Éveleji tanóratámogató segédanyag – IX. osztály

Informatika

9

magyar nyelven
a magyar tannyelvű iskolák/
osztályok számára,
9. osztály,
szakosított tanterv, elméleti
tagozat, reál szakirány,
matematika-informatika profil

SIGMA

1. Fejezet – Problémamegoldási stratégiák. A programozás alapjai
(1.3.; 1.4.; 2.1.; 2.3.; 2.4.; 3.3.; 3.4.; 4.1.; 4.3.; 4.4.; 5.3.; 6.3.)

1. Lecke *– Egy program elkészítésének szakaszai: az elemzéstől és tervezéstől a tesztelésig és hibakeresésig* 3
Feladatlap 1 5
2. Lecke *– Algoritmusok ábrázolása: folyamatábrák, pszeudokód, programozási nyelvek* 6
Feladatlap 2 11
3. Lecke *– Egy Python-program felépítése. Változók, adattípusok és kifejezések* 13
Feladatlap 3 16
4. Lecke *– Vezérlő szerkezetek: szekvenciális (lineáris), elágazó és ismétlő szerkezet* 17
Feladatlap 4 19

Útmutatók és megoldások 20

1. Fejezet – Problémamegoldási stratégiák. A programozás alapjai

1. lecke – Egy program elkészítésének szakaszai: az elemzéstől és tervezéstől a tesztelésig és hibakeresésig

*„Nem félek a számítógépektől. A hiányuktól félek” – **Isaac Asimov***

Informatika. Az adatok és az algoritmus

<def/> Az **informatika** olyan tudományterületek összessége, amelyek az információ számítógépes feldolgozásával foglalkoznak, és alapvető fontosságúak a *számítógépes gondolkodás* fejlesztésében.

Megj. Az *informatikai gondolkodás* az ember azon képessége, amelynek segítségével utasítássorozatokkal és ***algoritmusokkal*** megvalósított problémamegoldási módszereket dolgoz ki.

Az **informatikai gondolkodás fő jellemzői** a következők: a ***felbontás*** (az összetett problémákat egyszerűbb, könnyebben megoldható részproblémákra bontjuk), az ***általánosítás*** (azonosítjuk a hasonló mintákat vagy megoldásokat, hogy az új problémákat gyorsabban meg tudjuk oldani a már ismert megoldások felhasználásával), az ***absztrakció*** (csökkentjük a fölösleges részleteket), az **algoritmizálás** (kidolgozzuk a probléma megoldásának lépéseit) és az ***értékelés*** (ellenőrizzük, hogy a javasolt megoldás helyes és hatékony-e).

<def/> Az **algoritmus** véges számú, meghatározott sorrendben végrehajtott lépésből álló műveletsor, amely *adatokat* dolgoz fel. Az adatokat ***nevük***, ***típusuk*** és ***értékük*** azonosítja. Az algoritmusok problémák megoldására használhatók a matematika, a fizika, a kémia területén, de a mindennapi életben is.

Megj. Az **adatokat** osztályozhatjuk: **típusuk szerint** (*elemi adatok*: numerikus, logikai, karakterlánc típusú adatok, illetve strukturált adatok), **felhasználásuk időpontja szerint** (*bemeneti adatok*, *közbenső/segédadatok és kimeneti adatok*), **értékük szerint** (*változó adatok és konstans adatok*).

<def/> A számítógépek nem képesek közvetlenül értelmezni az emberi nyelvet, ezért az **adatok** digitális (bináris kód – **0** és **1**) formában vannak ábrázolva. A számítógép tehát gépi kódból álló nyelvvel működik.

Megj. A programozási nyelvek az *assembly* nyelvtől a *magas szintű és objektumorientált nyelvekig* fejlődtek, és ma már a mesterséges intelligenciában használt nyelvekig jutottak el. Segítségükkel az algoritmusok átírhatók programokra, amelyek egy adott problémacsoportot hatékonyan oldanak meg.

Egy program kidolgozásának szakaszai

Egy algoritmust megvalósító program elkészítéséhez a következő lépéseket kell végigjárni:

- **A probléma elemzése:** a bemeneti és kimeneti adatok, valamint a közöttük fennálló kapcsolatok azonosítása.
- **A megoldási mód kidolgozása vagy moduláris tervezés:** a problémát részproblémákra bontjuk, kiválasztjuk a megoldási módszert, és leírjuk a végrehajtandó lépések sorrendjét.
- **A megoldó algoritmus implementálása egy programozási nyelven:** a probléma megoldására szolgáló algoritmust egy programozási nyelven valósítjuk meg, annak szintaktikai szabályait betartva (például Pythonban).
- **A program tesztelése és hibakeresés (debug):** a program futtatásával azonosítjuk és kijavítjuk a szintaktikai vagy logikai hibákat.
- **A komplexitás elemzése és a program optimalizálása**: az algoritmus hatékonyságának értékelését jelenti a felhasznált memória és a végrehajtási idő szempontjából.

• Támpontok ezen szakaszok alkalmazásához problémamegoldás során

1. Elemzés
- a bemeneti és kimeneti adatok azonosítása;
- a feladat követelményeinek megértése;
- a feltételek és korlátozások meghatározása.

2. Tervezés
- a megfelelő adatszerkezetek kiválasztása;
- az algoritmus vázlatának kidolgozása (pszeudokódban vagy folyamatábrában);
- a probléma részproblémákra bontása (moduláris tervezés).

3. Megvalósítás
- az algoritmus átírása programozási nyelvre;
- a szintaktikai szabályok betartása;
- a programkód világos és jól olvasható szerkesztése.

4. Tesztelés
- a program ellenőrzése különböző bemeneti adatokra (normál esetek, határesetek, hibás adatok);
- a kapott eredmények összehasonlítása a várt eredményekkel;
- a szintaktikai, logikai vagy futási hibák azonosítása.

5. Hibakeresés
- a tesztelés során azonosított hibák helyének meghatározása és kijavítása;
- a változó értékek nyomon követése;
- a program újraellenőrzése minden egyes javítás után.

Tesztek kidolgozásának szempontjai

– A **bemeneti adatok** érvényesítéséhez ellenőrizzük:
- hogy a megadott adatok megfelelnek-e az elvárt típusnak (egész szám, karakterlánc stb.);
- hogy az értékek a megengedett tartományba esnek-e (például az osztályzat 1 és 10 között van);
- mi történik határértékek esetén (minimumérték, maximumérték, nulla);
- mi történik érvénytelen adatok esetén (negatív értékek, üres karakterláncok, nem várt karakterek).

– Az **algoritmus működésének** ellenőrzéséhez:
- tesztelés szokásos adatokkal (általános eset);
- speciális esetek tesztelése (üres lista, egyetlen elem, azonos értékek);
- a kapott eredmények összehasonlítása a várt eredményekkel;
- vizsgáljuk meg, hogy a program ugyanazt az eredményt adja-e ismételt futtatások során ugyanazokra a bemeneti adatokra.

Támpontok a változó értékek nyomon követéséhez

- kiíratás utasítások használata a változók értékeinek megjelenítésére a program fontos pontjain;
- az algoritmus követése lépésről lépésre (kézzel papíron vagy debugger-rel);
- a hiba típusának azonosítása:
  - **szintaktikai** hiba – a kód nem felel meg a programozási nyelv szabályainak;
  - **logikai** hiba – a program lefut, de helytelen eredményt ad;
  - **futási** hiba – a program végrehajtás közben leáll (pl. nullával való osztás)
- a hiba kijavítása és a program újbóli tesztelése.

Pl. A **programkészítés szakaszainak** szemléltetésére a következő feladatot vizsgáljuk meg: *Mihálynak informatikából **n** darab jegye van. Határozzuk meg a tanuló jegyeinek átlagát, majd állapítsuk meg, hogy átment-e vagy sem informatikából.*

ELEMZÉS →
- BEMENETI ADATOK: AZ N TERMÉSZETES SZÁM, AMELY A JEGYEK SZÁMÁT JELENTI, VALAMINT AZ EGYES JEGYEK
- KIMENETI ADATOK: A JEGYEK ÁTLAGA, AMELY VALÓS TÍPUSÚ ADAT LESZ
- AZ ADATOK KÖZÖTTI KAPCSOLAT: EGY SEGÉDADATOT HASZNÁLUNK, AMELYBEN MEGHATÁROZZUK A JEGYEK ÖSSZEGÉT.

TERVEZÉS

- A FELADATOT MODULÁRIS MÓDON VALÓSÍTJUK MEG;
- BEOLVASSUK A JEGYEK SZÁMÁT, ELLENŐRIZZÜK, HOGY POZITÍV-E ÉS NEM HALAD MEG EGY ADOTT HATÁRÉRTÉKET;
- BEOLVASSUK A JEGYEKET, AMELYEKNEK 1 ÉS 10 KÖZÖTTI ÉRTÉKEKNEK KELL LENNIÜK;
- KISZÁMÍTJUK A JEGYEK ÖSSZEGÉT, MAJD AZ ÁTLAGOT, ÚGY, HOGY AZ ÖSSZEGET ELOSZTJUK A JEGYEK SZÁMÁVAL, ÜGYELVE ARRA, HOGY AZ ÖSSZEGET VALÓS TÍPUSÚ ADATTÁ ALAKÍTSUK;
- ELLENŐRIZZÜK, HOGY AZ ÁTLAG MEGFELEL-E AZ ÁTMENÉSI FELTÉTELNEK (≥ 5), ÉS KIÍRJUK AZ EREDMÉNYT: ÁTMENT / NEM MENT ÁT.

MEGVALÓSÍTÁS

- AZ ALGORITMUST ÁTÜLTETJÜK A TANULT PROGRAMOZÁSI NYELVRE (PYTHON, C/C++)

TESZTELÉS, HIBAKERESÉS

ESETLEGES SZINTAKTIKAI VAGY LOGIKAI HIBÁKAT.
(PÉLDÁUL: HA N = 3 ÉS A BEOLVASOTT JEGYEK: JEGY1 = 8, JEGY2 = 9, JEGY3 = 10, AKKOR AZ ÁTLAG = 9 LESZ, ÉS A PROGRAM AZ „ÁTMENT" MINŐSÍTÉST JELENÍTI MEG; HA N = 2 ÉS A BEOLVASOTT JEGYEK: JEGY1 = 4, JEGY2 = 3, AKKOR AZ ÁTLAG = 3,5 LESZ, ÉS A PROGRAM A „NEM MENT ÁT" MINŐSÍTÉST JELENÍTI MEG.)

OPTIMALIZÁLÁS

- AZ ALGORITMUS MEMÓRIAHASZNÁLAT SZEMPONTJÁBÓL HATÉKONY, MIVEL ÁLLANDÓ MENNYISÉGŰ MEMÓRIÁT HASZNÁL (O(1)), ÉS VÉGREHAJTÁSI IDEJE LINEÁRIS A JEGYEK SZÁMÁNAK FÜGGVÉNYÉBEN (O(N)).

Feladatlap 1

Hozzátok létre a *Feladatlap1* nevű mappát. Oldjátok meg a következő feladatokat, a helyes válaszokat pedig jegyezzétek le egy *valaszok.docx* nevű fájlba.

1. *Állapítsátok meg a következő állítások igazságértékét:*

a. Az informatikai gondolkodás fő jellemzői lehetnek a következők: felbontás, általánosítás, absztrakció, algoritmizálás és értékelés;

b. Az algoritmus lépések végtelen sorozata, amelyek segítségével sajátos feladatok oldhatók meg;

c. A kimeneti adatok azok, amelyeket a program futása során használnak fel;

d. Értékük szerint az adatok lehetnek változók vagy konstansok.

2. *Rendezzétek sorrendbe az alábbi algoritmus lépéseit két pohár tartalmának felcserélésére: az egyikben gyümölcslé, a másikban víz van.*

Töltsd meg a kék poharat gyümölcslével. ☐
Vegyél egy üres, lila színű poharat. ☐
Öntsd a gyümölcslevet a kék pohárból a lila pohárba. ☐
Töltsd meg a sárga poharat vízzel. ☐
Öntsd a gyümölcslevet a lila pohárból a sárga pohárba. ☐
Öntsd a vizet a sárga pohárból a kék pohárba. ☐

Önértékelés!

Töltsétek ki a válaszokat egy *onertekeles1.docx* nevű fájlban, olyan jelöléseket használva, mint: 1 – *igen*, 2 – *nem*, 3 – *nem vagyok biztos*. Mentsétek el a fájlt a *Feladatlap1* mappába.

	A lecke végén tudom...	*Igen*	*Nem*	*Nem biztos*
1.	mi az algoritmus.			
2.	mik az adatok.			
3.	melyek egy program elkészítésének lépései.			
4.	mi az algoritmus lépéseinek helyes végrehajtási sorrendje.			

5. Nehézségeim adódtak a következőkben: …

Ne felejtsétek el értékelni a viselkedéseteket és a tevékenységeteket az óra során!

A lecke végén így éreztem magam:

2. lecke – Algoritmusok ábrázolása: folyamatábrák, pszeudokód, programozási nyelvek

1. Folyamatábrák

<def/> **A folyamatábra** az algoritmusok grafikus ábrázolásának egyik módja.

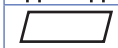
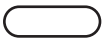
Az algoritmus vezérlő szerkezeteit **grafikus alakzatok** ábrázolják, amelyek az algoritmus lépéseit jelképezik, míg azok végrehajtási sorrendjét (a lépések egymásutánját) nyilak jelzik:

Emlékezzünk vissza!

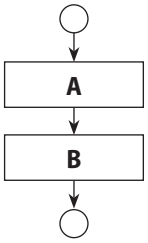
✓ Az ***algoritmus bemeneti adatokat*** dolgoz fel, és ***kimeneti adatokat*** szolgáltat. Néha szükség lehet ***segéd- (közbenső) adatokra*** is a feldolgozás során.

✓ A ***strukturált programozás*** alapelvei a modularizálás és az adatok strukturálása. Ez utóbbi három alapvető vezérlési szerkezetre épül: a szekvenciális (lineáris), az elágazó (alternatív) és az ismétlődő (ciklusos) szerkezetre.

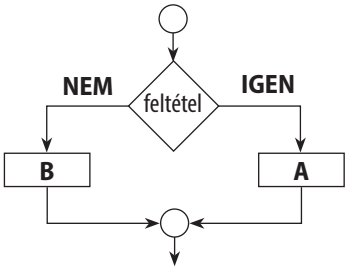
✓ Az ***algoritmusok ábrázolására többféle módszer létezik***: folyamatábrák (logikai sémák), *pszeudokód*, illetve *programozási nyelvek* segítségével.

	Folyamatvonal (nyíl): összekapcsolja a blokkokat, és jelzi a lépések végrehajtásának sorrendjét
	*Műveleti blokk:* különböző műveletek és értékadások végrehajtására szolgál.
	*Alprogramblokk:* egy alprogram meghívását jelöli, amelynek saját, külön folyamatábrája van.
	*Bemeneti/kimeneti blokk:* az adatok beolvasásának, illetve kiírásának műveletét jelöli.
	*Döntési blokk:* egy feltételt jelöl, két lehetséges kimenettel (Igen/Nem vagy Igaz/Hamis).
	*Kezdő/záró blokk (terminátor):* az algoritmus kezdetét (START) vagy végét (STOP) jelöli.
	*Összekötő blokk (konnektor):* a folyamatábra két vagy több pontjának összekapcsolására szolgál.

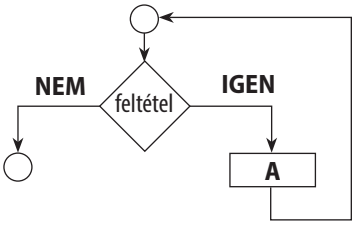
<def/> Bármely algoritmus leírható folyamatábrával a következő **vezérlőszerkezetek** felhasználásával:



lineáris szerkezet



alternatív szerkezet

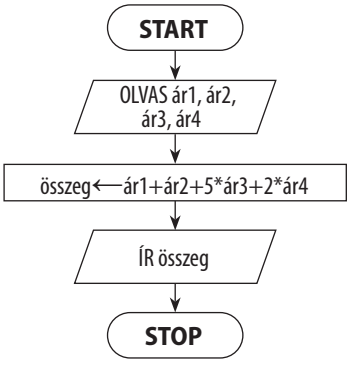


ismétlő szerkezet

Lineáris szerkezet

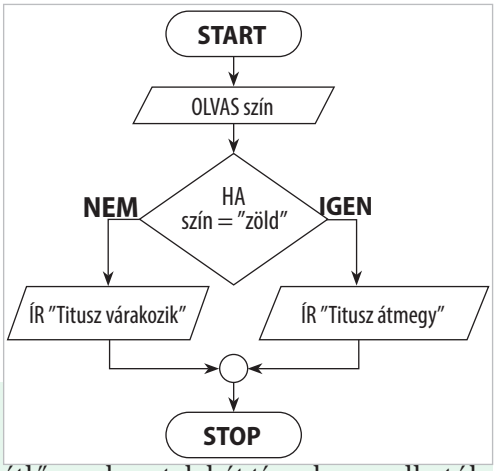
<def/> A **lineáris szerkezet** az utasítások adott sorrendben történő végrehajtását jelenti. Nem tartalmaz ismétléseket, minden utasítás pontosan egyszer hajtódik végre.

Pl. Ilonka szeretne vásárolni egy iskolatáskát, egy tolltartót, öt grafitceruzát és két töltőtollat. Ismerve az egyes termékek árát, határozzuk meg, hogy mekkora összegre van szüksége a tervezett vásárlásokhoz.



Alternatív szerkezet

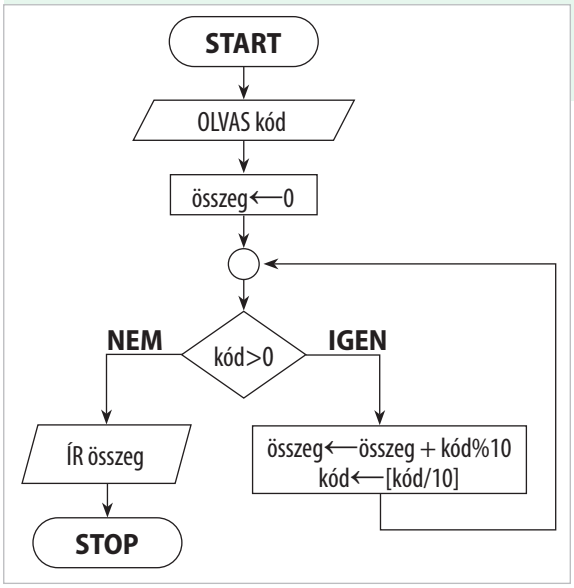
<def/> Az **alternatív szerkezet** lehetővé teszi, hogy a kifejezés értékétől függően kiválasszuk a végrehajtandó utasítást.



Pl. Titusz a közlekedési lámpánál áll, és szeretne átkelni a gyalogátkelőhelyen. A jelzőlámpa színétől függően eldönti, hogy mit tegyen. Ha a lámpa pirosat mutat, Titusz várakozik, ha pedig zöldet mutat, akkor átkelhet az úttesten. Segíts neki a helyes lépések követésében!

Ismétlő szerkezet

<def/> Az ismétlő (ciklusos) szerkezet egy utasítás vagy utasítássorozat többszöri végrehajtását végzi, egy feltételtől függően. Az ismétlő szerkezetek két típusba sorolhatók: *ismeretlen számú* ismétléssel működő ciklusok és előre *ismert számú* ismétléssel működő ciklusok.



Meaj. - Egy ismeretlen lépésszámú ismétlő szerkezet lehet **elöltesztelő** (a feltételvizsgálat a szerkezet elején van) vagy **hátultesztelő** (a feltételvizsgálat a szerkezet végén áll).

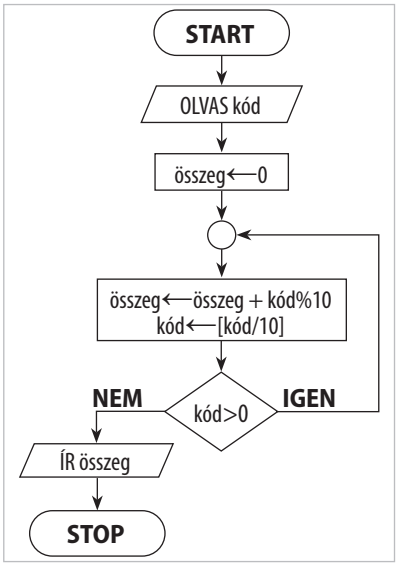
- Az **elöltesztelő** ismétlő szerkezetben a végrehajtás feltétele a ciklus elején, míg a **hátultesztelő** ismétlő szerkezetben a feltétel vizsgálata a ciklus végén történik.

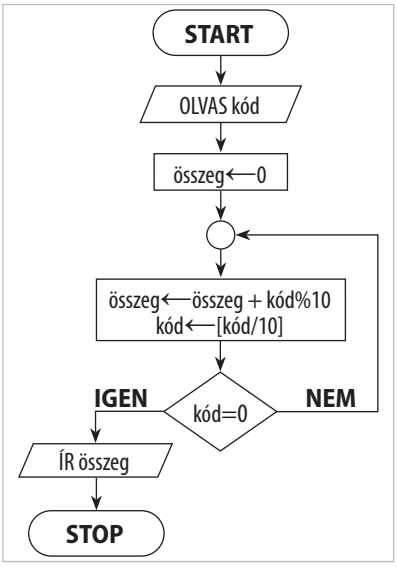
- Az előfeltételes és az utófeltételes ismétlő szerkezet közötti különbség az, hogy az utófeltételes szerkezetben az utasítások *legalább egyszer* végrehajtódnak, míg az előfeltételes szerkezetben csak akkor, ha a feltétel *igaz*.

Pl. Mirunának van egy doboza, amelyet egy 9 számjegyből álló kóddal ellátott lakat zár le. A doboz kinyitásához Mirunának meg kell határoznia a kód számjegyeinek összegét, majd ezt az összeget kell megadnia. Segíts Mirunának kinyitni a dobozt.

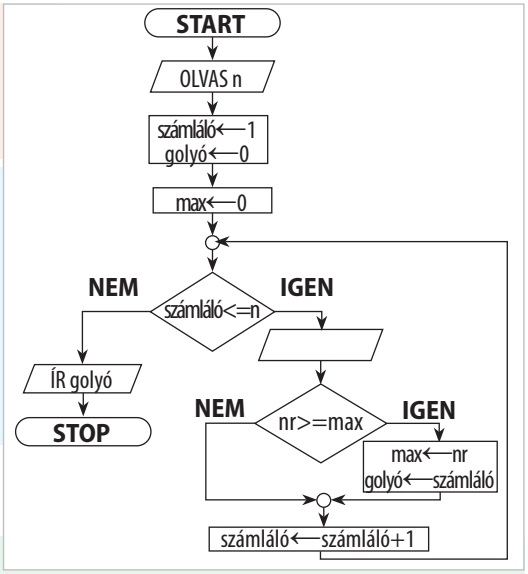
Meaj. A ***hátultesztelő ismétlő szerkezet*** kétféle formában fordulhat elő, attól függően, hogy a ciklus a feltétel teljesülése vagy a feltétel nem teljesülése esetén ismétlődik. Ezt a két változatot a mellékelt folyamatábrák mutatják be.

Meaj. Az ***ismert lépésszámú ismétlő szerkezetben*** a feltétel a szerkezet elején található, és akkor használjuk, amikor előre tudjuk, hogy az utasításokat hányszor kell megismételni.





Pl. Annának ***n*** darab doboza van. Minden dobozban el van rejtve egy-egy golyó, minden golyón egy természetes szám van. Írjunk algoritmust, amely segít Annának meghatározni, hogy melyik dobozban van az a golyó, amelyen a legnagyobb szám látható.



Megj. - Bármely ismert lépésszámú ismétlő szerkezet átalakítható ismeretlen lépésszámú ismétlő szerkezetté. ***Ennek a fordítottja nem mindig igaz!***

- Az algoritmusok folyamatábrákkal történő ábrázolása ***előnyös***, mert világosan szemlélteti a végrehajtandó lépéseket, és segíti a hibák azonosítását. Ez az ábrázolási mód azonban ***hátrányos*** lehet nagy méretű, összetett algoritmusok esetén.

2. Pszeudokód nyelv

<def/> ***A pszeudokód nyelv*** (a görög eredetű *pseudo* = hamis szóból) algoritmusok tervezésére szolgál. Meghatározott írási szabályokból és az azt alkotó utasítások sajátos szintaxisából áll. Az utasítások a leírásuk sorrendjében hajtódnak végre.

<def/> Bármely algoritmus leírható ***pszeudokód nyelven*** a következő vezérlőszerkezetek felhasználásával:

```
adat_tip változók
beolvas adat
utasítás 1
utasítás 2
.....
kiír adat
```

lineáris szerkezet

```
ha <feltétel> akkor
    <utasítások 1>
[különben <utasítások 2>]
ha vége
```

alternatív szerkezet

```
amíg <feltétel> végezd el
    <utasítások>
vége amíg
```

ismétlő szerkezet

Lineáris szerkezet

Pl. Mihály egy olyan utcán sétál, ahol 8 fa van. Minden fán egy tábla látható, amelyen fel van tüntetve a fa fajtája és magassága.

Mivel Mihály nagyon kíváncsi fiú, szeretné meghatározni a fák átlagos magasságát. Segíts neki úgy, hogy elkészíted a megoldás algoritmusát pszeudokód nyelven.

```
algoritmus átlag_magasság
valós h1, h2, h3, h4, h5, h6, h7, h8, atlag
beolvas h1, h2, h3, h4, h5, h6, h7, h8
átlag ← (h1+h2+h3+h4+h5+h6+h7+h8) / 8
kiír átlag
algoritmus vége
```

Alternatív szerkezet

Pl. Egy napon Anett az utcán sétált. Találkozott két kislánnyal, akik arra voltak kíváncsiak, hogy közülük melyikük idősebb. Anett megkérdezte a kislányokat, hogy hány évesek, majd megmondta nekik, melyikük az idősebb. Segíts Anettnek pszeudokód segítségével szemléltetni azokat a lépéseket, amelyek a rejtély megoldásához vezettek.

```
algoritmus ev
természetes x, y
beolvas x, y
ha x > y akkor kiír "Az első kislány nagyobb mint a második."
különben kiír "A második kislány nagyobb mint az első."
ha vége
algoritmus vége
```

Ismétlő szerkezet

Pl.

Példa az ismeretlen lépésszámú elöltesztelő ismétlő szerkezet alkalmazására: Anna egy zsákból természetes számokat tartalmazó cédulákat húz ki, addig, amíg a 0 számot nem húzza. Segíts Annának olyan algoritmust írni, amely meghatározza a kihúzott számsorozatban szereplő páros számok összegét.

```
algoritmus páros_számok
természetes x, s ← 0
beolvas x
amíg x > 0 végezd el
    ha x % 2 = 0 akkor s ← s + x
    vége ha
beolvas x
vége amíg
kiír s
algoritmus vége
```

Megj.

A **hátultesztelő ismétlő szerkezet** esetén kétféle ábrázolási forma létezik. A két forma közötti különbség a következő: az ***első*** esetben az utasítások addig ismétlődnek, ***amíg a feltétel igaz***, a ***második*** esetben az utasítások addig ismétlődnek, ***amíg a feltétel igazzá nem válik***.

```
algoritmus páros_szám
természetes x, s ← 0
ismételd
beolvas x
    ha x % 2 = 0 akkor s ← s + x
    vége ha
ameddig x = 0
kiír s
algoritmus vége
```

```
algoritmus páros_számok
természetes x, s ← 0
végezd
beolvas x
    ha x % 2 = 0 akkor s ← s + x
    vége ha
amíg x > 0
kiír s
algoritmus vége
```

Pl.

Példa az előre ismert számú ismétléssel működő ismétléses szerkezet alkalmazására: Freya egy nagyon okos, játékos kislány, aki szereti a számokat. Kisebb öccsével, Eliasszal játszik, és szeretné megtanítani őt kétjegyű számok összeadására. Ezért ***n*** darab kártyát mutat neki, amelyeken természetes számok szerepelnek, és arra kéri, hogy válassza ki azokat, amelyek pontosan kétjegyűek, majd határozza meg ezek összegét. A megoldás algoritmusát a mellékelt ábra szemlélteti.

```
algoritmus páros_számok
természetes n, i, s ← 0, nr
beolvas n
minden i ← 1, n végezd el
    beolvas nr
    ha nr >= 10 and nr <= 99 akkor
        s ← s + nr
    vége ha
vége minden
kiír s
algoritmus vége
```

Megj.

Az algoritmusok pszeudokód nyelven történő ábrázolása ***előnyös*** lehet, mivel könnyen érthető, szemléletes, és független a programozási nyelvektől. ***Hátrány***a viszont, hogy az algoritmus közvetlenül nem futtatható és nem tesztelhető, ezért később át kell ültetni egy programozási nyelvre.

3. Programozási nyelvek

<def/>

A **programozási nyelv** a programozó és a számítógép közötti kommunikáció eszköze. **Vokabulárium**ból (szókészletből) és az utasítások leírására szolgáló **szabályrendszer**ből áll. A programozási nyelvek segítségével az algoritmusok **programok** formájában valósíthatók meg.

Megj.

A C/C++ nyelven írt algoritmusok írásához szoftverfejlesztői környezetet (pl. CodeBlocks/MinGW) használunk, amely lehetővé teszi a forráskód ***szerkesztését***, ***fordítását***, a ***hibakeresést*** és a ***tesztelést***.

<def/>

A C/C++ nyelv **szókészlete** a következő elemekből épül fel: *karakterkészlet* (kis- és nagybetűk, számjegyek és speciális karakterek), *azonosítók*, *kulcsszavak*, *elválasztók* és *kommentek*.

<def/>

Az algoritmusok által kezelt ***adatoknak*** három jellemzője van: *név*, *típus* és *érték*. Egy ***adattípus*** meghatározza, hogy az illető adatnak milyen értékei lehetnek, valamint azt is, hogy miként tárolódik a memóriában. Az adattípusok két nagy csoportba sorolhatók: *elemi* és *strukturált* adattípusok. Az elemi adattípusok a következők: *egész numerikus típusok* (int, short, long, unsigned int, unsigned long, long long), *numerikus valós* (float, double), *karakter* (char, unsigned char), *logikai* (bool).

Az adatok két csoportra oszthatók: ***változókra*** (amelyek értéke változhat) és ***konstansokra*** (értékük nem változik).

A **kifejezés** egy vagy több **operandusból** áll, amelyeket **operátorok** választanak el egymástól.

Az operátorok alapján a kifejezések **aritmetikai** vagy **logikai** kifejezések lehetnek.

Emlékezzünk! Operátor típusok!

✓ Aritmetikai: unáris (+, -), bináris (+,-, *, /, %)
✓ Reláció: egyenlőtlenség (<, <=, >, >=), egyenlőség (==, !=)
✓ Inkrementálás/Dekrementálás: ++, --
✓ Logikai: !, &&, ||
✓ Értékadó: egyszerű (=), összetett (+=, -+, *=, /=, %=)
✓ Explicit konverzió: (típus) operandus
✓ Feltételes: e ? kifejezés1: kifejezés2

A C/C++ programozási nyelven írt **vezérlési szerkezetek** a következők:

Lineáris szerkezet	Alternatív szerkezet	Ismétlő szerkezet
int nr1, nr2, s; cin>>nr1>>nr2; s=nr1+nr2; cout<<s;	**if** (feltétel) utasítás 1; **else** utasítás 2;	**while**(feltétel) utasítás ;
Példa: Anett biológiából három jegyet kapott. Segíts neki kiszámítani a jegyeinek átlagát.	*Példa: János felír egy számot egy papírlapra. Ennek értékétől függően barátja, András megállapítja, hogy a szám páros vagy páratlan. Segíts Jánosnak olyan programot készíteni, amely kiírja a kívánt eredményt.*	*Példa: Tánja kap egy cetlit, amelyen egy többjegyű természetes szám és a következő feladat szerepel: „Határozd meg, hány páratlan számjegye van a számnak!”. Írjunk programot, amely meghatározza ezt az értéket, és ha a telefonszám egyetlen páratlan számjegyet sem tartalmaz, akkor írja ki a „Nem létezik” üzenetet.*
#include <iostream> **using namespace** std; **int** main() { **unsigned int** nt1, nt2, nt3; **double** media; cin>>nt1>>nt2>>nt3; media=(nt1+nt2 +nt3)/3.0; cout<<media; **return** 0; }	*#include <iostream>* **using namespace** std; **int** main() { **unsigned int** nr; cin>>nr; **if**(nr%2==0) cout<<"A szám páros"; **else** cout<<"A szám páratlan"; **return** 0; }	*#include <iostream>* **using namespace** std; **int** main() { **unsigned int** nr, k=0; cin>>nr; **while**(nr!=0) { **if**(nr%2==1) k++; nr/=10; } **if**(k!=0) cout<<k; **else** cout<<"Nem létezik"; **return** 0; }

Hátultesztelő ismétlő struktúra

A C/C++ nyelvben a **hátul-tesztelő ismétlő** szerkezet egyetlen formában létezik: a ciklus végén történő feltételvizsgálattal megvalósított ciklusként.

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int nr, k=0; cin>>nr;
    do
    {
        if(nr%2==1)
            k++;
        nr/=10;
    }while(nr!=0);
    if(k!=0)
        cout<<k;
    else
        cout<<"Nem létezik";
    return 0;
}
```

Ismert lépésszámú ismétlő struktúra

Példa: A testnevelő tanár szeretné megtudni, ki a legmagasabb tanuló az osztályban, valamint hogy hány tanuló van, aki vele azonos magasságú. Írjunk programot, amely elvégzi ezeket a műveleteket.

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int n, h, hmax=0, counter=0; cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>h;
        if(h>hmax)
        {
            hmax=h;
            counter=1;
        }
        else
            if(h==hmax)
                counter++;
    }
        cout<<hmax<<'    '<<counter;
        return 0;
}
```

Megj. Az algoritmusok programozási nyelven történő ábrázolása számos előnnyel jár. Például pontos, lehetővé teszi az algoritmus végrehajtását, hatékony és jól teljesít, támogatja a kód újrafelhasználását. Ugyanakkor hátrányt is jelenthet, mivel a programozási nyelvek szigorú szintaktikai szabályokat alkalmaznak, amelyeket pontosan be kell tartani.

Feladatlap 2

Hozzátok létre a *Feladatlap2* nevű mappát. Oldjátok meg a következő feladatokat, a helyes válaszokat pedig jegyezzétek le egy *valaszok.docx* nevű fájlba.

1. *Kövesd a mellékelt folyamatábrán látható algoritmust, majd oldd meg az alábbi feladatokat:*

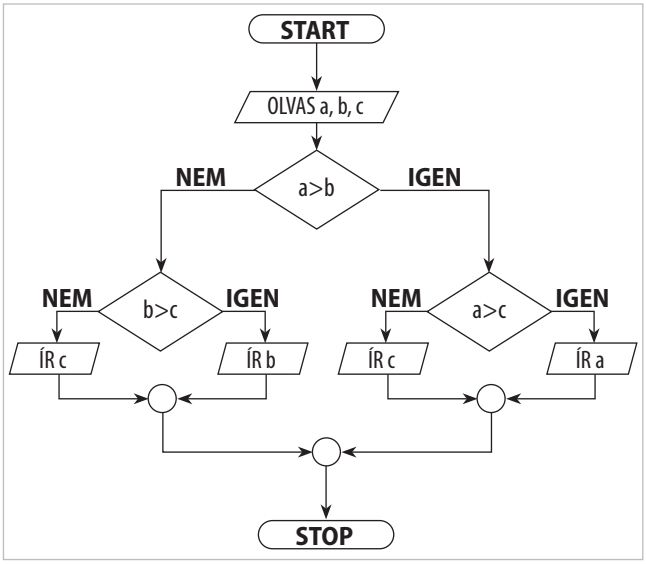
a. Állapítsd meg, hogy mit ír ki az algoritmus a 175, 100 és 23 értékek bevitele esetén. Mi lesz a kimenet a 2, 2 és 0 értékek bevitelekor?

b. Írd le ide, hogy mit végez ez az algoritmus:

..... .

c. Alakítsd át pszeudokóddá ezt a folyamatábrát.

d. Írd át az algoritmust C/C++ nyelvre.



2. ***Határozd meg az alábbi kijelentések logikai értékét:*** **a.** Az ismert lépésszámú ismétlő szerkezet átalakítható ismeretlen számú ismétléssel működő ismétlő szerkezetté. Ennek fordítottja is mindig igaz! **b.** A szekvenciális (lineáris) szerkezet lehetővé teszi az utasítások végrehajtását abban a sorrendben, amelyben le vannak írva. **c.** A hátultesztelő ismétlő szerkezetnek kétféle leírási módja van pszeudokódban is, és a C/C++ programozási nyelvben is. **d.** Egy vezérlési szerkezet lehet: szekvenciális, elágazó vagy ismétlő.

3. ***Rendezd helyes sorrendbe az alábbi utasításokat úgy, hogy a kapott algoritmus két egész szám legnagyobb közös osztójának meghatározását végezze.***

algoritmus vége
r ← a % b
beolvas a, b
egész a, b, r
algoritmus lnko
a ← b
r ← a % b
amíg r!=0 végezd el
b ← r
kiír b
vége amíg

4. ***Kövesd a mellékelt képen pszeudokód nyelven megadott algoritmust, majd oldd meg az alábbi feladatokat:*** **a.** Mit ír ki, ha a következő értékeket visszük be? 5, 13, 2, 0, 28, 29. **b.** Tudva, hogy n értéke 6, add meg azt az egy egymástól különböző számokból álló sorozatot, amelynek beolvasása esetén a program által kiírt érték 0 lesz. **c.** Írj pszeudokódban a megadottal egyenértékű algoritmust, amelyben a „minden ... végezd el” ciklust egy „amíg ... végezd el” (előfeltételes) ismétlő szerkezettel helyettesíted. **d.** Írd meg az algoritmusnak megfelelő C/C++ programot.

```
algoritmus feladat
egész n, nr, k, ok, i, d
beolvas n
k ← 0
minden i ← 1, n végezd el
    beolvas nr
    ok ← 1
    ha nr < 2 akkor ok ← 0
    vége ha
    ha nr % 2 = 0 && nr<> 2 akkor ok ← 0
    vége ha
    minden d ← 3, [nr/2], 2 végezd el
        ha nr % d = 0 akkor ok ← 0
        vége ha
    vége minden
    ha ok = 1 akkor k ← k + 1
    vége ha
vége minden
kiír k
algoritmus vége
```

5. ***Írd meg pszeudokódban és C/C++ nyelven a következő feladatot megoldó algoritmust:*** Beolvasunk egy n elemű egész számsorozatot. Minden beolvasott számra határozzuk meg és írassuk ki azt a prímtényezőt, amely a szám prímtényezős felbontásában a legnagyobb hatványon szerepel.

Önértékelés!

Töltsétek ki a válaszokat egy *onertekeles2.docx* nevű fájlban, olyan jelöléseket használva, mint: *1 – igen, 2 – nem, 3 – nem vagyok biztos benne*. Mentsétek el a fájlt a *Feladatlap2* mappába.

	A lecke végén tudom, hogyan lehet...	***Igen***	***Nem***	***Nem biztos***
1.	azonosítani a feladat megoldásához szükséges vezérlési szerkezeteket.			
2.	egy algoritmust folyamatábrával ábrázolni			
3.	egy algoritmust pszeudokóddal ábrázolni			
4.	egy algoritmust C/C++ nyelven ábrázolni			
5.	felismerni az algoritmusok különböző ábrázolási módjainak előnyeit és hátrányait.			

6. Nehézségeim voltak a következőkben: …

Ne felejtsétek el értékelni a viselkedéseteket és a tevékenységeteket az óra során!

A lecke végén így éreztem magam:

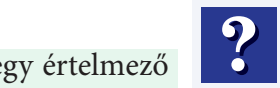
3. lecke – Egy Python-program felépítése. Változók, adattípusok és kifejezések

Egy Python-program felépítése

A **Python nyelv**ben írt program utasításait egy értelmező egymás után hajtja végre.

A fordítást igénylő nyelvekkel ellentétben a Python-programok futtatásához nincs szükség a program előzetes, teljes lefordítására.

- A **Python** használatához telepíteni kell a fejlesztői környezetet, a következő weboldalról: https://www.python.org/downloads/.
- Az IDLE mellett más fejlesztőkörnyezetek is használhatók:
  - *PyCharm* – korszerű fejlesztőkörnyezet Python-programok készítéséhez;
  - *Google Colab* – online környezet, amely lehetővé teszi programok írását és futtatását közvetlenül a böngészőben, telepítés nélkül.



Tudod-e?

✓ A Python nyelven írt programok hordozhatók

✓ A Python könnyen tanulható és használható, ingyenes, objektumorientált programozási nyelv

✓ A Python nyelv fejlesztését 1989-ben Guido van Rossum kezdte el

- A telepítés után az **IDLE** (**I**ntegrated **D**evelopment and **L**earning **E**nvironment) indításához kattints a Windows **Start** gombjára, majd keresd az „**IDLE**” alkalmazást. A konzolban (Interactive Python Shell) a parancsok a „>>>” karakterek után írhatók. Üzenet megjelenítésére a **print()** függvényt használjuk.

```
Python 3.14.4 (tags/v3.14.4:23116:
A11D64) J on win32
Enter "help" below or click "Help
>>> print ("Üdvözöllek benneteket!")
Üdvözöllek benneteket!
```

Új program létrehozásához válaszd a **File – New File** menüpontot (Ctrl+N). A megnyíló ablakban szerkeszthető a Python-kód.

```
print("Üdvözöllek benneteket!")
print("Ez az első programom!")
print("*****************")
```

A program mentéséhez a **File → Save As** (**Ctrl+Shift+S**) menüpontot kell választani, majd meg kell adni a mentés helyét és a fájl nevét. A Python-programok fájlkiterjesztése **.py**. A program futtatása a **Run → Run Module** (**F5**) paranccsal történik, a futás eredménye pedig a Python konzolban jelenik meg.

- A legegyszerűbb program a **print** függvény segítségével egy karaktersorozatot (szöveget) jelenít meg. Pythonban az utasítások pontosvesszővel (;) is elválaszthatók egymástól, de ennek használata nem kötelező. Az olvashatóság javítása érdekében ajánlott minden utasítást külön sorba írni. A Python megkülönbözteti a kis- és nagybetűket (case-sensitive).

A Python nyelv szókészlete

A Python nyelv **szókészlete** – más programozási nyelvekhez hasonlóan – a következő elemekből áll:

– A programok írásához használt *karakterkészlet*, amely magában foglalja az angol ábécé kis- és nagybetűit, a 0–9 számjegyeket, valamint a speciális karaktereket.

– *Azonosítók,* amelyek a változók, konstansok, függvények vagy modulok elnevezésére szolgálnak. Tartalmazhatnak betűket, számjegyeket és az „_” (aláhúzás) karaktert, de nem kezdődhetnek számjeggyel.

– *Kulcsszavak (reserved words)*, amelyeknek a nyelvben különleges jelentésük van (if, else, elif, for, true, while, false, continue, break, from, return, not, and, or).

– *Elválasztójelek,* amelyek az utasítások vagy a program elemeinek elkülönítésére szolgálnak, például: szóköz, behúzás (indentálás), zárójelek, „:”, „,” vagy „;”.

– *Megjegyzések (kommentek),* # karakterrel kezdve, a programkód részeinek magyarázatára szolgálnak.

Pythonban a behúzás (indentálás) különösen fontos, mivel az utasításblokkok határait jelöli, és a nyelv szintaxisának szerves részét képezi.

Pythonban egy változó **létrehozásához** elegendő nevet adni neki, majd értéket rendelni hozzá. A nyelv automatikusan meghatározza a változó adattípusát a hozzárendelt érték alapján, így nincs szükség az adattípus explicit megadására.

```
valtozo  = 23
print(valtozo )      #kiírja: 23
valtozo  = -55
print(valtozo )      #kiírja: -55
valtozo  = "egy sor"
print(valtozo )      #kiírja: egy sor
```

Az **input()** függvény üzenetet jeleníthet meg a konzolon, adatokat olvas be a billentyűzetről, és mindig egy karakterláncot (string) ad vissza. Az eredményt egy változóhoz rendelhetjük.

```
nev = input('Hogy hívnak? ')
kor = input('Hány éves vagy? ')
print(nev+','+kor+' éves vagy')
```

A **print()** függvény üzeneteket, változók értéit vagy ezek kombinációit jeleníti meg. A karakterláncok a + operátorral egyesíthetőek.

- Pythonban a változók értékei szöveges üzenetekkel együtt formázott karakterláncok segítségével írathatók ki. Erre használható a **format()** függvény, illetve az **f-string** szintaxis, amelynél az idézőjelek elé ***f*** vagy ***F*** kerül. Az *f-stringekben* a változókat és kifejezéseket *{ }* kapcsos zárójelek közé kell írni, a nyelv pedig automatikusan azok értékével helyettesíti őket.

```
nev = input('Hogy hívnak?')
kor = input('Hány éves vagy? ')
print(f'Szia {nev}. {kor} éves vagy.')
```

```
nev = input('Hogy hívnak? ')
kor = input('Hány éves vagy? ')
print('Szia {0}. {1} éves.'.format(nev,kor))
```

Adattípusok

A Python nyelvben a következő adattípusok léteznek: ***numerikus, logikai***, ***karakterláncok***, ***listák***, ***szótárak*** és ***tuple***-ök. A változók típusa a hozzárendelt érték alapján automatikusan meghatározódik, így az adattípust nem kell előre megadni.

```
nr1 = int(input("Első szám:"))
nr2 = int(input("Második szám:"))
print(nr1 + nr2)
print(nr1 - nr2)
print(nr1 * nr2)
```

```
a = int(input("a="))        # a=2
b = float(input("b="))      # b=3.5
print(a + b)                # 5.5
```

Megj. **Numerikus adattípusok:** az **int** (egész), a **float** (valós) és a **complex** (komplex szám). Az **int**(), **float**() és **complex**() függvények adattípus-konverzióra szolgálnak. Mivel az **input**() függvénnyel beolvasott adatok alapból szöveges típusúak, numerikus műveletek elvégzése előtt szükség esetén át kell alakítani őket a megfelelő numerikus adattípussá.

Ha egy kifejezésben egész és valós számok is szerepelnek, akkor az eredmény típusa valós szám lesz. A Python a valós számokat a szükséges pontossággal jeleníti meg.

```
lanc1 = str("Ez egy szöveg!")
print(lanc1)
lanc2 ="*********"
print(lanc2)
lanc3 = str('''Ez egy példa több sorba írt karakterláncra.''')
print(lanc3)
```

Megj. A változókhoz közvetlenül hozzárendelhető egy karakterlánc, amelyet apostrofok vagy idézőjelek közé kell tenni. A ***str()*** függvény használata opcionális – ez más adattípusokat alakít át karakterláncokká.

Többsoros karakterláncok esetén három aposztrófot vagy három idézőjelet használunk a szöveg határolására.

```
L1 = list()
L2 = list(('Matek', 'Info', 'Fizika'))
print(L1)     # kiírja: []
print(L2)     # kiírja: ['Matek', 'Info', 'Fizika']
```

A lista egy Python-adattípus, amely vesszővel elválasztott, [ ] közé tett elemek gyűjteményét tárolja. Az elemek indexelése ***0***-tól ***len(lista)*** - ***1***-ig történik, ahol a ***len()*** függvény a lista hosszát adja vissza. A listát közvetlenül a [ ] szimbólumokkal vagy a ***list()*** függvénnyel lehet létrehozni.

Pl.

```
szinek = ['piros', 'zold', 'kek']
print(szinek[0]) #piros
```

A listák rendezettek, véges számú elemet tartalmaznak, és módosíthatók. Az elemeknek nem kell azonos típusúaknak lenniük. A hozzáférés index alapján történik: ***0***-tól ***len-1***-ig előre, vagy -1-től visszafelé.

Kifejezések

<def/> A kifejezés operandusok és operátorok meghatározott sorrendben álló együttese, például: 12 - 3 * 2. Kiértékelésük során figyelembe kell venni az operátorok *prioritását* (precedenciáját), valamint az operátorok *asszociativitását* (balról jobbra és jobbról balra). Az azonos prioritású operátorok azonos asszociativitással rendelkeznek.

• Aritmetikai operátorok

<def/> Az aritmetikai operátorok két csoportba sorolhatók: **unáris operátorok**, ezek egyetlen operandussal dolgoznak (+, -, ~), és **bináris operátorok**, amelyek két operandus között végeznek műveletet (+, -, *, **, /, //, %). Az operandusok típusa lehet egész, valós vagy komplex.

```
a = 5
print(+ a)          #5
print(- a)          #-5
print(~ a)          #-6
b = 3
print(a + b)        #8
print(a - b)        #2
print(a * b)        #15
print(a / b)        #1.6666666666666667
print(a % b)        #2
print(a // b)       #1
c=2.25
print(a + c)        #7.25
print(a - c)        #2.75
print(a * c)        #11.25
print(a / c)        #2.2222222222222223
print(a % c)        #0.5
print(a // c)       #2.0
```

- A ~ operátor a bitenkénti negáció operátora.
- A ** operátor a hatványozás műveletét valósítja meg. Az operátor asszociativitása jobbról balra érvényesül, ezért a 2**3**2 kifejezés értéke 512, míg a (2**3)**2 kifejezésé 64.
- Az / operátor két egész vagy valós szám osztását végzi el. Az eredmény minden esetben valós (float) szám lesz.
- Az // operátor egész osztást végez. Az eredmény az osztás hányadosának lefelé kerekített egész része.
- A % operátor két szám osztásakor a maradékot határozza meg.
- A % és az // operátorokat gyakran használjuk valamely szám számjegyeinek feldolgozása esetén.

1.1. Szám utolsó számjegye - a % operátor

A % (modulo) művelet az osztás maradékát adja vissza. Mivel minden természetes számot **n= q×10 + r** formában lehet felírni, a **10**-zel való osztás maradéka pontosan az utolsó számjegy ($0 \le r \le 9$).

```
n = 712345 utolso = n % 10
# Utolso szamjegy
print('Utolso szamjegy:', utolso)   #5
# 4732=473×10+2; 4732 % 10 = 2
# 8900=890×10 + 0; 8900 % 10 = 0
```

1.2. Szám utolsó számjegyének törlése - az // operátor

A // (egész osztás) művelet az osztás egész részét adja vissza. Ha **10**-zel osztunk, az utolsó számjegyet töröljük: 4732 // 10 = 473, mivel 4732 = 473 × 10 + 2.

```
n = 712345
utolso = n % 10   # Utolsó számjegy
print('Utolso szamjegy:', utolso)   #5
n = n // 10   # Utolsó számjegy elhagyása
print('n utolsó számjegy nélkül:', n)
# 71234
```

1.3. Egy szám k-adik számjegye

Ha **n** és egy **k** pozíció (jobbról számolva, 0-tól kezdve) adott, akkor a **k** pozícióban lévő számjegy a következő képlettel számítható ki: **k_számjegy = (n //10^k) % 10**.

```
n = int(input())
k = int(input())
# Ha n és k azonos sorban vannak
# n, k = map(int, input().split())
szamjegy_k = (n // (10 ** k)) % 10
print(f'{k}-ik jobbrol: {szamjegy_k}')
# n = 4732, k = 2, 4732 // 100 = 47; 47 % 10 = 7
```

• Relációs (összehasonlító) operátorok

<def/> A relációs operátorok két operandus összehasonlítására szolgáló bináris operátorok: >, >=, <, <=, == és !=. Az összehasonlítás eredménye mindig egy logikai érték, amely True (igaz) vagy False (hamis). Ezt Boole-értéknek is nevezzük.

```
a, b, c = 5, 3, 5.0
print(a < b)     #False
print(a > c)     #False
print(a == c)    #True
print(b !=c)     #True
print(a >=b)     #True
```

• Logikai operátorok

<def/> A logikai operátorok: **not**, **and** és **or**. Az **and** és az **or** bináris operátorok, míg a **not** unáris operátor. Az operandusok lehetnek egész, valós, logikai stb. típusúak, az eredmény mindig logikai érték lesz.

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

not	True	False
	False	True

• Értékadó (hozzárendelő) operátorok

Pythonban az értékadás lehet egyszerű (=) vagy összetett (+=, -=, *=, /=, %=, //=, **=). Az értékadó operátorok jobbról balra asszociatívak, ezért először a jobb oldali kifejezés értéke kerül meghatározásra, majd ez az érték a bal oldalon szereplő változóhoz rendelődik.

Feladatlap 3

Hozzátok létre a *Feladatlap3* nevű mappát. Oldjátok meg a következő feladatokat, a helyes válaszokat pedig jegyezzétek le egy *valaszok.docx* nevű fájlba.

1. *Kövesd az alábbi Python-program működését és határozd meg, mit ír ki, ha a beolvasott szám 209.*
Magyarázd meg, mit végez ez a program.

```
szam = int(input("Kérek egy számot:"))
szam = szam // 10
b = szam % 10
szam = szam // 10
c = szam
ujszam = a * 100 + b * 10 + c
print("A létrejött szám: ", ujszam)
```

2. *Írd át a pszeudokód nyelven megadott algoritmust Python nyelvre. Teszteld a programot.*

```
algoritmus átlag_magasság
valós h1, h2, h3, h4, h5, h6, h7, h8, átlag
beolvas h1, h2, h3, h4, h5, h6, h7, h8
átlag ← (h1+h2+h3+h4+h5+h6+h7+h8) / 8
kiír átlag
algoritmus vége
```

3. *Oldd meg Pythonban a következő feladatokat:*

a. Olvassunk be két egész számot. Végezzük el velük az összeadás, kivonás, szorzás, osztás, egész osztás és maradékos osztás műveleteit.

b. Olvassunk be egy 4-jegyű egész számot. Határozzuk meg a számjegyeinek összegét.

c. Olvassunk be egy pontosan 2 számjegyű egész számot. Határozzuk meg az 1-től a beolvasott számig terjedő számok összegét és írjuk ki a beolvasott szám számjegyeinek négyzeteiből képzett szorzatot.

Önértékelés!

Töltsétek ki a válaszokat egy *onertekeles3.docx* nevű fájlban, olyan jelöléseket használva, mint: *1 – igen, 2 – nem, 3 – nem vagyok biztos*. Mentsétek el a fájlt a *Feladatlap3* mappába.

	A lecke végén képes vagyok...	*Igen*	*Nem*	*Nem tudom*
1.	létrehozni egy programot Python nyelven			
2.	azonosítani a Python-beli adattípusokat			
3.	deklarálni változókat			
4.	felsorolni az operátorokat és megmondani, mik a kifejezések			
5.	követni egy Python program működését			

6. Nehézségeim voltak a következőkben: …

Ne felejtsétek el értékelni a viselkedéseteket és a tevékenységeteket az óra során!

A lecke végén így éreztem magam:

4. lecke – Vezérlő szerkezetek: szekvenciális (lineáris), elágazó és ismétlő szerkezet

Szekvenciális (lineáris) szerkezet

<def/> A Pythonban a szekvenciális (lineáris) szerkezet az utasítások egymás utáni végrehajtását jelenti. Az utasítások végrehajtási sorrendje megegyezik a leírásuk sorrendjével, és minden utasítás pontosan egyszer hajtódik végre.

Pl. Írjatok programot, amely meghatározza egy kocka teljes felszínét és térfogatát, ha az élhosszát a konzolról olvassuk be.

```
#szekvenciális szerkezet
l = int(input("Élhossz: "))
felszin = 6 * l * l
terf = l * l * l
print(f"A kocka felszíne {felszin} és térfogata {terf}")
```

Elágazó / döntéshozó szerkezet

<def/> Az ***elágazó szerkezet*** egy logikai feltétel kiértékelésén alapul. A feltétel igazságértékétől függően a program az egyik vagy a másik utasításblokkot hajtja végre. Ha a feltétel igaz (True), akkor az *utasítások_1* blokk fut le, ellenkező esetben az *utasítások_2*. A Python nyelvben az else ág nem kötelező, ezért el is hagyható.

Szintaxis:
if *logikai_feltétel:*
    utasítások_1
else:
    utasítások_2

Pl. Készítsünk programot, amely beolvassa két gyerek életkorát (különböző életkorúak). Határozzuk meg és írja ki, hogy melyik gyermek idősebb.

```
kor1 = int(input("Az első gyerek kora: "))
kor2 = int(input("A második gyerek kora: "))
if kor1 > kor2:
    print("Az első gyerek nagyobb, mint a második")
else:
    print("A második gyerek nagyobb, mint az első")
```

Megj. Az elágazó szerkezet egy másik formája az **if ... elif ... else** szerkezet, amely több feltétel egymás utáni vizsgálatát teszi lehetővé. **Működése** a következő: először a ***logikai_feltétel1*** értékelődik ki, ha ez igaz, akkor az ***utasítások_1*** blokk hajtódik végre, ha hamis, akkor a program a következő (*elif*) ***logikai_feltétel2***-t vizsgálja. Amennyiben ez igaz, az ***utasítások_2*** blokk fut le, ellenkező esetben az else ágban szereplő ***utasítások_3*** blokk hajtódik végre.

Szintaxis:
if *logikai_feltétel1:*
    utasítások_1
elif *logikai_feltétel2:*
    utasítások_2
else:
    utasítások_3

Pl. Készítsünk programot, amely meghatározza a 2^n szám utolsó számjegyét, ahol n egész szám.

```
n = int(input("n="))
if n == 0:
    print(f"2^{n} utolsó számjegye 1")
elif n % 4 == 1:
    print(f"2^{n} utolsó számjegye 2")
elif n % 4 == 2:
    print(f"2^{n} utolsó számjegye 4")
elif n % 4 == 3:
    print(f"2^{n} utolsó számjegye 8")
else:
    print(f"2^{n} utolsó számjegye 6")
```

Megj. Amennyiben egy **if** szerkezetben több utasítás végrehajtására van szükség, az utasításokat egy közös blokkba kell foglalni. Pythonban a blokkhoz tartozó utasításokat behúzással (indentálással) jelöljük. A helyes behúzás a nyelv szintaxisának része, ezért annak hiánya értelmezési hibát eredményez. Az indentálás legegyszerűbben a Tab billentyű használatával végezhető el.

Az ismétlő szerkezet

<def/> Az ismeretlen lépésszámú ismétlő szerkezet esetén a program először kiértékel egy logikai feltételt. Ha a feltétel igaz, akkor végrehajtja a ciklus törzsében található utasításokat, majd ismét kiértékeli a feltételt. Ez a folyamat addig folytatódik, amíg a feltétel igaz. Amikor a feltétel hamissá válik, a ciklus végrehajtása befejeződik.

Szintaxis:
while *logikai_feltétel:*
    utasítások

Megj. - A Python nyelv csak az elöltesztelő ismétlő szerkezetet támogatja közvetlenül. Nincs egy kifejezett megvalósítása a hátultesztelő ismétlő szerkezetnek, ellentétben a C/C++ nyelvvel.

- Az elágazó szerkezethez hasonlóan az ismétléses szerkezet esetében is kiemelten fontos az utasítások helyes indentálása!

<def/> Az ***ismert lépésszámú ismétlő szerkezetet*** akkor használjuk, amikor egy lista, egy karakterlánc vagy más adatsor elemein szeretnénk végighaladni, és azokat a tárolás sorrendjében feldolgozni. Ennek a szerkezetnek a Python-beli szintaxisa a mellékelt ábrán látható.

Pl. Készítsünk programot, amely egy lista elemeinek bejárása során megszámolja a pozitív, a negatív és a nulla értékű elemek számát.

```
lista = list((0, -3, 12, 45, 3, 0, -5, 9))
poz = 0
neg = 0
nulla = 0
for nr in lista:
    if nr > 0:
        poz = poz + 1
    elif nr < 0:
        neg = neg + 1
    else:
        nulla = nulla + 1
print(f"{poz} pozitív, {neg} negatív és {nulla} nulla érték van.")
```

Megj. Az előző példában a lista elemeinek bejárására a for ciklust használtuk. Számsorozatok bejárására a Python a **range()** függvényt biztosítja. A **range(kezdő_érték, végérték, lépésköz)** függvény egy olyan sorozatot hoz létre, amely a kezdőértéktől indul, a végértéket nem tartalmazza, és a megadott lépésközzel halad. A kezdőérték és a lépésköz megadása nem kötelező. Alapértelmezés szerint a sorozat ***0***-tól indul, és a lépésköz 1.

Pl. Írjunk programot, amely meghatározza az ***1***-től ***n***-ig terjedő számok összegét, ahol ***n*** egy beolvasott érték.

```
n = int(input('n='))
s = 0
for nr in range(1, n+1):
    s = s + nr
print(f'A számok összege 1-től {n}-ig: {s}')
```

A break és continue utasítások

<def/> A **break** utasítást ismétlő szerkezetekben használjuk, ha azok végrehajtását kényszerített módon meg akarjuk szakítani.

Pl. Írjunk programot, amely számokat olvas be ***0*** végjelig, majd meghatározza a beolvasott számok összegét.

```
ossz = 0
while True:
    nr = int(input("nr="))
    ossz = ossz + nr
    if nr == 0:
        break
print(f"A számok összege  {ossz}")
```

<def/> A **continue** utasítás hatására a program kihagyja a ciklus törzsében hátralévő utasításokat és közvetlenül a következő iterációval folytatja a végrehajtást. A **continue** használata nem fejezi be a ciklust, csak az aktuális iterációt szakítja meg.

Pl. Írjunk programot, amely beolvassa az ***n*** értékét, majd kiírja az ***1*** és ***n*** közötti páratlan számokat.

```
n = int(input("n="))
for i in range(1, n + 1):
    if i % 2 == 1:
        print(i, end=" ")
    else:
        continue
```

[1] A print() függvény end paramétere határozza meg, hogy mi jelenik meg a megjelenített szám után – ahelyett, hogy új sorra lépne, a megadott karaktert (ebben az esetben egy szóközöt) helyezi el.

Feladatlap 4

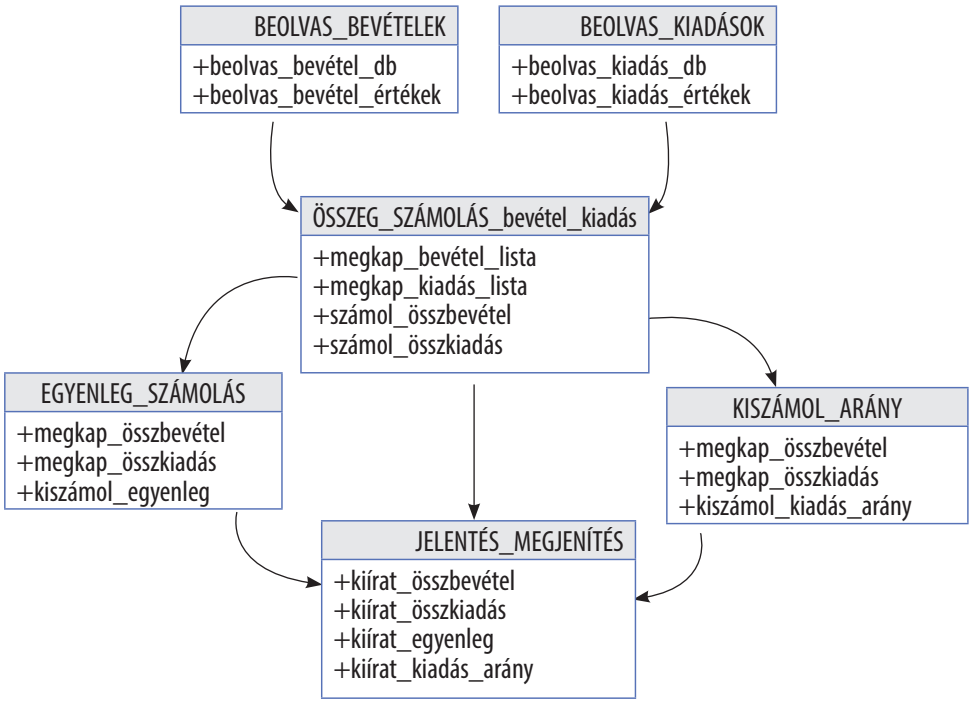
Hozzátok létre a *Feladatlap4* nevű mappát. Készítsétek el az alábbi Python-programot (alkalmazást), majd mentsétek el azt a *Feladatlap4* mappába.

Az alkalmazás egy személy havi költségvetését kezeli. A billentyűzetről a következő adatokat kell beolvasni:

- a bevételek száma és az egyes bevételek értéke;
- a kiadások száma és az egyes kiadások összege.

A program jelenítse meg:

- bevételek összegét;
- kiadások összegét;
- végső egyenleg (pozitív, negatív vagy nulla);
- a bevételekhez viszonyított kiadások arányát.



Önértékelés!

Töltsétek ki a válaszokat egy *onertekeles4.docx* nevű fájlban, olyan jelöléseket használva, mint: *1 – igen, 2 – nem, 3 – nem vagyok biztos*. Mentsétek el a fájlt a *Feladatlap4* mappába.

	A lecke végén képes vagyok...	*Igen*	*Nem*	*Nem tudom*
1.	használni a lineáris (szekvenciális) szerkezetet Pythonban			
2.	használni az elágazó szerkezetet Pythonban			
3.	használni az ismétlő szerkezetet Pythonban			
4.	alkalmazni a break és continue utasításokat			

5. Nehézségeim voltak a következőkben: …

Ne felejtsétek el értékelni a viselkedéseteket és a tevékenységeteket az óra során!

A lecke végén így éreztem magam:

ÚTMUTATÓK ÉS MEGOLDÁSOK

1. Fejezet

Feladatlap 1:

1. ***a)*** igaz; ***b)*** hamis; ***c)*** hamis; ***d)*** igaz.

2. 1 - (Töltsd meg a kék poharat gyümölcslével),
2 - (Töltsd meg a sárga poharat vízzel),
3 - (Vegyél egy üres lila poharat),
4 - (Öntsd a kék pohárból a lila pohárba a gyümölcslevet),
5 - (Öntsd a sárga pohárból a kék pohárba a vizet),
6 - (Öntsd a lila pohárból a sárga pohárba a gyümölcslevet)

Feladatlap 2:

1. ***a)*** 175 és 2; ***b)*** az algoritmus meghatározza a beolvasott 3 szám közül a legnagyobbat.

2. ***a)*** hamis; ***b)*** igaz; ***c)*** hamis; ***d)*** igaz.

Feladatlap 3:

1. Vegyük ki egy háromjegyű szám számjegyeit (az utolsót, a másodikat és az elsőt), majd alkossunk belőlük egy új számot, amelyben a számjegyek fordított sorrendben szerepelnek. Gyakorlatilag egy pontosan háromjegyű számot fordítunk meg.