek nevezzük a konzerv vásárlását és megmelegítését: „*Persze van egyszerűbb megoldás is*" [26 p. 67-]. A Python nyelvet a könnyen tanulhatóság miatt választják programozás tanulására. Pontosabban azért, mert könnyen lehet vele programot készíteni. Ezért divatos, ez adja a motiváció alapját. Amikor a típusalgoritmusok tanulása a feladat, akkor nem a programkészítés a cél, hanem a számítógép programozásához szükséges gondolkodásmód fejlesztése. Ha nem várjuk el a tanulótól, hogy megértse és alkalmazza a típusalgoritmusokat, hanem a programozási nyelv alkalmazkodik hozzá, az olyan, mintha a csecsemővel gügyögve beszélnénk. Ennek fejlesztő hatása maximum nulla, sőt, a könnyen szerzett sikerélmény miatt a gondolkodás megerőltető lesz, ezért a fejlődés ellen hat, így negatív.

Egy példa erre a csere algoritmusa, amelyet algoritmizálás és blokk-programozás során gyakorolhatott a diák. A tankönyv az algoritmusra hivatkozás nélkül, a rendezés kapcsán a Python-os kettős értékadást mutatja [28 p. 190]:

```
lista[egyik], lista[másik] = lista[másik], lista[egyik]
```

Ez a formula azt sugallja, hogy a cseréléshez nem szükséges átmeneti tároló, lehet „levegőben, feldobva" cserélni adatokat. Fel sem merül a gondolat, hogy vagy az adatokat kell áthelyezni, vagy az adatokra történő hivatkozásokat kell megkeresni és módosítani.

5.3.1. Szemléletnek megfelelő típusalgoritmusok

A megszámlálás, másolás és kiválogatás algoritmusa könnyen megérthető, a típusalgoritmusok tanulása előtt is több példában szerepel. Ezekre a típusalgoritmusokra mondhatjuk azt, hogy úgy kell megírni az algoritmust, ahogy az ember is csinálná. A megszámlálás algoritmusa helyett használható `count()`, illetve lista hosszára a `len()`. Használatukat már egy évvel a típusalgoritmusok tanulása előtt javasolja a tankönyv [24 p. 96]. A másolás és kiválogatás algoritmusok haladó algoritmusok, mert sorozatot adnak eredményül. A természetes lépésenkénti megoldás helyett a map vagy a lista értelmezés (list comprehension) is használható. Valójában nem az algoritmus nehéz, hanem a helyette használható függvények paraméterezése. A map első paramétere egy függvénypointer, ami legfeljebb formálisan érthető, mivel az sem teljesen tisztázott, hogy az int egy objektum típusa vagy függvény, ami után esetleg véletlenül lemaradtak a zárójelek [28 p. 174]. Ugyanitt, a listaértelmezés még ennél is összetettebb nyelvi varázslat, lényegében lambda kifejezésként adja meg a másolást és kigyűjtést. [28 p. 174].

5.3.2. Összegzés típusalgoritmus

Az összegzés algoritmusának függvénnyel történő helyettesítése újabb oktatásmódszertani problémát vet fel: nem igaz, hogy az ember úgy végzi el a műveletet, ahogy az algoritmus. Az ember iskolában tanulja a számok összeadását és több szám esetén egymás alá írva, helyiértékenként végzi el a műveletet, nem pedig kumulációval. Ezért azok a diákok is, akik táblázatkezelésben, adatbáziskezelésben már esetleg tanulták a `sum()` függvényt, megakadnak azon, hogy hogyan algoritmizálják a helyiértékes összeadást. Külön, megtanulandó és gyakorlandó a számítógépes összegzés algoritmusa.

5.3.3. Eldöntés, keresés, kiválasztás

Az eldöntést – és ebből következően a keresést és kiválasztást is – általában teljesen más módszerrel végzi az ember egy géphez képest. A lineáris és bináris keresés tanulás eredménye. Egy átlagos eldöntendő kérdésre az ember ránézésre válaszol. Van-e szőke ember a buszon? – nem kezdi el sorban megnézni, mert „szemmel látható" foltot vesz észre (vagy nem vesz észre). Az ember már kisgyermek kortól gyakorolja a képek összehasonlítását, a növények, tárgyak közötti különbségek felfedezését. A

különbségeket, hibákat, adott részleteket szkenneléssel keresi, nem képpontonkénti összehasonlítással. Amikor listában a „van-e" kérdésre az `in` operátorral adunk választ, akkor nem gondolunk bele, hogy a gép másképp működik, mint az emberi agy [26 p. 59]:

```
if vizsgált in lista: print('Benne van.') else: print('Nincs benne.')
```

Az 'in' lebontása algoritmusra a legnehezebb a típusalgoritmusok tanulása során, mert a vizsgált adatokra „ránézésre" megállapítható az eredmény. Ráadásul általános, hogy az adott programozási nyelven vannak olyan logikai vizsgálatok, döntések, amelyek alacsonyabb szintű implementációban már eldöntés tétellel oldhatók csak meg. Például két szám összehasonlítása processzor szintjén az első eltérő bit megkeresést jelenti, az egyenlőség annak eldöntése, hogy minden bitjükben megegyeznek-e.

Azzal, hogy Pythonban az adatsorozat lista-objektum, erősítjük a sorozat vizsgálata helyett az objektum tulajdonság megközelítést, amihez a kívülről meghívott `in` operátort használjuk. Egy tömb sokkal nyitottabb, az egyes elemeit közvetlenül elérjük (nincs elrejtve a belsőjébe a szerkezete). Ezért sokkal jellemzőbb, hogy lineáris kereséssel adunk választ a kérdésre.

A kétféle megközelítés – mi a döntés és mi az, amihez az eldöntés típusalgoritmust kell alkalmazni – az elmúlt 3 évben az ELTE-s programozás csoportjaimban a hallgatók több, mint felének okozott problémát. Például a feltételes maximum-kiválasztást eldöntés tétel és maximum-kiválasztás tétel kombinációjának specifikálták.

A keresés és kiválasztás típusalgoritmusok operátorral és függvénnyel történő helyettesítése egy harmadik szempont szerint is hibás, ez pedig a hatékonyság. A tankönyvi példák közül tipikusan az indexre rákérdező – melyik elem rendelkezik az adott tulajdonsággal – feladatok megoldása rossz hatékonyságú. Például [26 p. 71]:

„*Természetesen ezúttal is létezik egyszerűbb megoldás – és ez is a két előző „egyszerűbb megoldás" összeépítése.*"

```
if 5 in bevételek:
print('Az ötpiculás fuvar sorszáma:', bevételek.index(5) + 1)
else: print('Nincs ötpiculás fuvar.')
```

A megoldásban az `in` operátor egy eldöntés típusalgoritmust rejt el, majd az `index()` a kiválasztás típusalgoritmusát használja. A kód futtatási ideje általában a vizsgálandó adatok számától négyzetesen függ. Ez az a kombináció, ami miatt például a már ismert INDEX(…HOL.VAN()) függvénykombináció tömeges használata lelassítja a táblázatkezelő újraszámítási eljárását. Így az egyszerűbb megoldás nem csak gondolkodásban jelent eltunyulást, lustaságot, hanem a program futását is lassítja.

Összefoglalva: az `in` operátor és `index()` függvény használata az eldöntés, keresés és kiválasztás típusalgoritmusok helyett

- nem teszi lehetővé az adatsorozat elemeinek közvetlen elérését;
- nem fejleszti az algoritmikus gondolkodást;
- nem teszi lehetővé a program idő (és ezzel erőforrás) hatékony felhasználását.

Emiatt ezeknek az algoritmusoknak a függvényekkel történő helyettesítése kiemelten káros.

5.3.4. Szélsőérték-kiválasztás

Az eldöntéshez hasonló problémákkal találkozhatunk a szélsőérték kiválasztása során is. Amikor egy alkalmazásban a `max()` függvényt használjuk, nem gondolunk bele a függvény működésének algoritmusába. Amikor mi magunk keressük a maximumot, akkor viszont nem hasonlítgatjuk össze egyenként az adathalmaz elemeit. Egy hatalmas tömegből úgy választunk maximumot, hogy kiválasztunk néhány, a környezeténél magasabb embert és ezeket hasonlítjuk össze, jó eséllyel nem sorban, hanem páronkénti kiejtéssel. A maximum kiválasztás algoritmusának megírása ezért más, nem a természetes gondolkodásmódot igényli.

Ennél a típusalgoritmusnál is nehézséget okoz, hogy a lista-objektum adatai nehezen érhetők el közvetlenül: „*Ötlet: vezessünk be egy változót, aminek az értéke a feladatban elképzelhető legkisebb eredmény*…" a következő feladatban: „…*a farkas legkisebb libáját tároló változó kezdőértékéül a 100-at választottuk, bátor döntés*…" A típusalgoritmust követően: „*Kihívást jelentő feladat: Módosítsuk úgy az előző programunkat, hogy biztosan helyesen adja meg a farkasnak jutó legkisebb liba tömegét!*" [26 p. 73-74].

A típusalgoritmus végiggondolásába Python nyelven nem fér bele, hogy az első adat legyen a kezdőérték. A további gyakorlófeladatok során a kezdőérték jellemzően 0. Ha nem az értéket kell visszaadni, akkor az erre használt változó `None` értékű.

A maximum-kiválasztás algoritmusa összetettebb feladatok esetén kifejtve is megtalálható, de mindig van mód az „egyszerűbb, függvényes" megoldásra. Például [28 p. 162]:

```
legtöbb = max(heti_hiányzások)
print('A(z) ', heti_hiányzások.index(legtöbb)+1, '. héten volt
          a legtöbb hiányzás.', sep='')
```

A megoldás itt nem a kezdőérték választása miatt pongyola, hanem azért, mert a függvényekkel nem egy, hanem két kiválasztási algoritmus szükséges, ami miatt nő a futási idő. A feladat függvényes megoldása táblázatkezelőben az INDEX(…HOL.VAN(MAX(),…)) függvénykompozíció, ott ez megfelelő megoldás, de tudni kell, hogy az újraszámítások miatti tömeges használata lelassíthatja a gép működését. Ezért érdemes megtanulni egy hatékonyabb megoldást, ami nem a függvények összetett alkalmazása, hanem közvetlen, hatékonyabb megoldást ad.

Az eldöntés típusalgoritmusának alkalmazása bonyolult keresési feltétel esetén nehezen kerülhető el, ezzel szemben a szélsőérték-kiválasztásának típusalgoritmusa – esetleg előzetes kiválogatással kombinálva – minden esetben helyettesíthető függvénnyel, ezzel a szélsőérték-kiválasztás algoritmusát teljesen elfedi.

A táblázatkezelés során használt függvények programozási nyelvben történő alkalmazása nem fejleszti a gondolkodási képességet, legfeljebb ismétli a már tanultakat. Mivel nem visz tovább annál, mint amit táblázatkezelőben 4 kattintással megkapunk, nem ad értelmet a programozási nyelven való megvalósításnak. Emiatt nem csak a gondolkodás fejlesztése marad el, hanem a motiváció is elkopik.

5.3.5. Összetett algoritmusok és függvényeik

Az összetett algoritmusok az egyszerű típusalgoritmusra épülnek. Kivéve, ha függvényeket alkalmazunk az egyszerű típusalgoritmus helyett, mert akkor az összeépítés nehézséget fog okozni. Ezért az összetett algoritmusok megértése és felépítése helyett a tankönyvben előtérbe kerülnek az ezeket megvalósító függvények. Ebből adódóan a speciális igények megvalósítása nem a tanultak módosított alkalmazása (a tudás fejlesztése), sokkal inkább internetes keresés és formális behelyettesítés.

„*Megjegyezzük, hogy a Python sort() és sorted() függvénye egyébiránt remek dolgokat tud, például a kedvencek kétdimenziós listát a kis listák utolsó eleme – a kor – alapján akár fordított irányba is képes rendezni. A szintaxis nem egyszerű, de az érdeklődők számára megmutatjuk:* [28 p. 192]

```
kedvencek.sort(key=lambda x: x[-1], reverse=True)
```

Vajon mit gondol az érdeklődő diák a „key=lambda…" kifejezésről? Volt diákom elbeszélése alapján: „sokat tanultam, de nem jutottam semeddig". Egy, adatbáziskezelésben néhány kattintással vagy egyszerű SQL utasítással megoldható problémához minden eddigi ismeret Python nyelven kevés. Ha egy rendezés megfordításához ennyi plusz ismeret kell, akkor mi kell egy játékprogram elkészítéséhez?

6. Összegzés

A Python világszerte elismert, az informatika több területén jól használható programozási nyelv. Objektumorientált, a memóriát dinamikusan kezeli. A gyakori feladatokra kész, a humán gondolkodáshoz alkalmazkodó megoldásokat tartalmaz, ezért Python nyelvet használva a programozásban járatlanoknak is sikerélményt ad az első programok elkészítése.

A Python első programozási nyelvként azoknak ajánlott, akiknek a gépelés nem jelent problémát, emiatt a felnőttképzésben, az egyetemi tanulmányok kezdetén jellemző a Python nyelvű programozásoktatás.

A közoktatásban 2020-ban bevezetett tanterv szerint a programozásoktatás az általános iskola alsó tagozatán kezdődik. Ekkor a Python még nem használható, ugyanakkor a különböző blokk alapú programozási környezetekben a programozás – kód begépelésén kívüli – elemei játékos formában tanulhatók. Egyes blokk programozási környezetekben kódnézet is rendelkezésre áll, ez alkalmat ad például a Python nyelvre való átállásra, a Python nyelvvel való ismerkedésre. Erre a nyelvváltásra még nincsenek gyakorlati tapasztalatok.

Hosszú távon a középiskolai programozásoktatás építhet az általános iskolás években tanult elméleti és blokk-programozás során tanult ismeretekre, ezért a középiskolás programozásból elvileg nem kezdő. Azonban jelenleg, a tanterv több ponton történt bevezetése miatt, vannak olyan évfolyamok, amelyeknél még nem számíthatunk az általános iskolában tanult programozási ismeretekre, ezért reális lehet Python nyelven tanítani a középiskolás tananyagot.

Összehasonlítva a tanfolyami, egyetemi előkészítő és közoktatási tantervet, azt tapasztaljuk, hogy a közoktatásban az általános iskola végére elérendő cél – a Python nyelv használatától eltekintve – megegyezik az ICDL és más, felnőtteknek szánt bevezető képzések céljával.

A középiskolai programozás tananyag ezen túllép: az adat absztrakció a virtuálisan tapasztalttól fizikai megvalósításhoz közelít, az adatok kezelésére a strukturált programozás elveit figyelembe vevő tervezés és kódolás jellemző. Ezzel betekintést nyújt a programozás (program tervezés, adatstruktúra tervezés, objektum tervezés) szakmákba. A tananyag érzékelhetően túllép a belépő szinten, a programkészítés felszíne alá nézve elméleti és gyakorlati ismeretek elmélyítésére és rendszerezésére törekszik.

A középiskolai Digitális kultúra tantárgy Algoritmizálás és (formális) programozási nyelv használata témakör 9–11. évfolyamos fejezetei a tananyagot Python nyelv használatával dolgozza fel. Pontosabban, a Python nyelvet és programozói gondolkodásmódot tanítja a kitűzött tananyag mentén. Talán a terjedelmi korlátoknak köszönhető, de jól látható, hogy a nyelv nem a tantervi cél megvalósításához használt eszköz, hanem fordítva, a cél a Python nyelv és gondolkodásmód használata, amihez a tantervhez kapcsolódó feladattípusok szolgálnak eszközül. A cél-eszköz kapcsolat módosításának indoka, hogy a Python nyelv használata motiválja a diákokat a programozásra.

A tananyag elemzése során azt vizsgáltuk, hogy a Python nyelv oktatása során megtaníthatók-e a tantervben előírt fogalmak, ismeretek, kompetenciák, illetve, ha megtaníthatók, akkor mi a motiváló tényező. Másképp: mennyire jó, mennyire használható a Python nyelv a tananyag elsajátítására. Az elemzés egyértelműen kimutatta, hogy a tantervi követelményekhez a Python nyelv nem tartalmazza az eszközöket. Ennek oka leginkább az, hogy a Python nyelvben a memóriában tárolt adathoz csak pointeren keresztül, azaz indirekt módon lehet hozzáférni:

- Python nyelven nincs karakter típus.
- Az elemi adattípusokat csak műveleteik jellemzik, nincs méretük, memóriában lefoglalt helyük.
- A változónév dinamikusan van az adathoz társítva, így az adat a változó deklarálásával nem absztrahálható.

- Nincs tömb (vektor) adatstruktúra. Mivel az adat sem érhető el közvetlenül, az adatsorozatra (intervallumra) optimalizált típusalgoritmusok adattípusa hiányzik. Ebből következően nincs egyszerű kétdimenziós „táblázat" adatszerkezet se, amiben az adatot sor és oszlop indexen keresztül lehet elérni.
- Nincs a tantervben elvárt típusalgoritmusokhoz számlálós ciklus a strukturált programozás ciklustípusaiból csak az elöl tesztelő ciklus található meg az eredetivel azonos működéssel.

A tantervi tananyag elsajátításának további akadálya, hogy a Python nyelvet nem arra optimalizálták, hogy a programozó részletesen leírja, hogyan működjön a program, hanem arra, hogy a kész modulokból válasszon. Ezért a tantervben kitűzött kompetenciafejlesztés helyett a Python nyelv használata kerül előtérbe.

- Nincs rekord, struktúra. Helyette a szótár típust használja, aminek a tantárgy másik, adatbáziskezelés témakörével való kapcsolatát ezen a tudásszinten nem lehet bemutatni.
- Rekord implementációra másik megoldás lehetne az osztály adatstruktúra, de – úgy tűnik, – ez a nyelv fejlesztése, ezért szakember feladata, nem nyelvhasználónak való. Ebből következően haladó szintű ismeret a mező szelekció (más nyelvekben a pont operátor). A misztifikálás miatt nem tud kapcsolódni a tananyag többi témakörében és a programozás témakörön belül is gyakran emlegetett objektumokhoz. (Például: betű, kép, táblázat, ellipszis, pixel.)
- A lista mindenre használható, strukturálatlan rekordnak is. Emiatt a különböző adatstruktúrák absztrakciójára nem használható.
- A Python programozói szemlélet a típusalgoritmusok használata elé helyezi a tipikus statisztikai feladatokra kész függvényeket. A tankönyvben ezért a függvények is típusalgoritmusként jelennek meg. Ezzel megkerüli a tantervi előírások kompetenciafejlesztési céljait. A típusalgoritmusok helyett használt függvények zavart okoznak az absztrakciós készség fejlesztésében, mivel a „könnyű" feladatok algoritmizálása helyett függvényt alkalmazva a „nehéz" feladatok algoritmusának elkészítéséhez nincs alapozás.
- Az eldöntés, keresés, kiválasztás típusalgoritmusának operátorral helyettesítése zavart okoz az elvárt absztrakciós szint meghatározásában. Az objektumban „in" operátorral kapott eredmény tulajdonság, de az objektumba belépve sorozaton történő keresés. A tantervben leírt cél a magasabb absztrakciós szint (elemibb megvalósítás), aminek az a megoldás nem tesz eleget.
- Az összeg, szélsőérték-kiválasztás, eldöntés, keresés, kiválasztás – az egyszerű típusalgoritmusok 2/3-a –, ha látó ember végzi, akkor a szem 2D képalkotásból következően nem sorfeldolgozással, hanem szkenneléssel, folt érzékeléssel, memóriában tárolt kép rávetítéssel történik. Ezért a típusalgoritmus nem a természetes végrehajtás számítógépes megvalósítása, hanem egy számítógépre optimalizált, ezért absztrakciót igénylő, megtanulandó módszer. A függvény elfedi a feldolgozásbéli különbséget.

Amikor a tantervben kitűzött célok eléréséhez eszközöket rendelünk, alapfeltétel, hogy az eszköz rendelkezzen a célok eléréséhez szükséges funkciókkal. Amennyiben a konkrét eszköz valami miatt nem elérhető (például életveszélyes lenne), akkor a céloknak megfelelő szimulációval helyettesíthető a funkció. A fenti lista azt mutatja, hogy a Python nyelvből hiányoznak a tantervben megnevezett adattípusok és vezérlési struktúrák, a helyettesítésre használt eszközök más szemléletmódú megoldásokat nyújtanak.

A Python nyelv alkalmas a programozás iránti érdeklődés felkeltésére, de dinamikus adatkezelése elrejti az adatot. Az adatabsztrakció és a típusalgoritmusokat helyettesítő függvények tekintetében a táblázatkezelőkhöz hasonló.

Az általános iskolás korosztály programozásoktatásához a Python – több blokknyelvekkel együtt – megfelelő eszköz, de az erre épülő középiskolai (gimnáziumi) képzésben a kitűzött célok elérését gátolja, hogy a nyelv jellegéből adódóan a gyakorlat az elmélettel nincs összhangban.

Bibliográfia

1. Edsger W. DIJKSTRA: *A Case against the GO TO statement* (letter EDW 215) *Go-to statement considered harmful.* In Commun, ACM (11) (1968); 3, p. 147–148. Online archívum: https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF [Megtekintés: 2022.10.31.]
2. Edsger W. DIJKSTRA: *Notes on Structured Programming* In: E.W.D Archive the manuscript of Edsger W. Dijkstra, EWD249 https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF [Megtekintés 2022.10.31.]
3. Edsger W. DIJKSTRA: *What led to "Notes on Structured Programming"*. Nuenen, 2001.06.10 In: E.W.D Archive the manuscript of Edsger W. Dijkstra, EWD1308 https://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1308.PDF [Megtekintés 2022.10.31.]
4. Vuorikari, R., Kluzer, S. and Punie, Y.: *DigComp 2.2: The Digital Competence Framework for Citizens - With new examples of knowledge, skills and attitudes*. EUR 31006 EN, Publications Office of the European Union, Luxembourg, 2022, ISBN 978-92-76-48882-8, doi:10.2760/115376, JRC128415. https://publications.jrc.ec.europa.eu/repository/bitstream/JRC128415/JRC128415_01.pdf [Megtekintés 2022.11.04.]
5. Brodnik et all: Programming for All: Understanding the Nature of Programs. (2021) https://arxiv.org/pdf/2111.04887.pdf [Megtekintés 2022.11.04.]
6. *ECDL Kódolási ismeretek (Python). Syllabus 1.0.* NJSZT, Budapest (2021) https://njszt.hu/hu/ecdl/modul/kodolasi-alapismeretek-python [Megtekintés 2022.10.31.]
7. *Why is Python So Polular?* Pulumi (2022) https://www.pulumi.com/why-is-python-so-popular/ [Megtekintés: 2022.11.06.]
8. David Beazley: *Foreword* In: P. Wentworth, J. Elkner, A.B. Downey, C. Meyers: *How to Think Like a Computer Scientist: Learning with Python 3 (RLE).* (2002) https://openbookproject.net/thinkcs/python/english3e/foreword.html [Megtekintés: 2022.11.06.]
9. *Robotika, algoritmizálás, programozás.* In: Digitális kultúra tankönyv 5., p. 7–34, Oktatási Hivatal (2020) ISBN: 978-615-81539-4-2. https://www.tankonyvkatalogus.hu/pdf/OH-DIG05TA__teljes.pdf [Megtekintés: 2022.11.01.]
10. *Robotika, algoritmizálás, programozás.* In: Digitális kultúra 5 (NAT 2020) okostankönyv. Oktatási Hivatal (2020) https://www.nkp.hu/tankonyv/digitalis_kultura_5_nat2020/fejezet_01_fejezetnyito [Megtekintés: 2022.11.01.]
11. *Digitális kultúra 5–8. évfolyam.* In: Kerettanterv az általános iskolák 5–8. évfolyama számára, Oktatási Hivatal, (2020) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/Digitalis_kultura_F.docx [Megtekintés: 2022.11.02.]
12. *Algoritmizálás, programozás, robotika.* In: Digitális kultúra tankönyv 6., p. 35–60, Oktatási Hivatal (2020) ISBN: 978-615-6256-38-6. https://www.tankonyvkatalogus.hu/pdf/OH-DIG06TA__teljes.pdf [Megtekintés: 2022.11.01.]
13. *Algoritmizálás, programozás, robotika.* In: Digitális kultúra tankönyv 7. p. 39–60, Oktatási Hivatal (2022) ISBN: 978-663-436-290-6. https://www.tankonyvkatalogus.hu/pdf/OH-DIG05TA__teljes.pdf [Megtekintés: 2022.11.01.]
14. *Algoritmizálás, programozás, robotika.* In: Digitális kultúra 6 (NAT 2020) okostankönyv. Oktatási Hivatal (2021) https://www.nkp.hu/tankonyv/digitalis_kultura_6_nat2020/fejezet_03_fejezetnyito [Megtekintés: 2022.11.01.]
15. *Algoritmizálás, programozás, robotika.* In: Digitális kultúra 7 (NAT 2020) okostankönyv. Oktatási Hivatal (2022) https://www.nkp.hu/tankonyv/digitalis-kultura-7-nat2020/fejezet_04_fejezetnyito [Megtekintés: 2022.11.01.]
16. *Algoritmizálás és blokkprogramozás. Robotika.* In: Digitális kultúra 5–8. okosgyűjtemény. Oktatási Hivatal (2022) https://www.nkp.hu/tankonyv/digitalis_kultura_okosgyujtemeny_5-8/fejezet_01_fejezetnyito [Megtekintés: 2022.11.01.]

17. *Digitális kultúra 9–11. évfolyam.* In: Kerettanterv a gimnáziumok 9–12. évfolyama számára, Oktatási Hivatal, (2020) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/Digitalis_kultura_K.docx [Megtekintés: 2022.11.04.]
18. *Bevezetés.* In: Kerettanterv a szakgimnáziumok 9–12. évfolyama számára. Oktatási Hivatal (2020) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/szakkepzes/Bevezetes.docx [Megtekintés: 2022.11.04.]
19. *Digitális kultúra.* In: Közismereti kerettanterv a szakképző iskolák számára. Oktatási Hivatal (2020) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/digitalis.doc [Megtekintés: 2022.11.04.]
20. *A Hátrányos Helyzetű Tanulók Arany János Tehetséggondozó Programja* http://www.ajtp.hu/kozerdeku?csu=AAAVIXVC [Megtekintés: 2022.11.04.]
21. *Digitális kultúra* In: Arany János Kollégiumi Program 9/AJKP évfolyam Oktatási Hivatal (2021) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/AJKP/9_digitkult_ajkp_uj_VD.doc [Megtekintés: 2022.11.04.]
22. *Digitális kultúra (3 óra/hét) 9. évfolyam.* In: Arany János Tehetséggondozó Program kerettanterve, Oktatási hivatal (2021) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/AJTP/Digitalis_kultura_9AJTP_evfolyam_3_ora_2021.docx [Megtekintés: 2022.11.04.]
23. *Digitális kultúra (4 óra/hét) 9. évfolyam.* In: Arany János Tehetséggondozó Program kerettanterve, Oktatási Hivatal (2021) https://www.oktatas.hu/pub_bin/dload/kozoktatas/kerettanterv/AJTP/Digitalis_kultura_9AJTP_evfolyam_4_ora_2021.docx [Megtekintés: 2022.11.04]
24. *Algoritmizálás és programozási nyelv használata.* In: Digitális kultúra 9. tankönyv, p. 93–128, Oktatási Hivatal (2020) https://www.tankonyvkatalogus.hu/pdf/OH-DIG09TA__teljes.pdf [Megtekintés: 2022.11.04.]
25. *Algoritmizálás és programozási nyelv használata.* In: Digitális kultúra 9. (NAT 2020) okostankönyv. Oktatási Hivatal (2021) https://www.nkp.hu/tankonyv/digitalis_kultura_9_nat2020/fejezet_05_fejezetnyito [Megtekintés: 2022.11.04.]
26. *Algoritmizálás és programozási nyelv használata.* In: Digitális kultúra 10. tankönyv, p. 41–90, Oktatási Hivatal (2020) https://www.tankonyvkatalogus.hu/pdf/OH-DIG10TA__teljes.pdf [Megtekintés: 2022.11.04]
27. *Algoritmizálás és programozási nyelv használata.* In: Digitális kultúra 10 (NAT 2020) okostankönyv. Oktatási Hivatal (2021) https://www.nkp.hu/tankonyv/digitalis_kultura_10_nat2020/fejezet_05_fejezetnyito [Megtekintés: 2022.11.04.]
28. *Algoritmizálás és formális programozási nyelv használata.* In: Digitális kultúra 11. tankönyv, p. 151–234, Oktatási Hivatal (2020) https://www.tankonyvkatalogus.hu/pdf/OH-DIG11TA__teljes.pdf [Megtekintés: 2022.11.04.]
29. *Algoritmizálás, formális programozási nyelv használata.* In: Digitális kultúra 11 (NAT 2020) okostankönyv. Oktatási Hivatal (2022) https://www.nkp.hu/tankonyv/digitalis-kultura-11-nat2020/fejezet_07_fejezetnyito [Megtekintés: 2022.11.04.]
30. *Algoritmizálás, programozás (Általános irány).* In: Digitális kultúra okosgyűjtemény 9-12. Oktatási Hivatal (2022) https://www.nkp.hu/tankonyv/digitalis-kultura-okosgyujtemeny-9-12/fejezet_01_fejezetnyito [Megtekintés: 2022.11.06.]
31. *Algoritmizálás, programozás (Tehetséggondozó irány)* In: Digitális kultúra okosgyűjtemény 9-12. Oktatási Hivatal (2022) https://www.nkp.hu/tankonyv/digitalis-kultura-okosgyujtemeny-9-12/fejezet_02_fejezetnyito [Megtekintés: 2022.11.06.]
32. Tim Peters: *The Zen of Python.* https://peps.python.org/pep-0020/ (2022.03.15.) [Megtekintve: 2022.11.08.]