Digitális kultúra

Programozás

1. rész

Géptermi szabályzat

1. A számítástechnika szaktantermekben tanulóknak tartózkodni csak szaktanári engedéllyel lehet.
2. A géptermek szünetekben, használaton kívül zárva tartandók, csak az arra felhatalmazást kapott személyek nyithatják ki.
3. Internethasználat, tanórán kívüli gyakorlás csak felügyelet mellett történhet.
4. Az internetezési, levelezési etikett (Netikett) betartása mindenkire kötelező.
5. Digitális adathordozót csak szaktanári engedéllyel szabad a termekbe behozni.
6. Élelmiszert, rágógumit behozni tilos.
7. Mobiltelefont csak a tanár által meghatározott feladatra szabad használni.
8. A hálózati főkapcsolókat, a központi számítógépeket bekapcsolni; a központi gépekhez nyúlni tanulóknak tilos.
9. A számítástechnikai eszközök belsejébe nyúlni, azokat rongálni anyagi felelősség terhe mellett szigorúan tilos.
10. A számítógépeket bekapcsolni csak a szaktanár engedélyével, az előírásoknak megfelelő módon szabad.
11. Csak a tanár által meghatározott szoftverek használhatók.
12. A szoftverek beállításainak megváltoztatása szigorúan tilos!
13. Ha a foglalkozás ideje alatt a tanuló rendellenességet, a beállítások megváltoztatását, rongálás észlel, ha a számítógépe meghibásodik, azt azonnal köteles jelenteni, ellenkező esetben magára vállalja a felelősséget.
14. A géptermek rendjéért és tisztaságáért, az asztalok, monitorok, gépházak, billentyűzetek, hangfalak (fülhallhatók), egerek állapotáért minden tanuló egyénileg is felelős.
15. Minden tanuló csak a saját helyén ülhet, melyet az ülésrendbe tanév elején be kell jegyezni.
16. A géptermi szabályzat mindenkire kötelező, aki bármelyik terembe belép!
17. Amennyiben valaki a géptermi munkarendet megsérti és ezzel anyagi kárt okoz – akár gondatlanul, akár szándékosan –, a szülő (törvényes képviselő) köteles a kárt a vonatkozó jogszabályok rendelkezése szerint megtéríteni.

Megjegyzések:

A géptermi szabályzat a 21. és 22. terem használatára egyaránt vonatkozik!

7. pont: Alapértelmezésben a termekben mobiltelefont használni tilos. Ez alól kivételt jelentenek azok a feladatok, amelyek során a mobiltelefon használata szükséges a feladat megoldásához. Ezen felül használható mobiltelefon vagy tablet az elektronikus segédanyagok (mint ez a dokumentum) eléréshez is.

12. pont: A szoftverbeállítások megváltoztatásának a tilalma a Windows rendszerre érvényes. A Linux (Ubuntu) rendszer esetében minden felhasználó saját beállításokkal rendelkezik, így azok szabadon megváltoztathatóak. Azonban a hálózati beállításokat itt sem szabad módosítani, mert az a gép működőképességére lehet hatással!

8. pont: A hálózati főkapcsolókra vonatkozó tilalmat szükség esetén felülírhatja, ha balaset- vagy tűzvédelmi okokból szükséges a terem azonnali áramtalanítása.

Jelmagyarázat

A dokumentum fő szövege ilyen betűkkel jelenik meg, azzal a háttérszínnel, amelyet ezen az oldalon látsz. Ez a szöveg magyarázatot, megjegyzendő ismereteket tartalmaz.

Az ilyen szövegdobozba írt szöveg kiegészítő ismereteket, megjegyzéseket, érdekességeket tartalmaz. Ezeket is célszerű elolvasni, mert tartalmazhat olyan információt, amelynek még hasznát veheted...

Amennyiben az oldal háttérszíne olyan, mint ennek a szövegdoboznak, akkor az egész oldal olyan kiegészítő ismereteket tartalmaz, amely nem része a hivatalos tananyagnak.

Amennyiben ilyen szövegdobozt látsz – esetleg szaggatott helyett folyamatos szegéllyel –, akkor abban egy megoldandó feladat leírását találod. Elképzelhető, hogy a feladatot közösen kell megoldanotok, lehet csoportos vagy egyéni feladat is.

Amennyiben az egész oldal egy vagy több feladatot tartalmaz, akkor a szegély elmarad, viszont az oldal háttérszíne megegyezik ennek a szövegdoboznak a háttérszínével.

Az ilyen szövegdobozban fontos definíciókat találsz. Ezeket mindenképpen meg kell jegyezned. Ezek a definiálhatnak egy-egy kifejezest, amelyre később még szükség lesz, vagy valamely fogalom pontos jelentését adják meg. Általában a definíció előtt vagy után további magyarázatot találsz a fogalommal kapcsolatban.

```
A programkód amennyiben nem képen látható, akkor ilyen szövegdobozban lesz mindig elhelyezve. Mint láthatod, itt a betűk fix szélességűek, amely lehetővé teszi, hogy azt is le lehessen a programkódról olvasni, melyik sor elején mennyi szóköznek kell szerepelnie.
```

Amennyiben a dokumentumban valahol egy ilyen QR-kódot látsz, akkor az egy linket takar. Amennyiben a használt PDF-megjelenítő támogatja, akkor a QR-kódra egérrel rákattintva egy webcímet fog a böngészőben betölteni. Ez lehet másik PDF dokumentum, vagy egy tényleges weboldal. Ugyanez a tartalom érhető el telefonnal a QR-kódot leolvasva, mivel a QR-kód ténylegesen tartalmazza az internetcímet.

Ilyen módon a számítógépen megnyitott dokumentum által hivatkozott címet akár az adott számítógépen, akár a telefonon (tableten) meg tudot nyitni.

Amennyiben a PDF megjelenítő a PDF dokumentumban levő címeket nem böngészőben nyitja meg, akkor a beállításokat kell megváltoztatni, amely sajnos a megjelenítő programtól függően történik. Egyes esetekben magában a szövegben is szerepel a cím, így onnan is átmásolható szükség esetén a böngészőbe.

Biztos előfordult már veled is, hogy valamilyen nem túl érdekes tevékenységet kellett valamilyen feladat kapcsán újra és újra végrehajtanod és szívesen rábíztad volna azt a tevékenységet valaki másra, aki nálad gyorsabban és esetleg pontosabban végre tudja azt hajtani.

Sok ember volt már korábban is ezzel így. Így született meg nagyon sok háztartási eszköz. De így születtek meg a különböző robotok is, vagy maga a számítógép is.

Igen, a számítógép egy olyan eszköz, amellyel automatizálható feladatokat lehet gyorsan és precízen végrehajtani. Feltéve, hogy van valaki aki megtanítja arra a feladatra.

Ilyen feladat lehet például amikor minden nap ugyanazt a weboldalt felkeressük az interneten azért, hogy megnézzük, aznap milyen idő várható. Ügyesebb emberek erre írnak egy kis programot, amely helyettük felkeresi azt az oldalt, kimásolja belőle a megfelelő helyen levő tartalmat és csak azt jeleníti meg a képernyőn. Ezzel időt lehet spórolni, hiszen nem kell megvárni, míg az oldal betöltődik, megkeresni az oldalon belül a kívánt részt.

A számítógép felépítése folytán egy speciális gondolkodást igényel a működtetése. Ez a speciális gondolkodási mód szükséges a programok megírásához. Persze ez nem olyan értelemben speciális, hogy túl nehéz dolog lenne elsajátítani ezt a gondolkodásmódot. Ráadásul a programozói szemlélet az élet más területein is hasznos lehet. Ezért javasolják a világon egyre többen, hogy azok is tanuljanak meg legalább alapszinten programozni, akiket egyáltalán nem érdekel a számítógépek világa.

A számítógépek olyan feladatokat hajtanak végre, amelyek pontosan definiált lépések sorozatából állnak. Az ilyen feladatok egy tevékenység automatizálását jelentik. A lépések leírását pedig *algoritmus*nak nevezzük. (Később szerepelni fog az algoritmus pontosabb definíciója is.) Az ilyen algoritmusokban gondolkodás az a gondolkozásmód, amelynek elsajátítása a célja a Digitális kultúra tantárgy **Algoritmizálás és programozási nyelv** nevű témakörének – magyarul: a programozás tanításának.

Akár programozónak készülsz, akár történésznek, az algoritmusok és az algoritmikus gondolkodás még hasznos lesz a számodra.

Ez a dokumentum az alapokat tartalmazza. Ehhez egy könnyen tanulható, de a programírás íratlan szabályainak elsajátítását bizonyos szigorúan vett szabályain keresztül is segítő nyelvet használ, amely manapság az egyik legnépszerűbb programozási nyelv a világon: a Python nyelvet.

Amennyiben a tantárgyat az alap heti két órában tanulod, akkor 9. osztályban ennek a kötetnek a tartalmát fogod megismerni, a második kötet tartalmát pedig a témakör folytatásában 11. osztályban. Amennyiben viszont emelt óraszámban tanulod a tantárgyat, akkor éppen ez az a témakör, amely a legtöbbet kapja a többlet órákból, így a második kötet tartalmát bőven megismered még a 10. osztály vége előtt.

Hol volt, hol nem volt, volt egyszer egy magyar származású amerikai matematikus, akit a világ *John von Neumann* néven ismert meg, holott a Magyarországról való távozásáig **Neumann János**nak hívták. Aránylag rövid élete ellenére nagyon terjedelmesnek mondható a munkássága. A számítógépek szempontjából a munkásságának a legfontosabb eleme mégis az, hogy még az elektronikus számítógépek fejlesztésének hajnalán, 1945-ben megfogalmazta (*First Draft of a Report on the EDVAC*) a számítógépek működési elvét. Az általa megfogalmazottaknak megfelelően működik mind a mai napig a legtöbb számítógép. Az angol nyelvű szakirodalomban az ezen elvek szerint működő gépeket mind a mai napig *von Neumann-architecture* néven emlegetik. Mi magyarok egyszerűen Neumann-elvről szoktunk beszélni (esetleg többes számban).

Neumann János (John von Neumann) a Los Alamos National Laboratry képén

Számunkra a Neumann-elvből most az a fontos, hogy Neumann János egyértelművé tette, hogy a számítógépeknek a működésükhöz a bináris számábrázolást kell alkalmazniuk, illetve hogy az utasításokat és az adatokat ugyanabban az elektronikusan működő memóriában kell tárolnia a gépnek.

A számítógép egyes elemeit egy-egy vezérlőjel irányítja. Ez a vezérlőjel a bináris működés esetén azt jelenti, hogy a megfelelő vezetékben vagy van feszültség vagy nincs. Ez a két érték egy bitnek felel meg. Amennyiben az egyes vezérlőjeleket egy meghatározott sorrendben egymás mellé tesszük, akkor minden egyes műveléthez hozzárendelhető egy meghatározott bitsorozat. Valójában ezt a bitsorozatot kell az adott műveletet előíró utasítás kódjának tekinteni. Ez az, amit a Neumann-elv szerint a memóriában tárolni lehet. Az ilyen utasításkódokat nevezzük **gépi kód**nak.

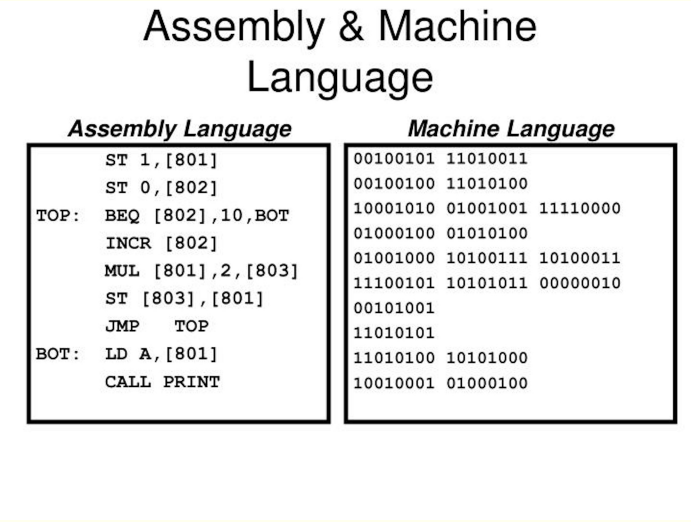
Röviden a Neumann-elv:

1. A számítógép legyen teljesen elektronikus és bináris működésű.
2. Az utasításokat és az adatokat ugyanabban a teljesen elektronikus működésű memóriában tárolja. Azok értelmezése csak a felhasználásuktól függjön.
3. Az utasításokat egy vezérlő egység hajtsa végre.
4. A számítási műveleteket egy aritmetikai-logikai egység hajtsa végre.
5. Az utasításokat a program által meghatározott sorrendben, sorban kell végrehajtani.
6. A számítógép legyen univerzális.

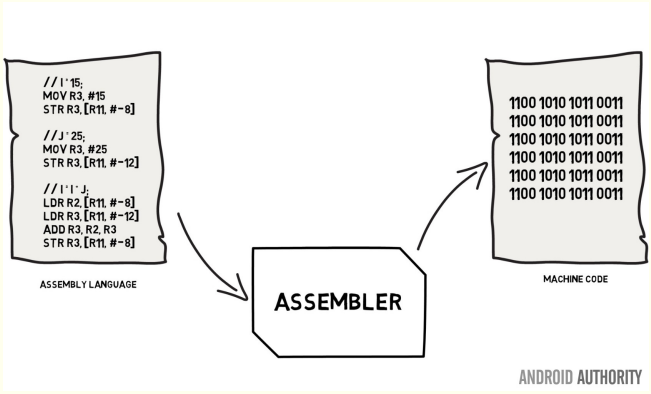
Természetesen nem minden bitsorozat fog értelmes műveletet jelenteni. Az egyes mai processzorokon értelmes bitsorozatok halmazát nevezzük az adott processzor *utasításkészletének*. A mai processzorok esetében persze már nem feltétlenül felel meg minden bit egy-egy vezérlőjelnek, de ennek csak az az oka, hogy egyes processzorok az utasításkód alapján a belső áramköreikkel állítanak elő valamilyen vezérlőjeleket.

Neumann János idejében a számítógépeket még főképpen komolyabb matematikai számítások gyorsabb elvégzésére tervezték. Ekkor még magától értetődő volt, hogy a végrehajtandó utasássorozatokat is matematikusok állították elő. Ám az ilyen bináris számsorozatok formájában történő programírás még egy matematikus esetében sem könnyű feladat. Egy-egy utasítás ugyanis nagyon alapvető elemi műveletet jelent, mint a memória valamelyik cellájának tartalmát átmásolni a processzor egy adott regiszterébe, két regisztert összeadni, stb. Így például egy gyökvonás megvalósítása is akár húsz-harminc elemi utasítás megfelelő sorrendben való végrehajtását igényli. Ehhez pedig még hozzájött, hogy ezeket az elemi utasításokat bináris számsorként kellett megadni.

Ezért elég hamar felmerült az igény arra, hogy az egyes elemi műveleteket a bináris számsor helyett valamilyen szimbólumokkal lehessen jelölni. Meg is született az egyes számítógépekre az a program, amely egy egyszerű szimbólumrendszerben felírt utasássorozatot át tudott írni az utasítások bináris kódjává. Így – bár még mindig matematikusokat igénylő feladat maradt – az egyes programok megírása valamivel könnyebbé vált. Az így kapott nyelvet nevezzük **assembly** nyelvnek. Ez még mindig csak egy jobb felírását adja a programnak a gépi kód helyett.



1.1. ábra: Az assembly és a gépi kód összehasonlítása



1.2. ábra: Assembly kód átalakítása gépi kódra

A számítógépek azonban továbbra is csak a saját gépi kódjukon megírt programot tudják végrehajtani. Így aztán minden egyes processzorfajtához a saját assembly nyelvén megírt kódot egy **assembler**nek nevezett programnak kell átalakítani.

Az assemblernek aránylag egyszerű a feladata: az egyes utasításszavakat át kell fordítania a megfelelő gépi kódra. Hogy pontosan melyikre, abban mindössze az ad némi nehézséget, hogy ugyanazt a műveletet különböző fajtájú adatokon bár ugyanazzal a névvel jelöli az assembly, ezek gépi kódja általában néhány bitben eltér.

Ahogy a számítógépek fejlődtek, olcsóbbá, ezáltal egyre több célra elérhetővé váltak, a velük elvégezhető műveletek kidolgozása a matematika egy új ágát, a *számítástudományt* hívta életre. Ennek az alkalmazott matematikai területnek eredményeként egyre többféle – először kifejezetten matematikai – feladat megoldására dolgoztak ki automatizált módszereket, amelyeket a megfelelő elemi műveletekre bontva azokat számítógépen végre lehetett hajtani. Gyakorlatilag az ilyen megoldási módokat nevezzük algoritmusoknak (amit kicsit később még pontosan definiálni is fogunk).

Ezeknek az algoritmusoknak a leírására már az assembly nyelvnél általánosabb leírást lehetővé tevő módszer kellett. Ez szülte az igényt arra, hogy a számítógépet megtanítsák az emberi nyelv megértésére.

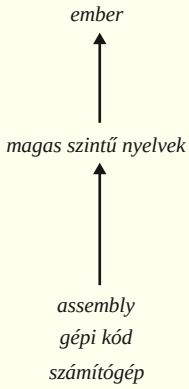
Bár ez a cél azóta sem teljesült, megszülettek újabb és újabb mesterséges nyelvek, amelyek az emberi nyelv (elsősorban az angol nyelv) leegyszerűsítésével próbáltak megoldás találni arra, hogy a számítógép programozási nyelve hasonlítson az emberi nyelvre.

Ezzel megszületett a *magas szintű programozási nyelv* fogalma, vele együtt a fentebb bemutatott nyelvekre (a gépi kódra és az assembly-re) az *alacsony szintű programozási nyelv* elnevezés.

A magas szintű programozási nyelvek már elvonatkaztathatóak egy adott számítógép-típustól. Így lehetővé teszik olyan program írását, amely bármely típusú számítógépen futhat – persze bizonyos feltételek teljesülése esetén. Cserébe sokkal nagyobb méretű kódot eredményezhet, amelynek nagyobb memóriafelhasználás és lassabb futás lehet az eredménye. Ezért sokszor a magas szintű nyelven megírt programoknak is a legkritikusabb részeit assembly nyelven írják meg a hatékony, gyors futást eredményező kód elérése érdekében.

A következőkben röviden bemutatásra kerül néhány ilyen programozási nyelv abból a rengetegből, amelyet mostanáig kifejlesztettek. Ezek némelyike ma már nem használatos és lesznek a jövőben újabbak, amelyeket ma még nem ismerünk.

Wikipédia: Alacsony szintű programozási nyelv szócikk



1.3. ábra: A számítógépes nyelvek hierarchiája a számítógép és az ember nyelve között

Wikipédia: Magas szintű programozási nyelv szócikk

Programozási nyelvek

Fortran: Az egyik első szélesebb körben elterjed programozási nyelv volt. Eredetileg az IBM még az 1950-es években fejlesztette ki, kifejezetten tudományos számítások elvégzéséhez. Azóta használják olyan bonyolult számításokat igénylő területeken, mint például a numerikus időjáráselőrejelzések, folyadékszimulációk, geofizika stb. Olyan programok írásához is használatos, amelyekkel a leggyorsabb szuperszámítógépek teljesítményét tesztelik le. A Fortran több később született nyelv mintájául is szolgált. Ezek közül az egyik a BASIC nyelv. 2021. márciusában a programozási nyelvek közül a 22. legnépszerűbbnek mérték.

Van egy mondás, miszerint az informatikus lusta. Már a Fortran megalkotása is bizonyítja ennek igazságát: John Backus mondta erről egy 1979-es interjúban. „Much of my work has come from being lazy. I didn't like writing programs, and so, when I was working on the IBM 701, writing programs for computing missile trajectories, I started work on a programming system to make easier to write programs."

Az idézet fordítása: „A munkám során sok dolog származik a lustaságomból. Nem szerettem programokat írni, így amikor az IBM 701-en dolgozva lőelemszámító programok írásával foglalkoztam, nekiálltam írni egy programfejlesztő rendszert a programírás megkönnyítésére."

A Fortran nyelvben a változók ha I-N betűvel kezdődnek, akkor alapértelmezésben egésznek (integer), egyéb esetben alapértelmezésben valósnak (real) számítanak, amit persze felül lehet bírálni. Ebből származott az a magyarra nem fordítható mondás, miszerint „In Fortran, GOD is REAL (unless declared INTEGER).”

ALGOL: Az *Algorithmic Language* rövidítéséből származik a nyelv neve. Eredetileg 1958-ban fejlesztették ki. A különböző tankönyvekben és egyetemi szakkönyvekben az algoritmusok leírásában szabvánnyá vált az ALGOL nyelv használata. A manapság használt programozási nyelvek nagyrészére szokás mondani, hogy a nyelv formája Algol-szerű. Ezek a nyelvek ugyanis többé-kevésbé az Algol nyelvnél alkalmazott elveket követik. A Fortran nyelv több hátrányát kiküszöbölte, utat nyitva a modernebb nyelvek létrejöttének.

COBOL: Egy angol-szerű programozási nyelv. 1959-ben fejlesztették ki. Neve a *common business-oriented language* kifejezés rövidítéséből származik. 2002. óta objektum-orientált nyelv. Elsősorban üzleti, pénzügyi és adminisztrációs feladatokat ellátó rendszerek fejlesztésében használják, céges és kormányzati környezetben egyaránt. Nagygépes rendszereken manapság is széles körben elterjedt a használata ugyan, de manapság a COBOL-ban írt programokat egyre inkább átírják modernebb nyelvekre. Bár a többi nyelvnél jobban hasonlít az angolra, de pont ezt hozzák fel a bírálataként is, hiszen túl bonyolulttá, bőbeszédűvé teszi a programot. Még 2012-ben is az üzleti világban használt programok 60%-a COBOL-ban írt program volt.

BASIC (*Beginners' All-purpose Symbolic Instruction Code*): Egy általános célú, magas szintű programozási nyelv, amely megtervezésének célja a könnyű használhatóság volt. 1964-ban adták ki az első változatát a Darthmouth College-en. Tervezője Thomas E. Kurtz és az a John G. Kemeny volt, aki 1926. május 31-i születésekor még a Kemény János nevet kapta Budapesten. (Mellesleg a Dartmouth Collage 13. elnöke volt a Wikipédia szerint.) A nyelvet sokáig használták a programozás oktatására, ami mai szemmel nézve ugyan nem tűnik jó ötletnek. Abban az időben viszont még szinte bármilyen számítógép-felhasználáshoz a felhasználónak magának kellett programot írnia. Az akkoriban egyre olcsóbbá váló miniszámítógépeken ez a nyelv tette lehetővé a számítógépek szélesebb körű felhasználást. Több olyan számítógép is készült (például a Commodore cég gépei), amelyen alapból a BASIC nyelv valamely változata volt elérhető.

Simula: Amint a neve is utal rá, eredetileg szimulációs célra szóló alkalmazások készítésére szánt nyelv. Az első verzió, amely 1962-ben jelent meg (Norvégiában fejlesztették ki) valóban szimulációs feladatokra volt alkalmas, de a második változata, a **Simula 67** már általános célú nyelv volt. Ennél a nyelvnél jelent meg először az objektum-orientált programozás lehetősége. A modern, objektum-orientált nyelvek kialakulására nagy hatása volt.

Pascal: A svájci *Niklaus Wirth* az Algol egy fejlesztésének szánta eredetileg, végül a francia matematikus *Blaise Pascal* tiszteletére **Pascal** névre keresztelte ezt a nyelvet, amely 1970-ben jelent meg és elég hosszú ideig népszerű nyelv volt. Elsősorban a jó programozási gyakorlatok alkalmazását támogató nyelvnek szánta Wirth. Valóban kiszorította a kevésbé modern programozási módszereket alkalmazó BASIC nyelv használatát elég hamar. A nyelv később objektum-orientált elemekkel is bővült, amely elvezettett a ma is használt **Delphi** nyelv létrejöttéhez.

Ada: A világ első programozójának tekintik az angol költő, Lord Byron lányát Augusta Ada King bárónőt, aki Charles Babbage gépéhez készített programokat. Róla kapta a nevét az Ada programozási nyelv, amely többek között a Pascal nyelvből alakult ki.

1.4. ábra: Alfred Edward Chalon: Ada Lovelace portréja, akvarell 1840-ből

Dennis Ritchie (1941–2011)

Bjarne Stroustrup

BCPL, **B**, **C**: Eredetileg compilerek írásához tervezett, ma már nem használt programozási nyelv volt a BCPL (a compilerekről a következő leckében lesz szó). Ebből kiindulva alkotta meg *Ken Thompson* és *Dennis Ritchie* a **B** nyelvet. Ennek további bővítésével hozták létre a **C** nyelvet, amely a mai napig az egyik alapnyelvnek tekinthető. Sőt, Thompson és Ritchie egy operációs rendszer fejlesztésével is foglalkozott, amelyet bár először Assembly nyelven kezdtek el írni, de mivel a C nyelv minden ismert számítógépre rendelkezett fordítóval, így átírták az addig elkészült kódot C-re. Az így kifejlesztett operációs rendszer lett a *Unix*, amelyet a Windows kivételével minden ma használatos operációs rendszer ősének lehet tekinteni. Mivel az 1960-as években zajlott mind a C nyelv, mind a *Unix* kifejlesztése, hivatalosan 1970. január 1-t tekintjük a *Unix* születésnapjának. Ennek az a jelentősége, hogy a legtöbb számítógépen az időmérés az ekkortól eltelt idő alapján történik.

C++, C#, Java, JavaScript, Objective-C, PHP... Csak néhány példa azokból a nyelvekből, amelyeket közvetlenül vagy közvetve a C nyelvből fejlesztettek tovább.

Smalltalk: Az első igazán objektum-orientált programozási nyelv (a Simula mellett, amely ténylegesen az első ilyen nyelv volt).

C++: A dán *Bjarne Stroustrup* által fejlesztett nyelv, amely elsősorban a C nyelv továbbfejlesztési szándékával jött létre. Gyakorlatilag egy C nyelvű program kis módosítással C++ nyelvű programként is tekinthető. Elsősorban az objektum-orientált programozás lehetőségeit építette be a Simula és a Smalltalk nyelvekben alkalmazottak alapján, de más bővítést is kapott a nyelv. Mivel a változó értékének növelése a C nyelvben a ++ művelettel lehetséges, így a C++ név egyértelműen arra utal, hogy ez a C továbbfejlesztése.

Java: A C++ nyelv mintájára készült, kizárólag objektum-orientált programozást támogató programozási nyelv.

C#: A Microsoft által kifejlesztett; **Objective-C**: Az Apple által kifejlesztett C-nyelvre alapuló, objektum-orientált megközelítést alkalmazó programozási nyelvek. A Java, C# és az Objective-C gyakorlatilag egymás unokatestvérének tekinthetőek mind használatban, mint a nyelvek tulajdonságaiban.

JavaScript: Ennek a nyelvnek a nevén kívül semmi köze a Java nyelvhez! Kifejezetten webes alkalmazásokhoz fejlesztett C, C++ nyelvekre épülő nyelv, amelyet egy idő után már más célra is elkezdtek használni.

PHP: Szintén webes alkalmazások készítésére használt nyelv, amely szintén a C nyelv leszármazottjának tekinthető. A szerver oldali programok írására használatos, a kimenete általában egy olyan kód, amit a webszerver a böngésző programnak elküld megjelenítésre.

SQL: A fenti nyelvekkel ellentétben, amelyek általános célú nyelvek, az SQL egy célnyelv, amely adatbázisok kezelésére használatos. Nem lehet vele másmilyen feladatot végrehajtani, ezért nevezzük célnyelvnek.

Lisp: Egy speciális nyelvcsalád, amely bár általános célúnak számít, de abban speciális, hogy listákkal dolgozik, amelyeknek általában az első eleme a művelet, míg a lista többi eleme olyan adat, amellyel a műveletet el kell végezni.

Python: Egy egyre népszerűbb programozási nyelv, amely ugyan nem a C nyelv leszármazotta, de elég sok elemében látszik a C nyelv, mint minta használata. Könnyen tanulható ugyan, de sok olyan megkötése van, amely a használóját a helyes programozási szokások elsajátítására neveli. Ezért első programozási nyelvnek is kifejezetten alkalmas. A továbbiakban ezt a nyelvet fogjuk a programozás tanulására felhasználni, így a későbbiekben sok tulajdonságát megismered majd.

Az előzőekben bemutatott programozási nyelvek mellett még rengeteg nyelv létezett, létezik és fog megjelenni. Gyakorlatilag lehetetlen az összeset felsorolni, annyi van belőlük. A ma használatos programozási nyelvek legtöbbje valamilyen rokonságban áll a C nyelvvel, amiből adná magát, hogy akkor először nyilván a C nyelvet célszerű megismerni. Ám ha figyelembe vesszük, hogy a C nyelvet szokták *magasszintű assembly* néven is emlegetni, akkor talán ez már utal arra, hogy nem a legbarátságosabb nyelvről van szó.

Hasonló módon lehetne első nyelvnek a C nyelvből származó objektum-orientált programozásra alkalmas nyelvet tekinteni, hiszen manapság az objektum-orientált programozás számít az egyik legfejletteb módszernek. Ám ebben az esetben olyan fogalmakkal kellene már az első program megírása előtt megbarátkozni, mint az *osztály*, *polimorfizmus*, *egybezártság* stb. Figyelembe véve, hogy középiskolában az objektum-orientált programozási módszerek ismerete nem követelmény még az érettségin sem, így nem javasolt olyan nyelvet használni, amely objektum-orientált programozáshoz készült, így sem a Java, sem a C#, sem a C++ nem jó választás.

A Python ugyan szintén erősen épít az objektum-orientál megközelítésre, de úgy is lehet használni, hogy erről nem veszünk tudomást. Könnyen tanulható, jól olvasható nyelv, amely ráadásul nagyon sok feladatra alkalmas. A segítségével lehet egyszerű, de fárasztó számítógépes tevékenységeket könnyedén automatizálni. Talán pont ennek köszönhető a növekvő népszerűsége. Éppen ezért a továbbiakban a programozás alapjait a Python nyelven keresztül fogod megismerni.

További nyelvek is vannak még, amelyek egy részét a következő oldalon levő angol nyelvű Wikipédia oldalakról elindulva is meg lehet ismerni.

A magas szintű programozási nyelven írt programot a számítógép közvetlenül nem tudja végrehajtani. Kell egy tolmács hozzá. Ez a tolmács alapvetően kétféle lehet:

- **Compiler** (*fordító*): Olyan program, amely egy adott nyelvű forráskódot (a magas szintű nyelven írt programkódot) feldolgozva előállítja a vele azonos műveleteket tartalmazó gépi kódú programmá, azaz lefordítja azt gépi kódra. Ennek eredményeként egy új program keletkezik, amelyet a számítógép már végre tud hajtani. Előnye ennek a megoldásnak, hogy a lefordított program már közvetlenül, további segédeszköz nélkül futtatható a számítógépen. Másrészt a fordítást nem feltétlenül kell azon a gépen végezni, amin futtatni akarjuk. Hátránya, hogy a fordítás sokáig tarthat, mivel a fordító optimalizálja is a kódot.
- **Interpreter** (*értelmező*): Ebben az esetben az értelmező szimulál egy olyan számítógépet, amely képes végrehajtani a forráskódot. Azaz maga az értelmező fogja lépésről lépésre haladva a programot végrehajtani a megfelelő műveletek elvégzésével. Hátránya, hogy a program az értelmező nélkül nem futtatható. Előnye, hogy a program végrehajtására nem kell várni a fordítással. Így az értelmezőt használó nyelveket használva gyorsabban lehet tesztelni a programot, vagyis felgyorsul a fejlesztés. Azonban az értelmezők általában valamivel lassítják a program végrehajtását, így általában előbb-utóbb átírják a programot egy fordítót használó nyelvre.

A két fenti módszer hibridizálásával léteznek olyan megoldások is, amikor a forráskódot egy köztes byte-kódra fordítja le egy fordító, majd az így kapott kódot egy értelmezőként funkcionáló futtató környezet hajtja végre. Ez a módszer a fenti módszerek hátrányait egyesíti, mégis ezt használja például a Java és a .NET is.

Ezekről a módszerekről később még lesz szó.

A leckében szereplő képek forrásai

- https://commons.wikimedia.org/wiki/File:JohnvonNeumann-LosAlamos.jpg
- https://image3.slideserve.com/5674619/assembly-machine-language-n.jpg
- https://cdn57.androidauthority.net/wp-content/uploads/2016/03/assembler.jpg
- https://en.wikipedia.org/wiki/Dennis_Ritchie#/media/File:Dennis_Ritchie_2011.jpg
- https://en.wikipedia.org/wiki/Bjarne_Stroustrup
- https://en.wikipedia.org/wiki/Ada_Lovelace

További internetes címek a leckében szereplő témában

- Fortran: https://en.wikipedia.org/wiki/Fortran
- https://en.wikipedia.org/wiki/ALGOL
- https://en.wikipedia.org/wiki/COBOL
- https://en.wikipedia.org/wiki/BASIC
- https://en.wikipedia.org/wiki/List_of_programming_languages
- https://en.wikipedia.org/wiki/Python_(programming_language)
- https://en.wikipedia.org/wiki/Indentation_style

Abu Abdullah Muhammad bin Musa al-Khwarizmi, perzsa matematikus (kb. 780 – kb. 850) eredetileg arabul írt, latinra lefordított művének köszönhetően terjedt el Európában az indiai eredetű számok használata. Ennek a műnek köszönhetően ismerjük ezeket a számokat *arab számok*ként. A matematikus nevének jelentése valami ilyesmi: *Abdullah apja, Musa fia, Mohamed, aki Kwarizm-ből származik.* Nevében a származási helyre utaló kifejezésből alakult ki többszöri átalakulás révén az angol algorithm szó, amelyet magyarul algoritmusnak mondunk.

A szóban forgó mű valójában az indiai számokkal dolgozó matematikai műveleteket, mai szóval algoritumusokat ír le. A középkorban sokáig erre a szerző nevével utaltak, azért lehetett végül az algoritmus szó az ilyen műveletek elnevezése.

Érdekesség még, hogy a mű eredeti arab nyelvű címéből (al-Kitāb al-Mukhtasar fī Hisāb **al-Jabr** wal-Muqābalah) származik a matematika egyik ágának, az algebrának az elnevezése is.

2.1. ábra: Al-Kwarizmi szobra a teheráni Amir Kabir Egyetemen

Hogy érthető legyen, mi is az az algoritmus, nézzünk néhány egyszerű elméleti feladatot!

2.1. Fogalmazd meg a lehető legrészletesebben, milyen tevékenységek sorozatát kell végrehajtani ahhoz, hogy a mobiltelefonodon felhívhasd egy ismerősödet, akinek benne van a telefonszáma a névjegyzékben!

2.2. Milyen tevékenységeket kell végrehajtani ahhoz, hogy egy önkiszolgáló boltban vagy egy bevásárló központban vehess készpénzzel

- 1 liter tejet;
- 4 zsömlét;
- kiflit;
- 1 pohár joghurtot;
- 25 dkg vajat és
- 10 dkg felvágottat?

Mindkét feladat esetén írd le a szükséges lépéseket a füzetedbe!

Az algoritmus valamely feladat megoldására szolgáló, véges számú lépésekből álló műveletsorozat. Ahhoz, hogy valóban algoritmusról beszélhessünk, ennek a műveletsorozatnak az alábbi feltételeket teljesítenie kell:

1. Az algoritmusnak **véges számú lépésben véget kell érnie**. Amennyiben nem ér véget véges lépésben a megoldás, akkor azt legfeljebb *eljárás*nak nevezhetjük, de nem algoritmusnak.
2. A probléma **általános** megoldását kell adnia, azaz bármilyen, a feltételeknek megfelelő bemenő adatokra adja meg a megoldást.
3. Legyen **egyértelmű**: ugyanazokra a bemenő adatokra mindig ugyanazt a megoldást adja. Vannak ugyanakkor olyan eljárások is, amelyeknél nem garantált a mindig azonos végeredmény, mert például valamilyen véletlen eseményt is felhasználnak. Az ilyen eljárásokat szokás *heurisztikus algoritmusok*nak nevezni.
4. Egyértelműen definiálva van az algoritmus **bemenete** és **kimenete**. Azaz az algoritmus leírása pontosan definiálja, hogy mennyi, pontosan milyen bemenő adatot kell megadni az algoritmus számára. Az adatok esetében megadhatóak feltételek, mint például hogy pozitív egész számok legyenek, vagy hogy csak az angol ábécé betűiből álló szó legyen. Szintén pontosan meghatározottnak kell lennie, hogy az algoritmus mennyi és milyen jellegű kimenő adatot fog előállítani.

A definícióban szereplő eljárás szó nem azonos a később, a vezérlési szerkezeteknél említett eljárással. A vezérlési szerkezeteknél az eljárás általában egy algoritmus megvalósítását végző programegység, míg itt ez a kifejezés egyszerűen arra utal, hogy vannak olyan műveletsorozatok, amelyek nem felelnek meg az algoritmus definíciójának.

A számítógépek működési módjukból adódóan mindig valamilyen algoritmust hajtanak végre. Az algoritmusban szereplő műveletek az adott számítógép gépi kódjában létező utasítások. Ezek az algoritmusok így persze sokkal részletesebbnek tűnnek, mint például egy telefonálás leírása. Azonban amennyiben egy számítógép vezérelte robotot akarunk megtanítani telefonálni, akkor a telefonálás algoritmusát is ilyen részletes, elemi műveletek sorozatára kell bontani.

Algoritmus leíró módszerek

A következőkben néhány módszert ismerhetsz meg, amellyel egy algoritmust emberek számára érthető módon lehet megadni. Ezen módszerek ismerete azért szükséges, mert a továbbiakban sok algoritmust fogsz megismerni, amelyek mindegyike az itt bemutatott módszerek valamelyikével – akár ugyanaz az algoritmus több módszerrel is – lesz bemutatva.

Mondatszerű leírás

A **mondatszerű leírás**, vagy **pszeudokód** a normál nyelven írt szöveghez leginkább hasonlító, de egyben a programnyelvekre is hasonlító megoldás egy algoritmus leírására. Gyakran használja a programozási nyelvek struktúrális elemeit egyszerűsített formában a később bemutatandó vezérlési szerkezetek megadására. Azonban mivel a cél az emberi olvashatóság és nem a számítógép számára érthető fogalmazás, hiányoznak belőle olyan részletek, amelyet a programba bele kell írni, de az algoritmus megértéséhez nem szükségesek, vagy a megértést akár egyenesen nehezítenék is. Cserébe visztont olyan magyarázatokat is tartalmazhat, amelyek a programban nem szükségesek, de segítik az algoritmus megértését.

Hivatalos nyelvtan nem létezik a pszeudokódra, de egyes egyetemek esetleg megkövetelik a programozást tanuló diákjaiktól egy adott szabályrendszer alkalmazását.

A mondatszerű leírás alkalmazásának egyik előnye, hogy a leendő program forráskódjában is elhelyezhető megjegyzésekben. A programírás következő lépéseként az egyes részekhez megírható a neki megfelelő programrészlet, így haladva a programfejlesztésben a nagyobb elemektől a részletek felé. Egyben a program dokumentálásának egy része is elkészül ilyen módon, hiszen már eleve ott van a kódban a megkívánt magyarázó megjegyzés.

Az alábbiakban egy példa látható mondatszerű leírásra:

Algorithm LargestNumber
  Input: A list of numbers L.
  Output: The largest number in the list L.

  if L.size = 0 **return** null
  largest ← L[0]
  for each item **in** L, **do**
    if item > largest, **then**
      largest ← item
  return largest

Ugyanez magyarul valahogy így néz ki:

Legnagyobb elem:
Bemenet: L, számok listája.
Kimenet: a legnagyobb szám az L listában, vagy null, ha L üres.

ha L.size = 0 **visszaad**: null
legnagyobb ← L[0]
minden egyes elem-re L-ben, **csináld**:
  ha elem > legnagyobb, **akkor**
    legnagyobb ←elem
ciklus vége
visszaad: legnagyobb

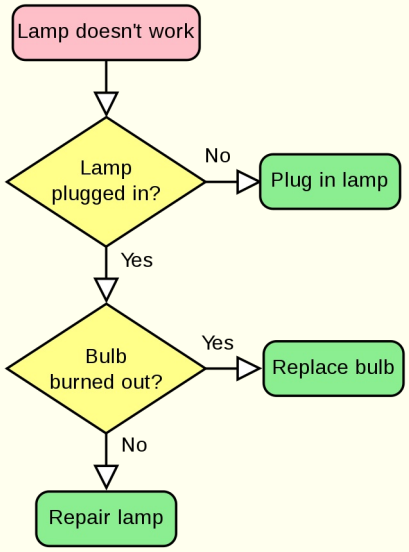
Folyamatábra

A folyamatábra (angolul: *flowchart*) egy folyamat vagy tevékenység grafikus ábrázolására alkalmas. A folyamatábrán ábrázolható egy algoritmus vagy bármely más olyan folyamat is, amelyet lépések sorozatára bontva akarunk ábrázolni.

Az egyes lépéseket különböző alakú blokkokként ábrázolja, amelyeket nyilak kötnek össze. A nyilak mentén lehet haladni az egyes blokkokról tovább a következőre. Ilyen módon grafikusan szemlélteti a végrehajtandó tevékenység lehetséges folyamatát.

A 2.2. ábra a Wikipédia angol oldaláról származik. Azt mutatja be, hogy mi a teendő, ha egy lámpa nem működik. A felső, rózsaszín hátterű blokk a kiinduló állapot, a sárga rombuszok elágazási pontok, ahol a feltett kérdésre adott válasz alapján dől el, hogy melyik nyilat kell követni. A zölt blokkok végrehajtandó tevékenységet jelölnek – egyben az ábrázolt folyamatot befejezését is.

A folyamatábrán alkalmazott blokkok formájára többféle konvenció is kialakult, de általában elmondható, hogy a folyamat kiindulópontját (amelyből mindig egy van), valamint a befejezőpontokat (amiből lehet több is) lekerekített sarkú téglalappal vagy ellipszissel jelzik. Ugyanúgy az egyes féltételek alapján történő elágazási pontokat vagy rombusszal vagy lapított hatszöggel szokás jelölni (utóbbit akkor alkalmazzuk, ha a beleírandó szöveg mennyisége miatt a rombusz nem megfelelő).



2.2. ábra: A wikipédia szócikk példaábrája folyamatábrára: Hogyan kell megoldani a nem működő lámpa problémáját?

Wikipédia: Flowchart szócikk

A program mint algoritmus

A legegyértelműbb algoritumus leíró eszköznek maga a program tűnik. Hiszen a programot használjuk arra, hogy egy adott algoritmus végrehajtására rávegyük a számítógépet.

Ez azonban nem ilyen egyértelmű abból a szempontból, hogy bizonyos programozási nyelvek esetén a program elég nehezen olvasható lehet (gondoljunk csak az Assembly nyelvre). Amennyiben az algoritmusnak egy emberi olvashatóságot biztosító megadására gondolunk, akkor a program nem tekinthető jó megoldásnak – ezért fejlődtek ki más algoritmus leíró módszerek, mint a bemutatott két példa.

Azonban maga a program valóban egy módszer az algoritmus leírására: a számítógép számára írja le az algoritmus lépéseit.

A **program** utasítások gyűjteménye, amelyet egy számítógép képes végrehajtani, ezzel egy adott tevékenységet megvalósítva.

A számítógépes programot általában egy programozó készíti egy adott programozási nyelven. Az ember által olvasható forráskódból valamilyen eszköz állítja elő a számítógép által érthető gépi kódot.

Programok, programkönyvtárak és a hozzájuk kapcsolódó adatok összességét nevezzük ***szoftver***nek.

A leckében felhasznált képek forrása

- https://en.wikipedia.org/wiki/Muhammad_ibn_Musa_al-Khwarizmi
- https://upload.wikimedia.org/wikipedia/commons/9/91/LampFlowchart.svg

További internetes címek a leckében szereplő témában

- https://en.wikipedia.org/wiki/Algorithm
- https://en.wikipedia.org/wiki/Algorithm_characterizations
- https://en.wikipedia.org/wiki/Flowchart
- https://en.wikipedia.org/wiki/DRAKON
- https://en.wikipedia.org/wiki/Unified_Modeling_Language
- https://en.wikipedia.org/wiki/Computer_program

Ahhoz, hogy bármilyen egyszerű programot tudjunk írni, ismerni kell legalább egy programozási nyelvet és egy olyan fejlesztői környezetet, amely segítségével azon a nyelven lehet programot írni.

Igaz, a legtöbb esetben a fejlesztői környezet elkerülhető, hiszen sokszor elegendő lehet akár egy jegyzettömb is a program forráskódjának a megírására, de mégis hasznosabb egy kifejezetten az adott nyelven történő programfejlesztésre szánt programot használni, mivel nagyon sok segítséget tud nyújtani a fejlesztői munkához.

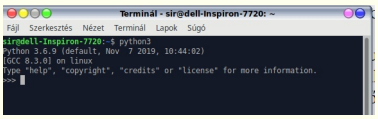
A programozással való ismerkedéshez a jelenleg használt programnyelvek közül talán a legmegfelelőbb a Python, amelynek nem csak egyszerűen tanulható a nyelvezete, hanem elég gyorsan ki is lehet próbálni a megírt programot, nem kell sokat várni a lefordítására. Rendkívül sok kiegészítő lehetőség létezik hozzá, amelynek köszönhetően sok területen jól használható annak ellenére, hogy értelmezett nyelvként elég lassú tud lenni egy-egy nagyobb Pythonban írt program.

Egy programot a forráskódjából kétféle úton lehet futtatni:

1. **Értelmező (Interpreter)** használata esetén az interpreter a program egyes utasításait egymás után végrehajtja. Ilyen esetben a számítógép helyett maga az interpreter hajtja végre a programot, amelynek jelen kell lennie a számítógépen a program minden egyes futásakor.
2. **Fordító (Compiler)** használata esetén a programot a fordítóprogram lefordítja a számítógép processzora által ismert nyelvre, azaz gépi kódra. Ezután az így kapott állományból a program bármikor futtatható a fordítóprogram jelenléte nélkül is.
3. Hibrid megoldásként létezik olyan megoldás, amikor a fordítóprogram egy köztes kódot állít elő, amelyet a program futtatásakor egy értelmező hajt végre. Ilyet használ például a Java és a C# is.

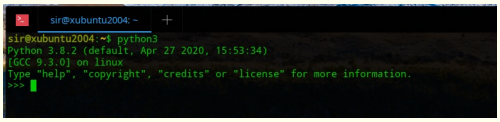
Kezdjük az ismerekedést a Python értelmezőjével és néhány egyszerű utasítás végrehajtásával!

Linux használata esetén nincs más teendőd, mint egy parancsértelmezőt indítani. Ehhez az Xfce grafikus környezetet használva elég egyszerűen az alsó panelen a fekete téglalapot mutató Terminálemulátor gombra kattintanod. A Deepin környezet használata esetén az alsó panel bal szélső, Indító gombjával elérhető rendszermenüt kell használni (ugyanezen a módon Xfce-ben is meg lehet a Terminálemulátort keresni). Itt a „*Minden kategória*” részen belül, a „*Rendszer*” kategóriában találod a „*Terminál*” nevű programot.



3.1. ábra: Python értelmező az Xfce terminálemulátor ablakában

Akármelyik – vagy bármely más – módszerrel sikerül egy parancsértelmezőt megnyitni, akkor oda a `python3` parancsot kell begépelned. Ennek hatására elindul a python értelmező, amelyet a 3.1. és 3.2. ábrán láthatsz.



3.2. ábra: Python értelmező a Deepin terminálablakában

Az értelmező először kiírja a pontos Python verziót, majd a legalsó sorban egy promptot jelenít meg, amely után utasítást vár, amit kész azonnal végrehajtani.

Próbáljuk is ki, hogyan lehet számológépként használni! Írd be a prompt után:

```
2 + 2
```

A szóközök az összeadásjel körül nem lényegesek ugyan, de illik figyelni az esztétikus és olvasható megjelenésre is, így azokat is gépeld be! Mi történik az Enter billentyű lenyomása után?

A feladatban szereplő kifejezésekben a műveleti jeleket szokás *operátor*nak is nevezni, míg az értékeket, amelyekkel az adott művelet elvégzendő, *operandus*nak nevezzük.

3.1.
Az előző oldali példa egy kifejezés, amely önálló utasításként is használható. Az értelmező az egyes kifejezések eredményeit kiírja válaszul a képernyőre. Ezt a lehetőséget felhasználva, derítsd ki, milyen műveleteket jelentenek az alábbiak! Bár az alábbi felsorolásban minden értéknél a 2 szerepel, használj minél többféle értéket az egyes műveletek viselkedésének tesztelésére. Az eredményt jegyezd fel a füzetedbe az egyes műveleti jelek jelentésének meghatározásaként!

```
- 2 + 2
- 2 - 2
- 2 * 2
- 2 / 2
- 2 // 2
- 2 % 2
- 2 ** 2
```

3.2.
Miután a fenti műveleti jelek jelentését sikerült megfejteni, próbálkozz összetett kifejezésekkel is! Például az alábbiakkal. Mit jelentenek az itt szereplő szimbólumok?

```
50 - 5 * 6
( 50 - 5 ) * 6
( 50 - 5 * 6 ) / 4
```

3.3.
Mindezek után próbáld ki az alábbi kódrészletet is és derítsd ki, mit csinál pontosan!

```
width = 20
height = 5 * 9
width * height
```

A feladatban látható nevek egy-egy *változót* jelölnek (később még sokat fogunk ilyeneket használni). A változó egy tároló, amelyben különböző értékeket lehet elhelyezni az értékadó utasításokkal. Ezeket mutatja a példa. Mint látható, amennyiben az adott változó értékét akarjuk egy kifejezés baloldali operandusaként felhasználni, akkor van lehetőség a kifejezés rövidítésére a megfelelő értékadó operátorral.

3.4. Próbáld ki, mi történik az alább kódrészlet hatására!

```
alma = 1                        alma /= 2
alma += 5                       alma
alma                            banán = alma
alma -= 2                       alma //= 2
alma                            alma
alma *= 2                       banán %= 2
alma                            banán
```

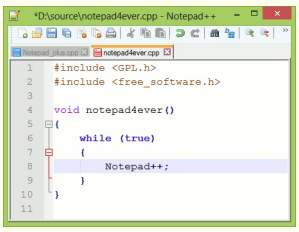
Miért kaptad azokat az eredményeket, amiket kaptál? Mit csinálnak ezek az utasítások?

3.5. Természetesen bonyolultabb kifejezéseket is lehet alkalmazni. Ám ilyenkor már a műveleti sorrendre is figyelni kell! Számold ki a Python interpreterét használva az alábbi kifejezések értékét!

$$\frac{23 \cdot 12}{15+3} \qquad \frac{4-11}{5+9} \cdot 21 \qquad \frac{42+13}{3 \cdot 8} \cdot \frac{1024^{-4}}{51/6}$$

A program forráskódját meg lehet írni egy egyszerű szövegszerkesztő programban, mint amilyen Windows-on a Jegyzettömb (notepad) vagy Linuxon például a Mousepad vagy a KWrite. Ebben az esetben egy egyszerű szöveges állomány szerkesztéséhez szükséges alapfunkciók állnak csak rendelkezésre. A program fordítása és futtatása ebben az esetben parancsorból történthet meg.

Ennél a „fapados” megoldásnál kicsivel több lehetőséget kínál Windows alatt például a notepad++ (4.1. ábra, az ábrára kattintva a program honlapjára lehet jutni) nevű program, amely már képes szintaxiskiemelésre is. A *szintaxiskiemelés* (syntax highlight) azt jelenti, hogy a forráskód különböző funkciójú elemeit másmás színnel jelzi, ami nagy segítség tud lenni például azzal, hogy a színekhez hozzászokva hamarabb észrevehető a nem megfelelő színből, ha valamit elírtunk.



4.1. ábra: A notepad++ program képernyőképe a program honlapjáról (https://notepad-plus-plus.org/)

Az igazán profi programfejlesztés során azonban kifejezetten az adott programozási nyelv lehetőségeit ismerő szerkesztő programokat használnak. Ezek a programok az ún. *fejlesztő környezetek*, vagy az eredeti angol nevük, az *Integrated Development Environment* rövidítéséből származó **IDE** néven emlegetett programok. Ezek a programok általában nem csak a forráskód begépelését teszik könnyebbé, hanem további, a fejlesztés során szükséges műveletet is elérhetővé tesznek. Általában egy jó fejlesztő környezet használata során lehet, hogy a program teljes fejlesztési folyamata az IDE-n belül elvégezhető.

Léteznek egy adott nyelvre specializálódott fejlesztő eszközök. De léteznek olyan programok is, amelyek akár azonnal, akár megfelelő bővítések hozzáadásával akár több, különböző programozási nyelv fejlesztéséhez is biztosítani tudják a megfelelő eszközöket.

Az egyik ilyen több nyelvhez is használható fejlesztő környezetet (Java nyelven) a *JetBrains* fejlesztette ki. Ennek a Python nyelvhez készült változata a **PyCharm**, amelyet órán használni fogunk.

4.2. ábra: A PyCharm indítógombja

Az iskolai gépeken, Xfce környezet használata esetén az alsó panelen a 4.2. ábrán látható gombot kell keresni (ez az ikonja a Python nyelvnek), a főmenüben a program nem elérhető, de parancsértelmezőben az alábbi parancs begépelésével is elindítható:

```
pycharm &
```

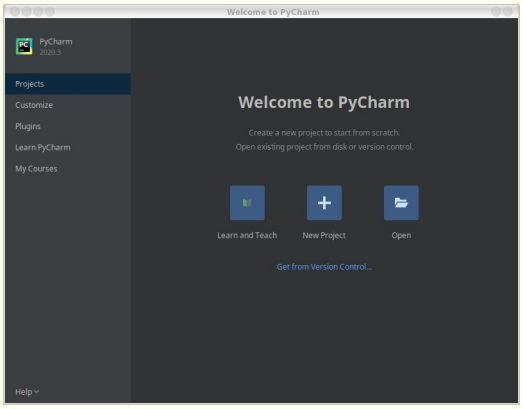
A parancs végén az & jel elhagyható, de ha elhagyjuk, akkor az adott parancsértelmező ablakot addig nem használhatjuk másra és nem is zárhatjuk be, amíg a PyCharm fut! Ez a karakter gondoskodik arról, hogy a parancsértelmezőben újabb parancsokat lehessen kiadni.

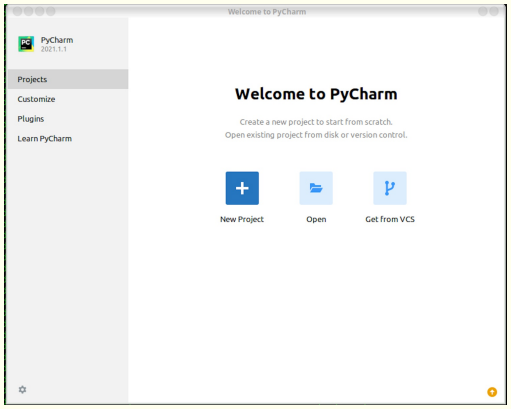
Linux parancsértelmezők esetében megkülönböztetünk előtérben és háttérben futó parancsokat. Alapesetben ha kiadunk egy parancsot, akkor az előtérben fut. Ez azt jelenti, hogy a parancs indításától a parancsértelmező vár, amíg a parancs végrehajtása megtörténik. Amennyiben a parancs egy programot indít, akkor annak a programnak a befejezéséig vár. Ekkor amíg a program fut, a parancsértelmező nem reagál semmire (esetleg a CTRL-Z billentyűkombinációval a parancs és vele a futó program megszakítható).

Ezzel ellentétben – különösen ha önálló ablakban futó programot akarunk elindítani – lehetőség van a parancsértelmezőt arra is rávenni, hogy a kiadott parancsot a háttérben hajtsa végre. Ebben az esetben a parancsértelmező nem várja meg a kiadott parancs/program végrehajtásának befejezését, hanem azonnal várja a következő parancsot. Ahhoz, hogy egy parancsot a háttérben indítsunk el, a parancs végére kell tenni egy & jelet, ezzel mondva meg a parancsértelmezőnek, hogy ez a parancs háttérben hajtandó végre.

Akár az előtérben, akár a háttérben indítottuk el a programunkat, a parancsértelmező ablakának bezárása az elindított program megszakadását eredményezheti!

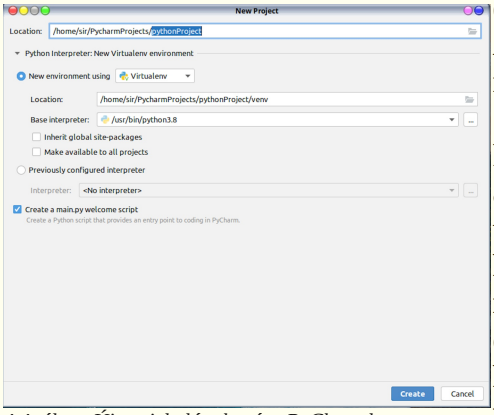
A PyCharm programnak három változata van: A Community változatot érdemes otthon telepíteni, mivel ez ingyenes és mindent tud, amire szükséged lehet. Esetleg választhatod az Educational változatot is, amely szintén ingyenes és tartalmaz néhány angol nyelvű oktató kurzust is, amellyel akár önállóan is ismerkedhetsz a nyelvvel. A harmadik változat a Professional, amely fizetős. Ezt a profiknak szánták, így egyelőre nem lesz rá szükséged!





4.3. ábra: A PyCharm nyitóképernyője. Baloldalt az eredeti, sötét színben, jobboldal világos színtéma beállítása után.

Az iskolai gépeken jelenleg a Community változat van telepítve, mert nem valószínű, hogy órán használnánk az Education többlet elemeit, de azokat utólag is fel lehet telepíteni felhasználónként. A program elindítása után a 4.3. ábrán láthatóhoz hasonló ablak fog megjelenni (a verzió változásával az ablak felépítése is megváltozhat).



4.4. ábra: Új projekt létrehozása PyCharmban

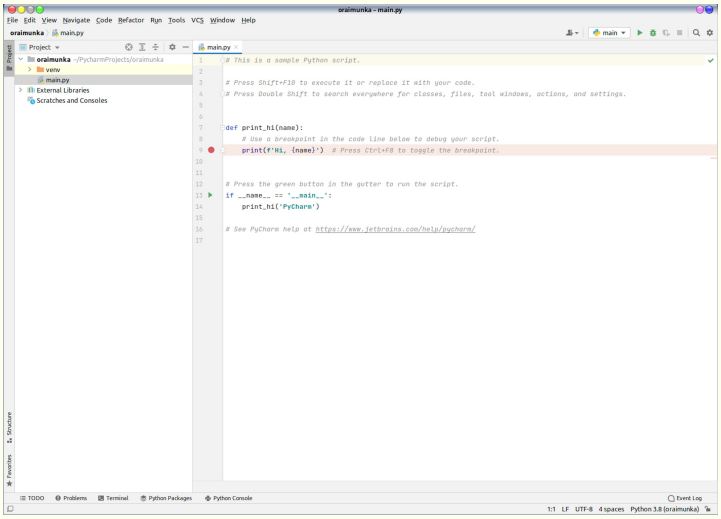
Ebben az ablakban lehet az esetleg már létező projektek közül választani vagy új projektet kezdeni. Amennyiben még nincsenek projektek – első alkalommal nincsenek –, akkor nyilván az első teendő egy új projekt kezdése.

A programfejlesztés során általában minden egyes program külön projekteket jelent. A projektek a legtöbb fejlesztő környezet esetében egy külön, az adott projekthez tartozó állományok összegyűjtésére használt könyvtárt jelentenek. A PyCharm is minden projektünk számára külön könyvtárt fog létrehozni. Mivel az iskolai gépek hálózatba vannak kapcsolva és egy projekt létrehozása során sok állományművelet történik, a hálózati adatforgalom csökkentése érdekében órán minden programunkat egy projektben fogjuk elhelyezni. Később esetleg egy-egy nagyobb témakörhöz majd elképzelhető újabb projektek létrehozása, de jelenleg elégedjünk meg egy **`oraimunka`** nevű projekt létrehozásával!

Válaszd ki a „*New project*” feliratot, amelynek hatására a 4.4. ábrához hasonló ablakot kell kapnod. Itt a kijelölt `pythonProject` szöveg helyére írd be: `oraimunka`!

Ezután elég a „Create” gombra kattintani, majd várni pár órát...

Az ábrán látható, hogy a PyCharm alapból létrehoz egy PycharmProjects nevű könyvtárat a home-könyvtáradban és minden projektnek ezen belül hoz létre alkönyvtárat. Ez az információ a későbbiekben még jól fog jönni, ha például egy-egy programot meg kell keresned (például, hogy hazavidd, leadd feladatmegoldásként vagy ha egy otthonról hozott programot a megfelelő helyre akarsz másolni).



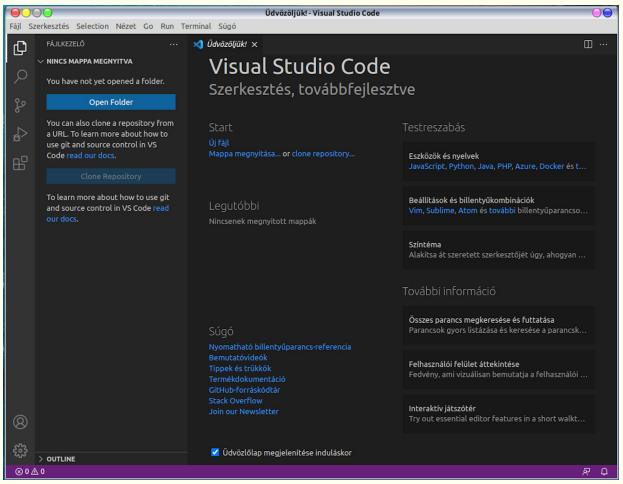
4.5. ábra: A PyCharm közvetlenül az új projekt sikeres létrehozása után, amennyiben semmilyen beállításon nem módosítottunk. A `main.py` állomány a leendő projekt feltételezett fő állománya, ahonnan a program végrehajtásának indulnia kell. Ebben az esetben erre az állományra nekünk most nem lesz szükségünk, de érdemes megjegyezni, hogy a több állományból álló programok általában mindig egy `main`.valami nevű állománnyal kezdődnek, ahol a valami helyén levő név mindig az éppen használt programozási nyelvtől függ. Ebben az állományban található a főprogram.

Mielőtt tovább ismerkednénk a használható fejlesztő környezetekkel, érdemes pár szót ejteni az azokban használt színekre. Ezúttal nem a programkód színezéséről van szó, hanem arról, hogy világos vagy sötét színösszeállítást használunk-e a programban.

Az itt bemutatott környezetek mindegyike eredetileg sötét háttért használ, de a dokumentumban az olvashatóság kedvéért a világos témára átállított programból vannak képek.

Az alapértelmezett sötét téma oka az, hogy a programozók sokat nézik ezeket a fejlesztő környezeteket a képernyőn. Márpedig minél világosabb képet nézünk sokáig, annál jobban fárasztja az agyunkat, annál jobban rongálja a szemünket. Ezért a fejlesztő környezeteketben egyre elterjedtebb az alapértelmezetten sötét színeket használó megjelenés. Általában legalább egy sötét és legalább egy világos témájú színösszeállítás közül lehet a beállításokban választani ezekben a programokban. Ha sokat használjuk, akkor azonban ergonómiai szempontból ajánlott a sötét témát választani.

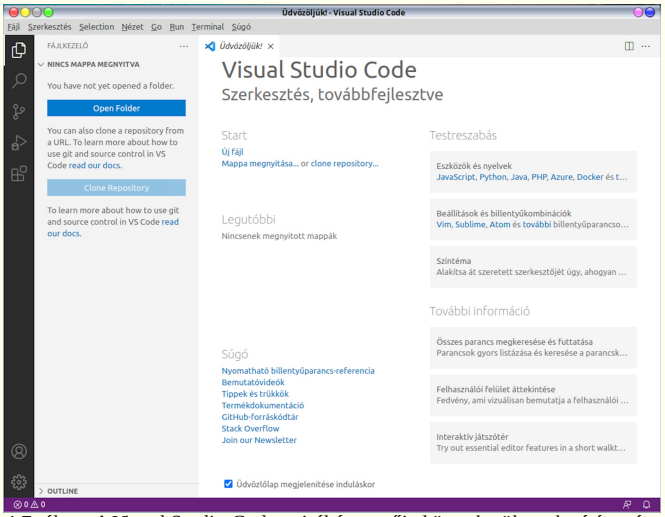
Egy másik hasznos fejlesztő környezet, amely szintén többféle programozási nyelvhez alkalmazható a Pythonon kívül, a Microsoft által fejlesztett *Visual Studio Code*, röviden: **VS Code**. Ez a program jó példa arra, hogy a Microsoft csak a Windows-t és az Office-t nem tudja felhasználóbarátra és probléma nélkül működőre fejleszteni: a VS Code egy kifejezetten sokoldalú fejlesztő környezet. Viszont sajnos nem olyan egyszerű használatba venni, mint a PyCharm-ot, mivel a telepítése után azonnal még semmilyen programozási nyelvet nem ismer. A program rengeteg kiegészítőjéből kell a megfelelőeket telepíteni ahhoz, hogy egy adott programozási nyelven történő fejlesztéshez használható legyen. Ám ezekkel a kiegészítőkkel nagyon kényelmes segítséget tud nyújtani a programfejlesztéshez.



4.6. ábra: A VS Code kezdőképernyője sötét témával

A 4.6. ábra az eredeti sötét színtémával, a 4.7. ábra pedig a világos témára váltás után mutatja a program nyitóképernyőjét. A színtémát a *File / Beállítások / Color Theme* menüponton keresztül lehet megváltoztatni.

A nyitóképernyő jobboldalán, az „*Eszközök és nyelvek*” részen belül található linkek (kék színű szöveg) vezet a leggyorsabban a nyelvtelepítési lehetőséghez. Aki már valamennyire ismeri a programot, az a bővítménytelepítőt a baloldali, a színtéma hatására sem változóan sötét hátterű sávon keresztül is elérheti. Egy adott nyelv használatához nagyon sok bővítmény érhető el, amelyek más-más segédeszközt kínálnak az adott programnyelven történő fejlesztéshez. Magának az egyes bővítmények leírásának átböngészése is rendkívül sok időt tud igénybe venni, emiatt a program használatba vétele bizony elég nehéz, ám a későbbiekben nagyon megkönnyítheti a programfejlesztést.



4.7. ábra: A Visual Studio Code nyitóképernyője közvetlenül a telepítés után, világos témára váltva. A használatba vételhez előbb Customize részen belül a „Tools and Languages” résznél a használni kívánt nyelv (Python) nevére kattintva kell elmenni arra a részre, ahol a megfelelő kiegészítők telepíthetőek.

Meg kell még említeni egy harmadik eszközt is, amivel Python nyelvű programokat lehet írni: A Google által kifejlesztett online eszközt, a Collaboratory nevűt. Ez egy olyan dokumentum, amelyben a szöveges blokkok mellett programkódot tartalmazó blokkok szerepelhetnek, amely blokkban a használható programozási nyelv a Python.

Bár az eszköz (még) nincs teljesen integrálva a Google többi felhőalapú szolgáltatásával – így a Classroom is furcsán teszi vagy nem teszi elérhetővé –, a létrejött dokumentumokat a Drive egy speciális könyvtárába teszi, megosztható, akár Classroom feladatba is beágyazható leadandó megoldásként. Így ha végképp nincs más mód a Python kód futtatására, akkor ez is megoldás lehet, hiszen csak egy böngésző és internetkapcsolat kell hozzá. Létezik Androidra alkalmazás is hozzá, így mobilon vagy inkább tableten is lehet vele dolgozni.

Arra érdemes figyelni, hogy az egyes kódblokkok között például a változók elvesznek. Az állománykezelést a hagyományos Python-tól eltérően kell kezelni: a Drive-on fent levő állományokat lehet elérni, nem a helyi meghajtón levőket.

Az előző oldalakon bemutatott eszközök a következő helyeken találhatóak meg az interneten:

A PyCharm telepítője letölthető (Windowsra, Linuxra, Macintoshra egyaránt):

https://www.jetbrains.com/pycharm/download/

Vigyázz, a jobboldali Communty verziót kell letölteni, mert a baloldali gombon elérhető változatért fizetni kell! A linuxos telepítő valójában nem telepítő, hanem ki kell csomagolni valahova és azonnal futtatható (hasznos hozzá némi Linux parancssor ismerete), ezért is van, hogy az iskolai linux környezetben a start menüben nem található meg.

A VS Code letöltése szintén platformfüggetlenül lehetséges az alábbi helyről:

https://code.visualstudio.com/download

A Google Collaboratory esetén nincs mit letölteni (legfeljebb a Google Chrome egy változatát). Az eszköz használatához a böngészőben be kell jelentkezni abba a fiókba, amelyben használni kívánjuk (iskolai feladathoz természetesen az iskolai fiókba), majd az alábbi címen elérjük a szolgáltatást:

https://colab.research.google.com/

Ezen a címen létrehozott bármely dokumentum ugyanúgy kezelhető, mint bármely más Drive dokumentum (a letöltés is lehetséges).

Az androidos alkalmazás hozzá az alábbi címen és az alatta levő QR-kódon keresztül érhető el.

https://play.google.com/store/apps/details?id=com.WeDevelopinPk.colabandroidwebview