Python ismeretek

Ez a fájl az érettségihez szükséges Python ismeretekről szól, tele példákkal és praktikákkal.

https://gyires.inf.unideb.hu/EFOP344/PythonHTML/index.html

https://www.cyberteen.info/python

https://www.w3schools.com/python/default.asp

1. Egyszerű adattípusok

A `type(változónév)` függvénnyel lekérdezhető egy változó típusa. Egyszerű adattípusok az egész szám (integer), a tizedestört (float), a karakter és a logikai (boolean) típusok.

Számoláshoz kétféle típust használunk: egész és tört (lebegőpontosnak is hívják).

Számtani műveletek:

- `a + b`, `a - b`, `a * b`, `a / b` a szokásos alapműveletek.
- `a**b` a hatványozás (a^b), pl `2**3` eredménye 8. Figyelj arra is, hogy a gyökvonást hatványozással is ki tudod számolni, pl. $\sqrt{8}$ kiszámítható $8^{0,5}$-ként: `8**0.5`
- `a//b` egész részes osztás (DIV-vel is jelölik), pl. `20//6` értéke 3
- `a%b` az osztás maradéka (MOD-dal is jelölik), pl. `20%6` értéke 2. Pl. egy számról így tudod eldönteni, hogy páros vagy páratlan: ha `szám%2` értéke 0, akkor páros és ha 1, akkor páratlan a `szám`.
- **Kerekítés**: `round(szám, hány számjegyre)`. Ha a 2. paraméter hiányzik, egészre kerekít. Pl. `round(2.456,1)` eredménye 2.5 és `round(5.678)` eredménye 6.
- **Konvertálás**: ha a szám szövegben (stringben) van, akkor számolás előtt az `int(szöveg)` vagy `float(szöveg)` függvénnyel számmá kell konvertálni.
- **max, min**: a legnagyobb, ill. a legkisebb, pl. `min(2,1,4)` értéke 1.

A **karakter** típus egy írásjel (szám, betű, felkiáltójel, aláhúzás stb.).

- A karaktereket is számokkal írjuk le, a kódolással mondja meg, hogy milyen szám milyen karaktert jelent (ezek a karakterkészletek). Kódolásból van sokféle, a legtöbbször használtak az ASCII (7 bit) és az UTF-8, illetve Unicode karakter készletet használunk.
- A Pythonban egy adott kódú karakter kiíratása a `char(kód)` függvénnyel történik, míg egy adott karakter kódját az `ord(karaker)` függvénnyel kérdezhetjük le. Mindkét függvény ismeri az ASCII és az Unicode kódolást.
- Speciális karakterek: `'\n'` új sor, `'\t'` tabulátor

A **logikai** típus: értéke lehet `True` vagy `False`. Főleg elágazásoknál, ciklusoknál van szerepe, feltételek eredménye logikai típus. Érdekesség, hogy pl. `if` után a nulla értékek hamisak, a nem-nulla értékek pedig igazak. Az üres lista, üres string is hamisat ad.

2. Összetett adattípusok

A Python számos olyan adattípust ismer, amit egyszerű adatokból lehet felépíteni. Az érettségi feladatok megoldása szempontjából a legfontosabbak:

- **LISTA (LIST):** egymás utáni adatok. Vannak 1 és több dimenziós tömbök, az indexek 0-tól indulnak. A Python a stringeket is listaként kezeli. A Pythonban a listára nem vonatkozik az a szigorú megkötés, hogy minden elem azonos típusú legyen, és több dimenzió esetén is lehetnek eltérő számú elemek.
  - **Létrehozás**: `lista = [1,'a',[2,3]]` de lehet így is:
    `paros_szamok = [i for i in xrange(1, 101) if i % 2 == 0]`

    Üres lista létrehozása:`lista = []`
    Lista másolása: `copy(lista)`
  - **Elem hozzáadása:**

    `lista.append(elem)` egy elemet hozzáfűz a lista végéhez

    `lista.extend(elem)` kibővíti a listát
  - **Elem törlése:** `lista.remove(elem)`
  - **Lista métere** (elemek száma): `len(lista)`
  - **Elemek elérése** – listaelemekre hivatkozhatunk ilyen formákban:

    `lista[i]` lista i-edik eleme, kezdőérték 0. Negatív i esetén visszafelé számol, vagyis `lista[-1]` az utolsó elemet jelenti.

    `lista[i:j]` s i-től j-ig vett része, szelete i indextől j indexig, de a j-edik elemet már nem tartalmazó, szelet ; i alapbeállítás szerinti értéke 0, j alapbeállítás szerinti értéke len(s)

    `lista[i:j:k]` s i-től j-ig vett szelete, k lépéssel.

    Például adott `lista = ['a', 'b', 'papa', 'z', 'mama']`. Ekkor a következők az értékek:

    `lista[1:3]` eredménye ['b', 'papa']

    `lista[1:-1]` eredménye ['b', 'papa', 'z']

    `lista[0:3]` eredménye ['a', 'b', 'papa']

    `lista[:3]` eredménye ['a', 'b', 'papa']

    `lista[3:]` eredménye `['z', 'mama']`, a harmadik indexűtől a végéig

    `lista[:]` eredménye ['a', 'b', papa', 'z', 'mama']
  - **Fontosabb listakezelő függvények:**

    `lista.count(elem)` megszámolja, hányszor fordul elő a listában az elem

    `lista.index(érték)` az első, értékkel egyenlő elem indexe

    `lista.sort(reverse=True|False, key=függvény)` rendezi a listát

    `lista.clear()` minden elemet kitöröl a listából

    `lista.insert(index,elem)` index pozícióra beszúrja elem-et

    `lista.pop(index)` az index pozícióról kitöröl egy elemet

`lista.reverse()` megfordítja a listát

- **Lista rendezése:** `lista.sort(key=függvény, reverse=True)`

  Példák: számlista növekvőbe rendezése: `lista.sort()`

  számlista csökkenpbe rendezése: `lista.sort(reverse=True)`

  szöveglista hossz szerinti rendezése: `lista.sort(key = len)`

- **Végiglépkedés**: vagy elemek szerint, vagy index szerint:

```
for elem in lista:
    print(elem)
for i1 in range(len(lista)):
    print(lista[i1])
```

- **Tartalmazás vizsgálat**: Az `in` operátorral lehet eldönteni, hogy egy elem benne van-e egy listában, pl.

```
if betu in ['e','i','o','ö','ü','ó','ő','ú','é','á','í']:
    print('a betű egy magánhangzó')
else:
    print('a betű egy mássalhangzó')
```

- **Több dimenziós listák, tömbök**: listák listája

  Példa több dimenziós tömbre: `szamok = [[1,3],[5,7],[9,11]]`. Itt pl. `szamok[2][0]` értéke 9.

  Például `lista = [['egy',1],['hét',7,'törpe']]`

  Itt `lista[1][1]` értéke 7, `lista[0][1]` értéke 1.

  Elemek elérése: több dimenziós listán végig lehet így menni:

```
for i in range(len(a)):
   for j in range(len(a[i])):
       print(a[i][j], end=' ')
   print()
```

  Több dimenziós lista rendezése: több lehetőség van.

  a) `lista.sort(key=lambda elem: elem[1])` pl. a 2. elemek szerint rendezi növekvőbe
  b) lista.sort(key=lambda elem: elem[0], reverse=True) az első elemek szerinti csökkenőbe rendezi
  c) `lista.sort(key=lambda e:(e[0],e[1])` a listát úgy rendezi, hogy az 1., azon belül a 2. elemek szerint növekvőbe
  d) `lista.sort(key=lambda e:(e[0],-e[1])` a listát úgy rendezi, hogy az 1. elemek szerint növekvőbe, azon belül a 2. elemek szerint csökkenőbe
  e) definiálhatunk egy rendező függvényt, pl. a következő minta rendezi a listát úgy, hogy az 1., majd a 2. szerint lesznek sorba rendezve.

```
def rendez(i):
```

```
        return i[0],i[1]
lista.sort(key=rendez)
```

Csökkenőbe így lehet rendezni: `lista.sort(key=rendez,reverse=True)`

- **HALMAZ (SET):** egy elem csak egyszer lehet benne, nem rendezett (nem lehet megmondni, melyik az első vagy utolsó elem és nem lehet indexxel hivatkozni egy-egy elemre) és az elemek nem változtathatók meg. Lehet viszont hozzárakni elemet, kiszedni belőle, tartalmazást vizsgálni, végiglépkedni az elemeken és a metszet és unió is megvalósítható beépített függvényekkel.
  Ha egy listából halmazt csinálunk, akkor azok lesznek benne, amik előfordulnak a listában.

  Pl. ha a `lista = [1, 1, 2, 2, 3, 4, 5, 5, 5, 6, 9]`, akkor a `halmaz=set(lista)` eredménye `{1, 2, 3, 4, 5, 6, 9}`. Ilyenkor a `2 in halmaz` eredménye `True` lesz.

  Új elem hozzáadása: `halmaz.add(új elem)`

  Egy elem törlése: `halmaz.remove(törlendő_elem)`

  Elemek elérése:
  ```
  for elem in halmaz:
      print(elem)
  ```

  Tartalmazás vizsgálat:
  ```
  if 1 in (1, 2, 3):
      print('Benne van!')
  ```

- **STRING (STR, SZÖVEG):** karakterből épül fel. A Pythonban a szövegek is listák, így ami listára működik, általában stringre is.

  Példák:
  ```
  szo = 'Egy egy string'
  if 'b' in szo:
      print('Van benne b betű')
  szo += ' újabb szó'
  ```

  String hossza: `len(szo)`

  String 5. karaktere: `szo[4]`, első 5 karakter: `szo[:5]`

  A szöveg megszabadítása a fölösleges karakterektől, pl. `szo.strip()` kiszedi az elejéről és a végéről a szóközöket, de használható `strip(amitől_meg_kell_szabadítani_a_szöveget)` alakban is.

  String feldarabolása: `szo.split(elválasztó)`, pl. egy mondat szavakra tördelése: `mondat.split(' ')`, eredménye egy **lista** a szavakból.

A többi típust nem részletezem, mert a fenti hárommal megoldhatók az érettségi feladatok. Azért kerülnek mégis ide, hogy ha látunk egy megoldást, amiben valaki ezt használja, ismerős legyen.

- tuple: nem megváltoztatható lista, pl. `primek=(2,3,5,7)` vagy `primek=2,3,5,7`
- szótár (dictionary): kulcs:érték párosokból áll. Előnye, hogy ezek használatával sokkal gyorsabb a program megvalósítása (lefutása). A kulcsok mindig azonos típusból kell kikerüljenek. Példa:
  ```
  csalad = {'apa': Rómeó', 'anya':'Júlia','gyerekek_szama':0}
  ```

- objektum (object): az objektumorientált programozás (OOP) rendkívül sokrétű lehetőségeket biztosít, különösen fontos, ha a programfejlesztés csapatban történik.
  Képzeld el, hogy olyan programot kell írnod, amiben 30 labda repked a képernyőn. A lényege az, hogy egy labdának (ez lesz az objektum) vannak tulajdonságai (színe, átmérője, x és y koordinátája) és valahogy viselkedik (mozog x irányba, mozog y irányba, ütközik, visszapattan valamiről). Ilyenkor leírnak egy labda osztályt (megadják, hogy általánosan milyen tulajdonságai legyenek, és hogyan viselkedjen – ezeket metódusoknak hívják), majd ez alapján labda egyedeket hoznak létre, amik már „élik” a saját életüket.
  Van még lehetőség örökölni is, pl. ha vannak robbanós labdák, akkor azok öröklik a sima labdák dolgait (tulajdonságok, viselkedés), de még kiegészülnek más tulajdonságokkal és viselkedéssel.

Logikai vizsgálatok

Vezérlési szerkezetek

Függvények

Legfontosabb beépített függvények, egymásba ágyazás

Saját függvények

Rekurzív függvények

3. Input és output műveletek

Adatbevitel (input)

Adatot a **billentyűzetről** az `input()` utasítással lehet bekérni, az eredmény string lesz:

```
szoveg = input('Add meg a születési évszámodat: ')
```

Ha a `szoveg` változóval számolni akarsz, **alakítsd egész számmá**:

```
szam = int(input('Add meg a születési évszámodat: '))
```

Fájlból az adatbevitel többféleképpen lehetséges: vagy tároljuk egy listában a fájl sorait, vagy (túl hosszú fájlok esetén) soronként beolvassul és úgy dolgozzuk fel. Ld. a Fájlkezelés részt.

Kiírás (output)

Az adatokat vagy képernyőre, vagy fájlba írjuk a `print()` függvénnyel.

Ha **fájlba** akarod írni, akkor nyiss meg egy fájlt írásra, és a `print()` utolsó eleme a `file = fájlazonosító` legyen, és ne felejtsd el le is zárni a fájlt. Példa:

```
fajl_id = open('eredmeny.txt','w')
print('Ez egy sor',file = fajl_id)
fajl_id.close()
```

A `print()` általános alakja:

```
print(valamik, sep='elválasztó karakter', end='végére írjuk ki ezt')
```

`sep` után azt kell megadni, hogy milyen karakterrel legyenek elválasztva a felsorolt dolgok, alapértelmezés szerint szóköz (ezt nem is kell megadni).

`end` után azt kell megadni, hogy milyen karakterrel zárja a kiírást, alapértelmezés szerint új sor ('\n').

Például `print('a',end='')` esetén a következő kiírást közvetlenül az 'a' karakter után kezdi, így megakadályozható a következő írásnál új sorba írás

Többféle alkalmazása van:

print('A keresett érték:',ertek)	kiírja egy szóközzel elválasztva a szöveget és az *ertek* változó értékét.
print(a,b,c)	kiírja egymás után szóközzel elválasztva *a*, *b* és *c* változók értékét
print('{}. utas: ' .format(utas_szam))	a {} helyére írja az *utas_szam* értékét
print("{:08.3f}".format(12.2346))	eredménye: 0012.235
print("{:5d}".format(12))	eredménye: 00012
print("{:8.3f}".format(12.2346))	eredménye: ' 12.235'
print("{:05d}".format(12))	eredménye: 00012

Ha a szám után kell ragasztani valamit, pl. 5. vagy 26%, akkor stringgé alakítjuk a számot, pl.

print(str(helyezes)+'. helyezés')

Százalék kiíratása (pl. a 2 a 7-nek hány százaléka, 2 számjegyre kerekítve):

print(str(round(2/7*100,2))+'%')

Ha az a feladat, hogy egy **stringkből** álló listát írjál ki pl. vesszővel elválasztva, használd a `join()` függvényt. Ha a listád számokból áll, akkor érdemes belőle egy stringes listát csinálni. A `join()` a végéről automatikusan lehagyja az elválasztót. Példa:

szamok = [1,2,3,4,5]

str_szamok = [str(szam) for szam in szamok]

print('Számok:',', '.join(str_szamok))

A kiírás eredménye: Számok: 1, 2, 3, 4, 5

A `print` további alkalmazásaira találsz példát a következő honlapokon:

- https://www.programiz.com/python-programming/methods/string/format
- http://people.ubuntu.com/~kelemeng/.ufp3/strings.html#formatting-strings
- https://realpython.com/python-string-formatting/

4. Adattípusok, adatszerkezetek

Feladatok

a) Írasd ki a következő értékek típusát: ’anya’ 5 3.14 [1,2,3]

Műveletek - Python Operátorok

Összehasonlítás

Az összehasonlító operátorok használata tetszőleges típusú objektumok között lehetséges. Az operátorok precedencia értéke (műveleti hierarchia, sorrend) azonos. **Pythonban az összehasonlításokat akár láncba is fűzhetjük, tehát a x < y <= z kifejezés ekvivalens azzal, hogy x < y and y <= z,** elég lusta kiértékelés mellett.

A különböző típusú objektumok soha nem lehetnek egyenlők összehasonlításkor, kivéve a numerikus típusúakat.

Az összehasonlító operátorok a következők:

< szigorúan kisebb mint

<= kisebb vagy egyenlõ

> szigorúan nagyobb mint

>= nagyobb vagy egyenlõ

== egyenlõ

!= nem egyenlõ ("<>" is megengedett)

is objektumok azonossága (objektumok azonosak-e, nem értékek)

is not tagadott objektumazonosság

Tagságot vizsgáló operátorok

x in y az eredmény 1 (True), ha x tagja az y sorozatnak, pl 2 in [1,2,3] igaz eredményt ad

x not in y az eredmény 1, ha x tagja az y sorozatnak

Minden numerikus típusra vonatkozó műveletek:

Alapműveletek: x + y, x – y, x * y, x / y

Hatványozás (x^y): x ** y

x // y x / y egész része, pl. 30 // 7 = 4 (lefelé kerekített integer típus; DIV)

x % y x / y maradéka (MOD), pl. 30 % 7 = 2

divmod(x, y) eredménye (x/y, x%y) tuple típusú változó

pow(x, y) x y-adik hatványa (x^y)

abs(x) x abszolút értéke

int(x) x értékét integer típusra konvertáljuk

long(x) x értékét long integer típusra konvertáljuk

float(x) x értékét lebegõpontos számmá konvertáljuk

-x x értékét -1-gyel megszorozzuk, ill. negáljuk (pl True-ból -1-et csinál, False-ból 0-t))

Kerekítés: round(szám, hány számjegy), pl. round(12.3456,2) eredménye 12.35

str() string-é konvertál, de a repr()-el ellentétben ez jobban olvasható,
formázott szöveges formát ad visssza.

sqrt(szám) négyzetgyök, de kell hozzá a from math import sqrt

Feladatok:

a) Kérd be n és k pozitív egész számokat, és írasd ki n DIV k, n MOD k és n^k értékét a megfelelő operátorokkal!

b) Kérd be n és k pozitív egész számokat, és írasd ki n DIV k és n MOD k értékét úgy, hogy nem használod a megfelelő operátorokat, csak az alapműveleteket és az int() függvényt!

Minden szekvencia típusú változóra (lista, tuple, sztring) alkalmazható operátorok:

Függvények: Hatásuk:

len(s) s hossza

min(s) legkisebb (string esetén ASCII értékű) elem s-ben

max(s) legnagyobb értékű elem s-ben

x in s igaz, ha ha x egy elme az s szekvenciának, különben hamis

x not in s 0 ha s egy elme azonos x-szel, különben 1

s + t s és t konkatenációja (összefűzése), pl. 'a'+'b' eredménye 'ab'

s * n s-t n-szer összefűzi, pl '%'*5 eredménye '%%%%%'

s[i] s i-edik eleme (kezdőérték 0), pl. [1,2][0] eredménye 1, mert a 0. indexű elem az 1

s[i:j] s i-től j-ig vett része, szelete i indextől j indexig, de a j-edik elemet már nem tartalmazó, szelet

i alapbeállítás szerinti értéke 0, j alapbeállítás szerinti értéke len(s)

negatív értéket a szekvencia jobb végétől számítjuk, vagyis s[-1] az utolsó elem

Pl. ha l = [1, 2, 3, 4, 5], akkor l[2:] eredménye [3, 4, 5], l[-2] eredménye 4, l[1:3] eredménye [2, 3]

s[i:j:k] s i-től j-ig vett szelete, k lépéssel.

repr() string-é konvertál, a tárolt érték tényleges karakteres formáját adja vissza

Logikai operátorok

not x x logikai értékét ellentettjére változtatja: igazból hamisat csinál és fordítva

x and y akkor igaz, ha x és y értéke is igaz

x or y akkor igaz, ha x vagy y közül legalább az egyik igaz

5. Alapvető programozási tételek

Programozáskor felmerülő problémák nagy része visszavezethető néhány alapproblémára, és ezekre létezik megoldási sablon. Az ilyen, egy-egy feladattípus megoldását nyújtó algoritmusokat nevezzük **programozási tételnek**.[1]

[1] A jegyzet egyik forrása a programozási tételek tárgyalása során többnyire a Szlávi-Zsakó párosnak a nevezéktanát, mintapéldáit követi.

A programozási tételek első csoportja az **elemi programozási tételek**, amelyekben egy értéket rendelünk egy sorozathoz. Ezek (vizsgakövetelmények): összegzés, eldöntés, kiválasztás, maximum-kiválasztás, megszámlálás, keresés, kiválogatás, elemi rendezések.

A következő dokumentum végigveszi ezeket, és mondatszerű leírást is ad rájuk:
http://szatyika.hu/files/programozasi-tetelek.pdf

Összegzés

- sum(lista, start) Összeadja a lista elemeit, és még start értékét hozzáadja.

  De végiglépkedhetünk indexxel is a lista elemein, és összeadhatjuk őket.

Összegzéskor használhatod a sum függvényt, például: `print(sum(számlista))`

- Átlagszámításkor működik a sum(lista)/len(lista) páros, például: print(sum(számlista)/len(számlista))
- Ezek is működnek: `eredmeny += elem`, `eredmeny -= elem` ..., pl. ha `a` értéke 3, akkor az `a += 1` eredményeként `a` értéke 4-re változik.
- Stringek egymás után írására a legjobb az összeadást használni: `string += karakter`

Az összegzés általánosan nem egyéb, mint egy gyűjteményes adattípus – például a lista, vagy egy range-objektum – elemeinek összeadása. Ez a pszeudokód a sorozatszámítás programozási tétele:

eredmeny = kezdőérték

ciklus a sorozat minden elem-ére

  eredmeny megváltoztatása elem-mel

ciklus vége

Az eredmény **kezdőértékének** megválasztására figyelni kell: összegnél 0, szorzatnál 1, betűk egymás után írásakor üres sztring, pl. `uj_szoveg = ''`

A "megváltoztatva" szó a pszeudokódban arra utal, hogy más és más műveletet végzünk, ha összeget vagy ha szorzatot keresünk.

Feladatok

a) Írj programot, amely bekér pár osztályzatot, eltárolja őket egy listában, majd kiírja őket egy sorban, vesszővel elválasztva, és új sorokba kiírja az összegüket, szorzatukat, átlagukat! Figyelj az intelligens kiírásra, azaz minden érték előtt írasd ki, hogy mi is az az érték!
b) Egy listában tároljuk, hogy a focicsapatunk az egyes meccsein nyert, vesztett, vagy döntetlent játszott: ['ny', 'ny', 'v', 'd', 'd', 'd', 'v', 'v', 'ny', 'ny', 'd']. Hány pontja van a bajnokságban? A győzelemért három, a döntetlenért egy pont jár.
c) Készíts egy 20 elemű listát 1 és 100 közötti egészszámokból! Készíts ezekből olyan listát, amelyikben az egymást követő számokból álló párokat (értsd: 1. és 2., 3. és 4. ...) összeadod, illetve egy másikat, amiben összeszorzod!

Eldöntés

Ha az eldöntés csak arról szól, hogy megvan-e egy elem egy gyűjteményes adatszerkezetben, akkor használható az `if keresett_elem in adatszerkezet` utasításforma is, például

```
if 'oroszlán' in allatok_listaja:
```

Az `in` egy operátor. Azt vizsgálja, hogy benne van-e egy elem például egy listában. Van ilyen alakja is:

```
if keresett_elem not in adatszerkezet:
```

Az eldöntés általánosan ezt jelenti: eldöntjük, hogy egy adott gyűjteményes adatszerkezetben (például listában vagy sztringben) van-e adott tulajdonságú elem.

Az eldöntés programozási tétele általánosan:

```
sorozat = valamilyen lista, range, vagy más bejárható objektum
ciklus sorozat minden elem-ére
   ha az elem keresett tulajdonságú:
      kiírjuk, vagy egy változóban jelezzük, hogy van
      break
   elágazás vége
ciklus vége
```

Valójában ez az egy algoritmus kétféleképp használatos. Az egyik esetben a bejárható objektumot a szokásos módon, érték szerint járjuk be:

```
vanilyen = False
for elem in bejárható_objektum:
   if elem olyan_amilyet_keresünk:
      vanilyen = True
      break
```

A másik esetben **indexek szerint** járjuk végig a sorozatot (indexnek hívtuk azt a számot, ami megmondja, hogy a lista eleme hányadik a listában, de nullával indul.) Ez olyankor kell, amikor az előző vagy következő elemekhez akarjuk hasonlítgatni az épp vizsgált elemet.

Először a `len()` függvénnyel meghatározzuk a bejárható objektum elemszámát, majd ezt betoljuk a `range()` függvénybe. Ugye egy tízelemű lista elemszáma tíz, de a `range(10)` csak 0-tól 9-ig adja vissza a számokat. Ami viszont nem baj, mert a tízelemű lista pont indexei pont nullától kilencig terjednek.

```
for index1 in range(len(bejárható_objektum)):
   if bejárható_objektum[index1] olyan_amilyet_keresünk:
      print(bejárható_objektum[index1])
      break
```

Feladatok

a) Kérj be egy nap nevét a felhasználótól és döntsd el, hogy tényleg egy nap nevét adta-e meg. Segítségül: tárold el a hét napjait egy listában és vizsgáld meg, hogy amit a felhasználó megadott, az eleme-e ennek a listának.
b) Kérjél be öt számot szóközzel elválasztva és döntsd el, hogy mennyi van 10 és 20 között!
c) Döntsd el egy 100-nál kisebb bekért számról, hogy prímszám-e! (Segítség: írd egy listába a 100 alatti prímszámokat, segít ebben a Négyjegyű függvénytáblázat.)
d) Feljegyeztük, hogy a házunk előtt mennyi autó ment el 4 és 12 óra között: [34, 35, 37, 37, 40, 42, 44, 45] és [31, 32, 34, 35, 35, 35, 40, 41]. Döntsd el, hogy folyamatosan nőtt-e ez a szám a vizsgált időben! (Segítség: Itt kell az indexes-bejárós változata a tételnek. Hasonlítsuk össze az előző elemmel a lista aktuális elemét, és ha nem nagyobb…)

Kiválasztás

Ha konkrétan az elem indexére vagyunk kíváncsiak, akkor használhatjuk a lista típus `index()` tagfüggvényét. Ha például van egy *allatok* listánk és arra vagyunk kíváncsiak, hogy hányadik elem a *hernyó*, akkor így keresünk rá:

keresett_index = allatok.index('hernyó')

Az eldöntés tételében azt vizsgáltuk, hogy van-e adott tulajdonságú elem a listánkban. Most abból indulunk ki, hogy van ilyen tulajdonságú elem és azt is meg kell határozni, hogy melyik az. Erre a kérdésre válaszol a kiválasztás tétele.

A programozás tétel általánosan:

sorozat = valamilyen lista, range, vagy más bejárható objektum

ciklus sorozat minden elem-ére
  ha az elem keresett tulajdonságú:
    kiírjuk az elemet, az indexét, vagy egy változóba tesszük az indexét
    break
  elágazás vége
ciklus vége

Feladatok:

a) Adott egy lista: [17, 74, 33, 28, 67, 12, 42, 63]. Írj programot, ami válaszol a válaszol a következő kérdésekre: Melyik az első páros szám? És az utolsó? Hányadik az első héttel osztható szám? Melyik az első 60 és 70 közé eső szám? Hányadik elem a 12? Mi a 12 indexe?
b) Add meg 68 és 54 legnagyobb közös osztóját!
c) Add meg 14 és 18 legkisebb közös többszörösét!
d) Add meg az első olyan hatjegyű számot, aminek a hatodik számjegye 7!
e) Írjál olyan programot, amely bekér egy adott szöveget, és abban az ékezetes karaktereket nem ékezetesre cseréli! Segítségül egy javaslat: készíts egy olyan listát, amiben az ékezetes betűk és cserélendő párjuk található, pl. *csere[['á','a'],['é','e'],...]* és a bekért szöveg minden karaktere esetén vizsgáld meg, hogy egyenlő-e valamelyik első elemmel a *csere* lista párjai közül, és ha igen, cseréld le a pár másik elemére. Tovább is fejlesztheted ezt a

megoldást, hiszen elég sok magyar betűt kell lecserélni az ’o’ vagy az ’u’ ékezetmentes karakterre.

Maximum kiválasztás

- maximumkiválasztáshoz egyszerű esetekben használhatod a `max()` függvényt
- minimumkiválasztáshoz egyszerű esetekben használhatod a `min()` függvényt

Pl. print(max(fizetesek),min(fizetesek))

Általánosan

sorozat = valamilyen lista, range, vagy más bejárható objektum

legnagyobb = az első elem a sorozat-ban

ciklus sorozat minden elem-ére

    ha az elem > legnagyobb:

        legnagyobb = elem

    elágazás vége

ciklus vége

Pythonban értékek szerint bejárva a bejárható objektumot:

legnagyobb = lista[0]

for elem in lista:

if elem > legnagyobb:

legnagyobb = elem

print(elem)

Feladatok

a) Adott a -2, 8, 7, -12, -22, 19, -5, 4 lista. Melyik a legkisebb eleme? Melyik a legkisebb pozitív eleme?
b) Határozzuk meg a fenti sorozat legnagyobb abszolútértékű elemét!
c) Keressük meg a fenti sorozatban a legnagyobb összegű szomszédos elempárt!
d) Dobj fel egy kockát 10.000-szer! Melyik szám jött ki legtöbbször?
e) Tárold listában néhány osztálytársad keresztnevét! Add meg a leghosszabb nevet! Ha több ilyen hosszúságú név is van, add meg mindet!

Megszámolás

A Pythonban működik a lista típus `count()` függvénye, pl.

['alma', 'körte', 'alma', 'alma'].count('alma') eredménye 3,

általános formája: listanév.count(keresett_elem)

Alapvetően arról szól, hogy

- definiálunk egy számlálót (egy változót), amiben számon tartjuk, hogy eddig hány, a keresett tulajdonsággal bíró elemet találtunk (a számláló kezdeti értéke nulla);
- végigjárjuk a bejárható objektum összes értékét, és ha egy elem adott tulajdonságú, akkor növeljük a számlálót.

Mondatszerű leírással:

sorozat = valamilyen lista, range, vagy más bejárható objektum

számláló = 0

ciklus sorozat minden elem-ére

  ha az elem keresett tulajdonságú:

    növeljük a számlálót

  elágazás vége

ciklus vége

Pythonban, indexek szerint bejárva a bejárható objektumunkat (ez kell, ha a szomszédos elemekhez akarjuk hasonlítgatni az adott elemet):

```
számláló = 0
for index in range(len(bejárható_objektum)):
  if bejárható_objektum[index] olyan_amilyet_keresünk:
    számláló += 1
```

De ez is működik:

```
számláló = 0
for elem in bejárható_objektum:
  if elem olyan_amilyet_keresünk:
    számláló += 1
```

Feladatok

a) Dobj fel egy kockát 100 000-szer, a dobásokat tárold listában! Hány hatos van?
b) Hány 10-nél nagyobb szám van a [3, 6, 12, 3, 14, 5, 18] listában?
c) Hozz létre egy 100 elemű listát 0 és 100 közötti véletlen számokból! Mennyi 30-nál kisebb, illetve 60-nál nagyobb szám van a listában?
d) Hány ’a’ betű, illetve ’e’ betű van a következő szövegben: ’Már egy hete csak a mamára gondolok.’
e) Dobj fel egy kockát 100 000-szer, a dobásokat tárold listában! Hányszor sikerült kidobnod az egyes számokat?
f) Hozz létre egy 100 elemű listát 0 és 100 közötti számokból! Hány olyan páros szám van benne, ami után szintén páros szám áll?

Keresés

Pythonban egy lista adott tulajdonságú elemeinek az indexeit kigyűjthetjük egy külön listába:

indexek = [i1 for i1 in range(len(lista)) if lista[i1] adott tulajdonságú]

Pl. ha egy számokból álló lista esetében arra vagyunk kíváncsiak, hogy milyen indexű elemek nagyobbak 10-nél:

```
indexek = [i1 for i1 in range(len(lista)) if lista[i1]>10]
if not indexek:
        print('Nem volt ilyen elem')
else:
        str_indexek = [str(elem) for elem in indexek]
        print('A megfelelő indexek:',', '.join(str_indexek))
```

Általánosan a keresés megmondja, hogy egy listában melyik az adott tulajdonságú elem. Ha nincs benne, akkor megmondja, hogy nincs. A lineáris keresés programozási tétel leírása a következő:

```
sorozat = valamilyen lista, range, vagy más bejárható objektum
ciklus sorozat minden elem-ére
   ha az elem keresett tulajdonságú:
      egy változóba tesszük az indexét
      logikai változóban felírjuk magunknak, hogy találtunk ilyet
      break (azaz kilépünk a ciklusból)
   elágazás vége
ciklus vége
ha volt keresett tulajdonságú elem:
   kiírjuk az elemet, vagy az indexét
   kilépünk a ciklusból
különben:
   kiírjuk, hogy nem volt ilyen elem
elágazás vége
```

Ez Pythonban, **értékek szerint bejárva** a listánkat:

```
for elem in bejárható_objektum:
   if elem olyan_amilyet_keresünk:
      print(elem)
      break
else:
        print('nem volt ilyen')
```

És indexek szerint bejárva a listánkat:

```
for index in range(len(bejárható_objektum)):
    if bejárható_objektum[index] olyan_amilyet_keresünk:
        print('Az elem helye:', index, 'maga az elem:', bejárható_objektum[index])
        break
    else:
        print('nem volt ilyen')
```

A Python egyszerűsítése:

- Az `in` operátorral meg tudjuk mondani, hogy van-e keresett érték, a lista objektumtípus `index()` tagfüggvényével meg azt, hogy ez az érték hányadik.
- Megjegyzés: az `index()` csak az első előfordulás indexét adja meg, azaz csak egy előfordulást talál meg. Ha több is lehet, akkor mást használunk, pl. egy számlálót, és nem lépünk ki a ciklusból az elem megtalálása után. Például ha a listában számok szerepelnek, és meg akarjuk mondani, hogy mely elemek 10-nél nagyobbak, így is eljárhatunk (nem használjuk a Python rövidítése lehetőségét):
  talalatok = []

```
for index1 in range(len(lista)):
    if lista[index1] > 10:
            talalatok.append(index1)
if len(talaltok) > 0:
    print('10-nél nagyobbak a lista következő helyein vannak: ',end='')
    for elem in talalatok:
            print(elem+1,end=' ')
else:
    print('Nincs 10-nél nagyobb elem')
```

Feladatok

a) Adottak Berci jegyei időrendben: 5, 4, 4, 3, 2, 4, 5, 5, 4 . Kapott-e egymás után azonos osztályzatokat? Hányadik helyen vannak azok, amiket azonos osztályzat követ?
b) Kérj be a felhasználótól egy mondatot és döntsd el, hogy van-e benne két egyforma betű!
c) Adott a 3, 6, 4, 9, 8, 11, 12, 20 sorozat. Van-e benne két olyan szomszédos szám, amelyek csak 1-gyel térnek el egymástól (felfelé vagy lefelé)? Írasd ki, melyek ezek a számpárok!

Kiválogatás

Ha adott egy lista, a Pythonban egyszerűen kigyűjthetjük egy másik listába az adott tulajdonságú elemeket: `eredmeny = [elem for elem in lista if elem adott tulajdonságú]`

Pl. eredmeny = [szam for szam in szamok if szam>10] a 10-nél nagyobb számokat gyűjti ki az eredmeny listába

A kiválogatás olyan másolás, ahol a második listába csak bizonyos feltételnek megfelelő elemeket másolunk át.

kiválasztottak = list() vagy kiválasztottak = []

for elem in bejárható_objektum:

    if elem olyan_amilyet_keresünk:

        kiválasztottak.append(elem)

Feladatok

a) Készíts egy 100 elemű listát 1 és 10 közötti véletlen számokkal! Mások át az 5, vagy annál nem nagyobb elemeket egy másik listába!
b) Készíts egy olyan programot, amelyik bekér egy mondatot a felhasználótól, és az ékezetes karaktereket ékezet nélkülire cseréli, valamint a szövegből kitörli az írásjeleket és a szóközöket! Magyarul egy ékezet nélküli karakterláncot kapsz majd a szövegből.
c) Készíts egy 100 elemű listát 1 és 10 közötti véletlen számokkal! Töröld ki belőle az 5-nél nagyobb számokat!
d) Válogasd ki a 100-nál kisebb, 7-tel osztható pozitív egészszámokat egy tömbbe!
e) Készíts egy 10 elemű listát 1 és 10 közötti véletlen számokkal! Válogasd ki a lista elemeit egy másik listába úgy, hogy minden elem csak egyszer szerepeljen!

Rendezés

Ha csak számokból áll a lista: `lista.sort()` növekvőbe rendezi, `lista.sort(reverse=True)` csökkenőbe.

Több dimenziós tömbök rendezése pl. `lista.sort(key = lambda e:e[1])`

Általánosan arról szól, hogy miként rendezhető egy lista, azaz miként lehet a 2, 3, 1, 4-ből 1, 2, 3, 4-et csinálni. Rendezésből sokféle van, de a Python beépített `sorted()` függvényénél többet nem kell tudnod róluk.

A `sorted()` függvény és a `list.sort()` tagfüggvény:

- a rendezendő lista a `sorted()` függvénynél, a `list.sort()` esetében nyilván a *list*-et rendezzük, például `sorted(autok)` vagy `autok.sort()`
- ha fordítva akarunk rendezni: `reverse=True`
- ha nem úgy akarunk rendezni, ahogy a józan ész diktálja: kulcsfüggvény.

A két *sort*-függvény lényegében ugyanaz, és mindkettő nagyon gyors, nagyon jó hírnek örvend a programozók világában. A kulcsfüggvény pedig az igazi menő dolog. Megjegyzés: ide kapcsolódnának a *lambdafüggvények*, íme egy példa.

Adott a következő lista (ki melyik napon született):

lista = [[’Hapci’,12],[’Morgó’,16],[’Tudor’,2],[’Vidor’,20]]

Ha a nap (vagyis az 1-es indexű elemek) szerint akarjuk rendezni a tömböt, a következőt csináljuk:

lista.sort(key=lambda elem: elem[1])

Feladatok

a) Nézz utána, mi a különbség `sorted(lista)` és `lista.sort()` között!
b) Állíts elő 100 tagú, 1 és 1000 közötti véletlen számokból álló rendezett listát!

6. Algoritmizálás, mondatszerű algoritmus-leíró eszköz[2]

Az ***algoritmus*** elemi lépések egymás utáni sorozata, melyek végrehajtása egy probléma megoldásához vezet. Az egyes részlépéseknek végrehajthatónak és egyértelműnek kell lennie. Ez biztosítja, hogy tetszőlegesen sokszor végrehajtva, mindig ugyanazt a részeredményt kapjuk.

Definiálhatjuk így: Az algoritmus egyértelműen végrehajtható tevékenység-, vagy utasítássorozat, amely véges sok lépés után befejeződik.

Az algoritmus jellemző tulajdonságai:

- ***végesség***, amely azt jelenti, hogy a folyamat véges számú utasítással, azaz véges számú lépés után befejeződik vagy eredményt szolgáltat. A lépések sorozatát úgy határozzuk meg, hogy bármely végrehajtott lépés után egyértelműen adódjon a következő.
- ***egymásutániság kitételének***, miszerint minden lépést követnie kell egy másik lépésnek –kivéve az utolsót – és ezek a lépések meghatározott sorrendben kövessék egymást.
- ***általánosság***, amely azt jelenti, hogy nem csak egy konkrét esetben használható, hanem az összes azonos jellegű feladatra.

Egy adott problémára több algoritmust is lehet készíteni, de mindig törekedni kell arra, hogy

- vagy a ***lehető legkevesebb lépéssel*** oldjuk meg a feladatot,
- vagy a ***lehető legegyszerűbb módszerrel*** (pl. oktatási célokra),
- vagy a lehető legrövidebb idő alatt.

Az algoritmus készítésének lépései:

- ***tervezés***, melynek során meghatározzuk a probléma megoldásához vezető lépéseket, feltételeket.
- az egyes lépések valamilyen formában való ***rögzítése***.

Az algoritmusok rögzítésére, leírására több módszer is létezik, és majdnem mindnek saját jelrendszere van. Legismertebb módjai a mondatszerű leírás, a folyamatábra és a struktogram.

Az algoritmusokat folyamatos szövegként is rögzíthetjük, ilyenek találhatók a szakácskönyvekben is.

Az algoritmus másik szöveggel szemléltetett formája a ***mondatszerű leírás***, amely már bonyolultabb problémák megoldását is jól szemlélteti. Az algoritmus egymást követő lépéseit rövid és tömör

[2] Ebben a fejezetben felhasználtam a következő forrást: https://tudasbazis.sulinet.hu/hu/informatika/informatika/informatika-5-evfolyam/algoritmusok-szoveges-rajzos-megfogalmazasa/algoritmusok-leirasa és https://tudasbazis.sulinet.hu/hu/informatika/informatika/informatika-5-evfolyam/algoritmusok-szoveges-rajzos-megfogalmazasa/mondatszeru-leiras ; letöltés: 2019. szept. 7.

mondatokkal, illetve mondatszerű szerkezetek egymásutánjával írjuk le. A folyamatot alkotó utasításokat kicsit beljebb kezdjük, ezzel áttekinthetőbbé válik a leírás.

Az algoritmus alapvető lépéstípusait meghatározott szavakkal kell megfogalmaznunk, például:

Az adatbevitel formája: Be: adat vagy felsorolás

Feltétel megadás formája: Ha feltétel Akkor tevékenység, Egyébként tevékenység, Elágazás vége.

Ha egy szám abszolút értékét kell meghatározni:

Be: szám

Feltételvizsgálat: Ha a szám pozitív, akkor abszolút értéke saját maga, egyébként az ellentettje.

Elágazás vége

Példák[3]:

```
Téglalap:
Be: a,b
k := 2*(a+b)
t := a*b
Eredmény: "A kerület: ", k
Eredmény: "A terület: ", t
Vége
```

```
Téglalap:
Be a,b
Ha (a>0) és (b>0), akkor
        k:=2*(a+b)
        t:=a*b
        Eredmény "A kerület: ", k
        Eredmény "A terület: ", t
Különben
        Eredmény "Hibás oldalhossz!"
Ha vége
Vége
```

Első_100_páros_szám:

Minden i:=2; i<200; i:=i+2 végezd el

    Ki: i

Minden vége

Szokás a fenti példát az alábbi módon leírni:

Első_100_páros_szám:

Ciklus i:=2-től 200-ig, lépésköz:=2

    Ki: i

Ciklus vége

[3] Forrás: http://trafo01.uw.hu/10F1_prelm/16_prelm.html

Feladat: Kódold le a fenti leírással megadott algoritmusokat!

7. Listák kezelése

8. Stringek kezelése

A Python a stringeket karakterek listájaként kezeli, ezért a listakezelésnél leírtak szerint érhetők el az egyes karakterek vagy rész-stirngek. Természetesen vannak stringkezelő függvények is, íme a legfontosabbak:

`szöveg.find('char')` visszatér a karakter pozíciójával

`format()` formázott kiíratás

`szöveg.index('rész')` visszatér a rész alstring pozíciójával

`szöveg.count(valami)` megszámolja, hányszor fordul elő a valami a szövegben

`szöveg.lower()` kisbetűssé teszi a szöveget

`szöveg.upper()` nagybetűssé teszi a szöveget

`szöveg.split(elválasztó)` listába szedi szét a szöveget az elválasztó szerint

`szöveg.splitlines()` a szöveget sorok listájába szedi szét

`szöveg.strip()` kipucolja a szöveget, pl.

txt = ",,,,,xxcczz.....banán....ccc"

x = txt.strip(",.xzc")

print(x)

eredménye: banán

txt = " banán "

x = txt.strip()

print(x)

eredménye: banán

9. Szubrutinok készítése és alkalmazása

Ha szét akarjuk szedni kisebb részekre a programot, vagy ha egy utasítás sorozatot több helyzetben akarunk végrehajtani, szubrutint készítünk. A Pythonban ezt függvény megírásával érjük el.

Szintaxisa: `def` függvénynév(paraméter1,paraméter2,...):

utasítások

Kilépés, visszatérés a függvényből: `return`, ha nem ad vissza értéket, illetve `return érték`

Pl. ha át akarunk váltani másodpercekre egy időpontot:

```
def mperc(ora, perc, masodperc):
    return masodperc + 60*perc + 3600*ora
```

10. Vezérlési szerkezetek

Elágazás

Az elágazás szintaxisa:

if (feltétel):
utasítások, amit végrehajt **ha a feltétel igaz**
elif (feltétel2):
új vizsgálat és utasítások, amit végrehajt **ha a feltétel hamis**
el is maradhat
else:
utasítások, amit végrehajt **ha a feltétel hamis**

A feltétel: logikai kifejezés, igaz vagy hamis lehet, de összekapcsolható a logikai operátorokkal.

Az utasítások: új beljebbkezdésben szereplő utasítások (ameddig a beljebbkezdés tart)

Pythonban nincsen többszörös elágazás.

while ciklus

A `while` ciklus szintaxisa:

while (feltétel):
utasitások
else:
utasítások, amelyek akkor futnak le, ha a feltétel nem
teljesül vagy ha már vége a ciklusnak; el is maradhat

A `while` a hagyományos elől tesztelő ciklus. Az `else` ág akkor fut le, ha a ciklusfeltétel nem teljesül (tehát a ciklus elhagyása után mindig, kivéve ha `break` utasítással hagyjuk el a ciklust).

A programozónak kell gondoskodnia a ciklusfeltétel HAMIS-sá válásáról, mert ellenkező esetben egy végtelen ciklusba kerül a program.

Kiugrási lehetőségek a ciklusból:

- `break` - rögtön a ciklus utáni utasításra kerül a vezérlés
- `continue` - a ciklusfeltétel tesztelésére ugrik a vezérlés

A while hátul tesztelő ciklussá is tehető:

```
while True:
    utasítások
    if (feltétel):
        break
```

Nincs GOTO utasítás! Nincs lehetőség egyszerre több ciklusból való kiugrásra.

for ciklus

A for ciklus szintaxisa:

`for` ciklus változó `in range` (kezdőérték, végérték, lépésköz) **:**
utasitások
else:
utasítások (ha a ciklus véget ér, ez fut le, de el is maradhat, ha nincs rá szükség)

Példa:

```
for i in range(1,11,2):
print('szám:',i )
else:
print('VÉGE a ciklusnak!!!')
```

A `for` ciklussal ***egy lista elemein*** is végiglépkedhetünk:

```
for elem in lista:
    print(elem)
```

Előfordulhat, hogy ***indexeléssel*** kell végigmenni az elemeken:

```
for index1 in range(len(lista)):
    print(lista[index1])
```

Feladatok:

a) Kérj be egy pozitív egész számot (n), és írasd ki a számokat 1-től n-ig egyszer a for, egyszer a while ciklussal!

b) Kérj be egy pozitív egész számot és írasd ki, hogy páros vagy páratlan.

c) Kérj be egy számot és ellenőrizd, hogy pozitív egész szám-e. Egészen addig folytasd a bekérést, amíg ez a feltétel nem teljesül!

d) Készíts egy 100 elemű listát 1 és 10 közötti véletlen egészekből. Készíts egy másikat ebből úgy, hogy minden elemét megduplázod!

11. Rekurzió[4]

A rekurzió egy programozási módszer. Azt az esetet nevezzük így, amikor egy eljárásban (függvényben) szereplő kód önmagát (ugyanazt az eljárást, függvényt) hívja meg. Természetesen a folyamatot ebben az esetben is véges számú lépés után meg kell állítani.

Egyszerűsítve: olyan alprogramokat nevezünk rekurzívaknak, amelyek meghívják önmagukat.

Jellemző példa egy szám faktoriálisának kiszámításának algoritmusa. Egy szám faktoriálisát úgy számíthatjuk ki, hogy kiszámítjuk a nála eggyel kisebb szám faktoriálisát és megszorozzuk a számmal. A rekurzió véget ér, ha eljutunk a 0-hoz.

```
Ha n = 0 akkor
        fakt =1          #hiszen 0! = 1
 Egyébként
fakt = fakt(n-1)*n
 Elágazás vége
```

Pythonban a faktoriális kiszámítása:

```
def faktorialis(szam1):
    if szam1 == 0:
        return 1
    else:
        return(faktorialis(szam1-1)*szam1)
szam = int(input('Adj meg egy pozitív egész számot: '))
```

[4] Forrás: https://wiki.prog.hu/wiki/Rekurzi%C3%B3

print(szam,'faktoriálisa:',faktorialis(szam))

További feladatok:

a) Egy számtani sorozat minden elemét úgy kapjuk, hogy az eggyel előző eleméhez hozzáadjuk a különbséget (differenciát): $a_n = a_{n-1} + d$

A sorozatra függvényként tekintve, írj rekurzív függvényt szamtani néven, amelyik kiszámolja egy számtani sorozat n-edik elemét! A függvény paraméterei: szamtani(n, kezdo, diff), azaz a keresett elem sorszáma, a kezdőérték és a differencia.

b) Fibonacci sorozat: első és második eleme 1, a harmadik elemtől kezdve úgy kapjuk meg az elemeket, hogy az előző két elemet összeadjuk.
Vagyis a sorozat így néz ki: 1, 1, 2, 3, 5, 8, 13 ...
Írj rekurzív függvényt, amely bekéri n pozitív egészet, és kiszámítja a sorozat első n elemét!

c) Valósítsd meg a hatványozást rekurzív módon, a következő mondatszerű leírás segítségével:

Függvény Hatvány(x: valós, n: egész): valós

Ha n > 0 akkor

Hatvány := x * Hatvány(x, n-1)

különben

Hatvány := 1

Elágazás vége

Függvény vége.

d)* Számítsd ki rekurzív függvény segítségével n alatt a k, vagyis $\binom{n}{k}$ értékét!

12. Fájlkezelés

Beolvasás txt fájlból, helyközzel elválasztott adatok esetén:

```
fajl = open('valami.txt')
forras = fajl.read().splitlines()
fajl.close()
```

Ekkor a fájl minden sora a *forras* lista 1-1 eleme lesz. Ha a sorokban szóközzel elválasztott adatok vannak, így rakhatjuk ezeket az *adatok* listába:

```
adatok = []
for sor in forras:
    adatok.append(sor.split(' '))
```

Vannak olyan esetek, amikor nem engedik beolvasni a teljes fájlt, mert túl nagy lenne, nem lehetene tárolni a memóriában. Ilyenkor soronként is végiglépkedhetünk a fájlon. Fontos, hogy a `read()` a sorvégi `\n` karaktereket is beolvassa, ettől meg kell szabadulni.

Pl. a szoveg.txt fájl minden sorában egy-egy szó van, és a leghosszabbat kell kiválasztani.

```
fajl = open('szoveg.txt')
leghosszabb = ''
for sor in fajl:
    if len(sor[0:len(sor)-1])>len(leghosszabb):
    leghosszabb = sor[0:len(sor)-1]
fajl.close()
print('A leghosszabb szó:',leghosszabb)
```

Ennyi információ elég a programozási érettségi feladatok megoldásához, de a következő linkeken további információt találsz:

- http://wiki.math.bme.hu/view/Informatika2-2015/Eloadas_7_Python-7_Fajl_kezeles_Kitekintes
- http://people.ubuntu.com/~kelemeng/.ufp3/files.html

Fájl írásra megnyitása: fajl = open('kepviselok.txt',mode='w',encoding='utf-8')

Általában így használjuk: fajl = open('kepviselok.txt','w')

Csak az utf-8-cal írja ki rendesen az ékezetes karaktereket!

Addig nem ír semmit a fájlba, amíg a `close()`-zal le nem zárjuk.

Fájl megnyitás módok: mode=...

'r'	olvasásra
'w'	írásra, de nem vizsgálja, hogy van-e már ilyen nevű fájl - lecseréli
'a'	hozzáfűzés a végéhez; ha nincs ilyen fájl, létrehozza
't'	megnyitás szövegfájlként (alapértelmezett)
'b'	bináris fájl megnyitása – bájtokat lehet bele írni
'+'	módosításra nyitja meg (írás vagy olvasás)

13. Tartalom