# BEVEZETÉS A FÁJL- ÉS ADATBÁZIS-KEZELÉSBE PYTHON NYELVEN

(Informatikai szakközépiskola – 11. évfolyam)

# 1. A fájlok haszna

A programjaink eddig csak nagyon kevés adatot kezeltek. Ezeket minden alkalommal a programtestbe kódolhatnánk (például egy listába). Ez az eljárás azonban alkalmatlan nagyobb tömegű adat kezelésére. Több nemkívánatos problémával találkozhatunk :

A program olvashatósága nagyon gyorsan fog romlani, amikor az adatok száma jelentősen megnő. Ennek következtében megnő a valószínűsége annak, hogy egy vagy több szintaxishibát fogunk ejteni a hosszú listában. Az ilyen hibákat nagyon nehéz kiküszöbölni.

Új adatok hozzáadása, vagy bizonyos adatok módosítása minden alkalommal a program forráskódjának megnyitásával jár. Ennek következtében kényelmetlenné válik ugyanannak a forráskódnak az újraírása, mivel nagyszámú terjedelmes adatsort fog tartalmazni.

Az adatcsere más programokkal (amiket esetleg más programozási nyelveken írtak) teljesen lehetetlen, mert az adatok a program részét képezik.

Ez utóbbi megjegyzés sugallja a követendő irányt: itt az ideje, hogy megtanuljuk külön fájlokba szétválasztani az adatokat és az őket kezelő programokat.

Ahhoz, hogy ez lehetséges legyen, el kell lássuk a programjainkat különböző mechanizmusokkal, amik lehetővé teszik a fájlok létrehozását, azokba adatok mentését, később azok visszanyerését.

A programozási nyelvek többé-kevésbé kifinomult utasításkészletet kínálnak ezeknek a feladatoknak az elvégzéséhez. Amikor az adatok mennyisége nagyon megnő, akkor szükségessé válik a közöttük fennálló kapcsolatok struktúrálása. Ekkor relációs adatbázisoknak nevezett rendszereket kell kidolgozni, amiknek a kezelése nagyon összetett lehet. Ez olyan specializált programoknak a feladata, mint az Oracle, IBM DB, Adabas, PostgreSQL, MySQL, stb. A Python tökéletesen alkalmas az ezekkel a rendszer való dialógusra, azonban ezt a későbbiekre hagyjuk.

Jelenlegi ambícióink sokkal szerényebbek. Nincsenek százezres mennyiségű adataink, egyszerű módszerekkel is megelégedhetünk, melyekkel egy közepes méretű fájlba bejegyezhetjük, és aztán kinyerhetjük onnan azokat.

## *Munkavégzés fájlokkal*

Egy fájl használata sokban hasonlít egy könyvéhez. Ahhoz, hogy egy könyvet használjunk, először meg kell azt találni (a címe segítségével), utána ki kell nyitni. Amikor befejeztük a használatát, be kell zárni. Amíg nyitva van különböző információkat lehet belőle olvasni és megjegyzéseket lehet bele írni, de általában a kettőt nem tesszük egyszerre. Az oldalszámok segítségével minden esetben meg tudjuk határozni, hogy hol tartunk a könyvben. A könyvek többségét a lapok normális sorrendjét követve olvassuk, de dönthetünk úgy is, hogy egy tetszőleges bekezdést olvasunk el.

Mindez, a fájlokra is alkalmazható. Egy fájl egy háttértárolóra (merevlemezre, CD-ROM-ra, stb) rögzített adatokból áll. A neve segítségével férünk hozzá (ami tartalmazhat egy elérési utat is). A fájl tartalmát mindig tekinthetjük egy karaktersorozatnak, ami azt jelenti, hogy ezt a tartalmat, vagy annak bármely részét a karakterláncok kezelésére szolgáló függvények segítségével kezelhetjük.

## *Fájlnevek, aktuális könyvtár*

A magyarázatok egyszerűsítése kedvéért csak a kezelt egyelőre csak a fájlok nevét használjuk. Ilyenkor a Python a szóban forgó fájlokat az aktuális könyvtárban fogja létrehozni és keresni. Ez szokás szerint az a könyvtár, ahol maga a program található, kivéve ha a

programot egy IDLE shell ablakból indítjuk. Ez esetben az aktuális könyvtár az IDLE indításakor van definiálva (Windows alatt ennek a könyvtárnak a definíciója részét képezi az indító ikon tulajdonságainak).

Ha az IDLE-vel dolgozunk, akkor biztos, hogy a Pythont az aktuális könyvtárának megváltoztatására akarjuk majd kényszeríteni azért, hogy az megfeleljen az elvárásainknak. Ennek megtételéhez használjuk a következő utasításokat a kapcsolat felépítésnek az elején. (Most feltételezem, hogy a használni szándékozott könyvtár a /home/user1/pythonprogik ). Használhatjuk ezt a szintaxist (vagyis szeparátorként a / karaktert nem pedig a \ karaktert: ez a konvenció érvényes a Unix-világban). A Python automatikusan elvégzi a szükséges konverziókat annak megfelelően, hogy MacOS-ben, Linuxban, vagy Windowsban dolgozunk.

```
from os import chdir

chdir("/home/user1/pythonprogik")
```

Az első parancs az ***os*** modul ***chdir***() függvényét importálja. Az ***os*** modul egy sor függvényt tartalmaz, amik az operációs rendszerrel (***os*** = *operating system*) való kommunikációt teszik lehetővé, bármi is legyen az operációs rendszerünk.

A második parancs a könyvtárat változtatja meg (***chdir*** - *change directory*).

*Megjegyzések :*

Vagy ezeket a parancsokat szúrjuk be a script elejére, vagy pedig megadjuk a file-ok nevében a komplett elérési utat, de ez azzal a veszéllyel jár, hogy megnehezítjük a programjaink írását.

Részesítsük előnyben a rövid fájlneveket. Mindenképpen kerüljük az ékezetes karakterek, a szóközök és a speciális tipográfiai jelek használatát.

## *Importálási formák*

Az imént alkalmazott utasítássorok alkalmat adnak egy érdekes mechanizmus magyarázatára. Tudjuk, hogy az alapmodulba integrált függvények kiegészítéseként a Python nagyszámú, specializáltabb függvényt bocsát a rendelkezésünkre, amik modulokban vannak csoportosítva. Már ismerjük a ***math*** és a ***Tkinter*** modulokat.

Ahhoz, hogy egy modul függvényeit használhassuk, importálni kell azokat. Ez kétféle módon történhet, amit a következőkben fogunk megnézni. Mindkét módszernek vannak előnyei és hátrányai.

***Feladat – Aktuális könyvtár 1***
Készítsünk olyan programot, amely kiírja az aktuális könyvtárat!


```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def kiir():
    import os
```

```
    utvonal = os.getcwd()
    mezo1.delete(0,END)
    mezo1.insert(0,utvonal)

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Aktuális könyvtár:')
gomb1 = Button(abl1, text='Kiír', command=kiir)
mezo1 = Entry(abl1, width=70)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row=1, sticky =E)
gomb1.grid(row =2, sticky =E, column =2)
mezo1.grid(row =1, column =2)

# indítás :
abl1.mainloop()
```

A kiemelt rész első sora teljes egészében importálja az ***os*** modult, ami számos függvényt tartalmaz az operációs rendszer eléréséhez. A második sor az ***os*** modul ***getcwd***() függvényét használja. Megállapítható, hogy a ***getcwd***() függvény az aktuális könyvtár nevét adja vissza (***getcwd*** = *get current working directory*).

***Feladat – Aktuális könyvtár 2***
Készítsünk olyan programot, amely kiírja az aktuális könyvtárat!


```
def kiir():
    from os import getcwd
    utvonal = getcwd()
    mezo1.delete(0,END)
    mezo1.insert(0,utvonal)
```

Ebben a példában az ***os*** modulból egyedül a ***getcwd***() függvényt importáltuk. Ezen a módon importálva a getcwd() függvény úgy épül be a kódunkba, mintha azt mi magunk írtuk volna. Azokban a sorokban, ahol használjuk, nem kell megismételni, hogy az ***os*** modul része.

Ugyanígy importálhatjuk ugyanannak a modulnak több függvényét:

**from *math* import *sqrt*, *pi*, *sin*, *cos***

Sőt a következő módon az összes függvényt importálhatjuk egy modulból :

**from *Tkinter* import ***

Ez az importálási mód megkönnyíti a kódírást. A hátránya az (különösen az utóbbi formának, amelyik egy modul összes függvényét importálja), hogy az aktuális névteret blokkolja. Előfordulhat, hogy bizonyos importált függvények neve megegyezik egy általunk definiált változó nevével, vagy más modulból importált függvények nevével. (Ha ez történik, akkor a két ütköző név egyike nyilvánvalóan már többé nem lesz hozzáférhető).

Az olyan programokban, melyek nagyszámú, különböző eredetű modult hívnak mindig inkább az első módszert érdemes előnyben részesíteni, vagyis amelyik a minősített neveket használja.

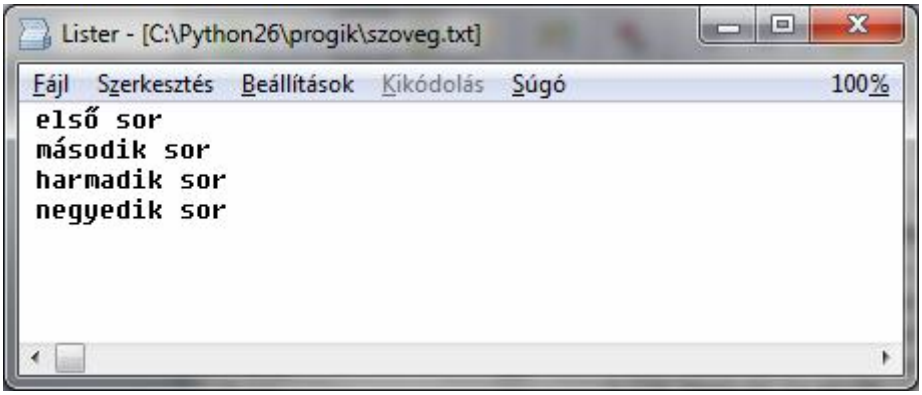
Általában ez alól a szabály alól kivételt teszünk a ***Tkinter*** modul speciális esetében, mert azokat a függvények, amiket ez a modul tartalmaz, nagyon gyakran hívjuk (már ha ennek a modulnak a használata mellett döntünk).

## *Szekvenciális írás fájlba*

A Pythonban a fájlokhoz való hozzáférést egy közbenső ***fájl objektum*** biztosítja, amit az ***open***() belső függvény segítségével hozunk létre. Ennek a függvénynek a hívása után a ***fájl objektum*** speciális metódusait használva olvasni és írni tudunk a fájlban.

Ha a fájl még nem létezik, akkor automatikusan létre lesz hozva. Ha viszont a név egy már létező és adatokat tartalmazó fájlra vonatkozik, akkor a bejegyzendő karaktereket hozzá fogja fűzni a meglévő karakterekhez.

***Feladat – Szöveges fájl létrehozása***
Készítsünk olyan programot, amely egy szövegbeviteli mezőbe beírt szövegeket fájlba ír!

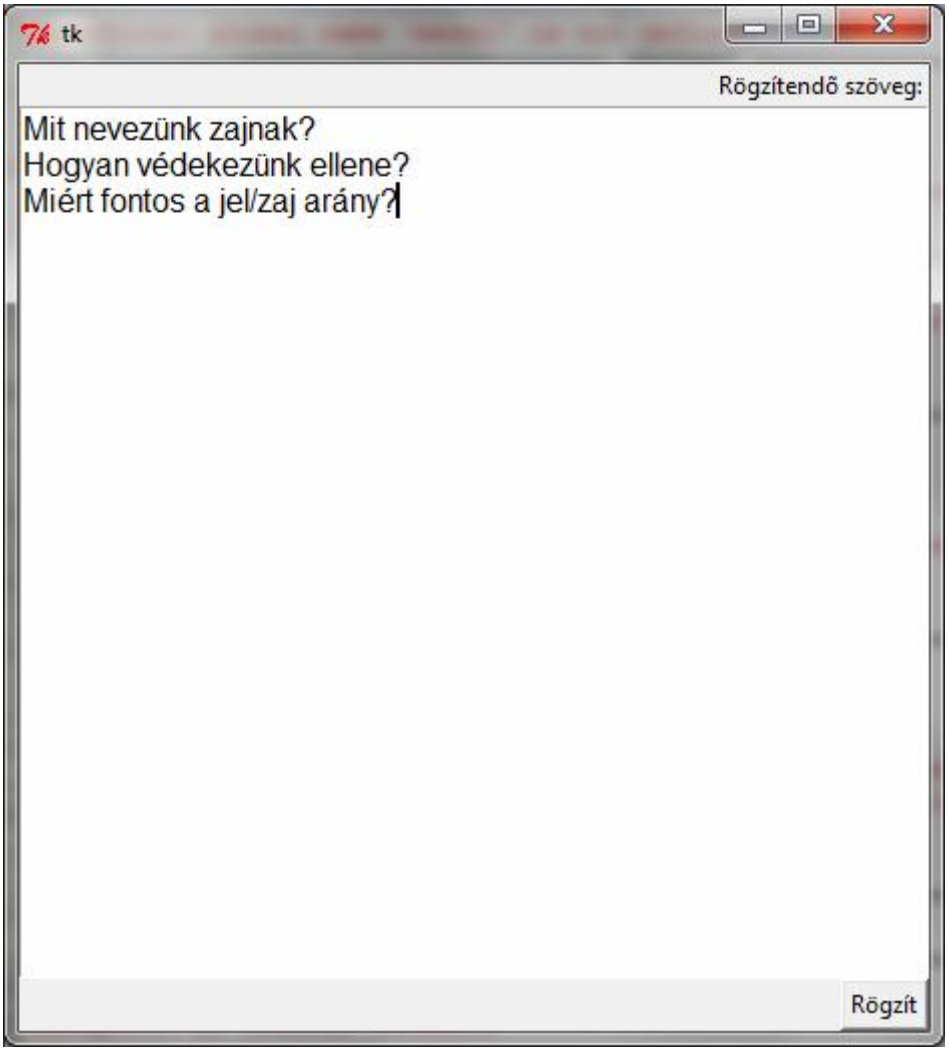



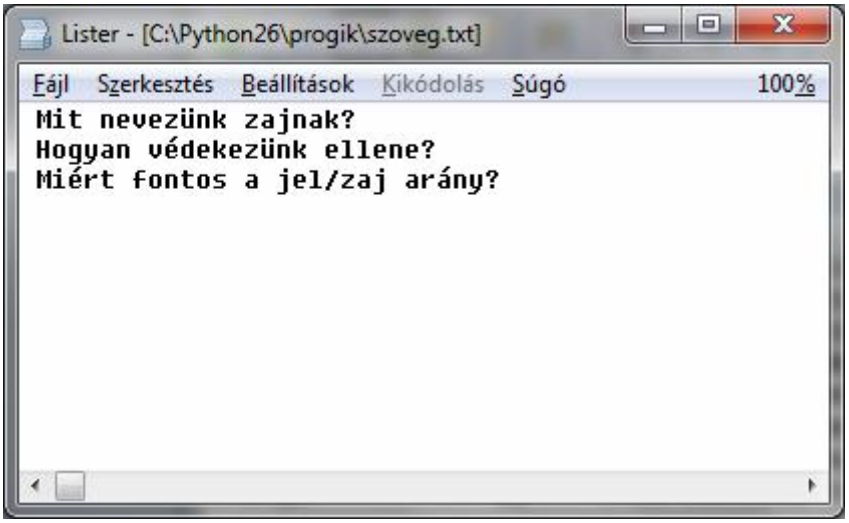

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def rogzit():
    f = open('szoveg.txt','a')
    sz = mezo1.get()+'\n'
    f.write(sz.encode("utf-8"))
    f.close()
    mezo1.delete(0,END)
```

```
abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Rögzítendő szöveg:')
gomb1 = Button(abl1, text='Rögzít', command=rogzit)
mezo1 = Entry(abl1, width=70)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row=1, sticky =E)
gomb1.grid(row =2, sticky =E, column =2)
mezo1.grid(row =1, column =2)

# indítás :
abl1.mainloop()
```

***Feladat – Szöveges fájl létrehozása 2***

Készítsünk olyan programot, amely egy listadoboz tartalmát fájlba írja ki!






```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def rogzit():
    f = open('szoveg.txt','w')
    sz = doboz1.get('1.0', END+'-1c')
    f.write(sz.encode("utf-8"))
    f.close()

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Rögzítendő szöveg:')
gomb1 = Button(abl1, text='Rögzít', command=rogzit)
import tkFont
doboz1 = Text(abl1, width=50, font=tkFont.Font(size=12))

# laptördelés a'grid' metódus segítségével :
txt1.grid(row=1, sticky =E)
gomb1.grid(row =3, sticky =E)
doboz1.grid(row =2)

# indítás :
abl1.mainloop()
```

*Megjegyzések :*

Az első kiemelt sor egy *f* nevű ***fájl objektumot*** hoz létre. Ez egy valódi fájlra hivatkozik (a merevlemezen vagy a hajlékony lemezen), aminek a neve **szoveg.txt** lesz. Ne keverjük össze a fájl nevét (**szoveg.txt**) a ***fájl objektum*** nevével ( *f* ), amin keresztül hozzáférünk a fájlhoz. A gyakorlat során ellenőrizhetjük, hogy a rendszerünkben (az aktuális könyvtárban) egy **szoveg.txt** nevű fájlt hozott létre a program, aminek a tartalmát bármelyik szövegszerkesztővel megnézhetjük.

Az ***open***() függvény két ***string*** típusú argumentumot vár. Az első a megnyitandó fájl neve, a második a megnyitás módja. Az **'a'** azt jelenti, hogy *hozzáfűzés* (**append**) módban kell a fájlt megnyitni, azaz a bejegyzendő adatokat a fájl végéhez, a már esetleg ott lévő adatokhoz

kell fűzni. A **'w'** (*írásra*) megnyitási módot is használhattuk volna, de ha ezt a módot használjuk, a Python mindig egy új (üres) fájlt hoz létre és az adatok írása ennek az üres fájlnak az elejétől kezdődik. Ha már létezik egy azonos nevű fájl, akkor azt előbb törli.

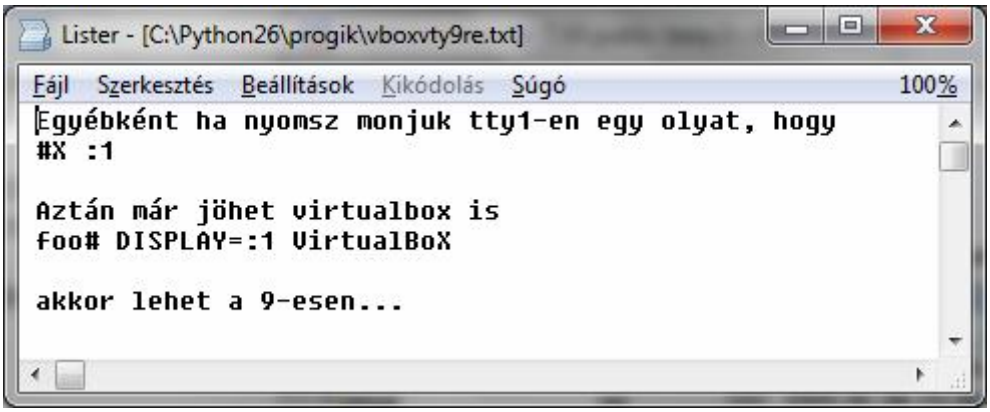
A ***write***() metódus végzi a fájlba írást. A kiírandó adatokat argumentumként kell megadni. Ezeket az adatokat sorban egymás után írja ki a fájlba (ezért beszélünk szekvenciális hozzáférésű fájlról). A ***write***() minden új hívása a már rögzített adatok után folytatja az írást.

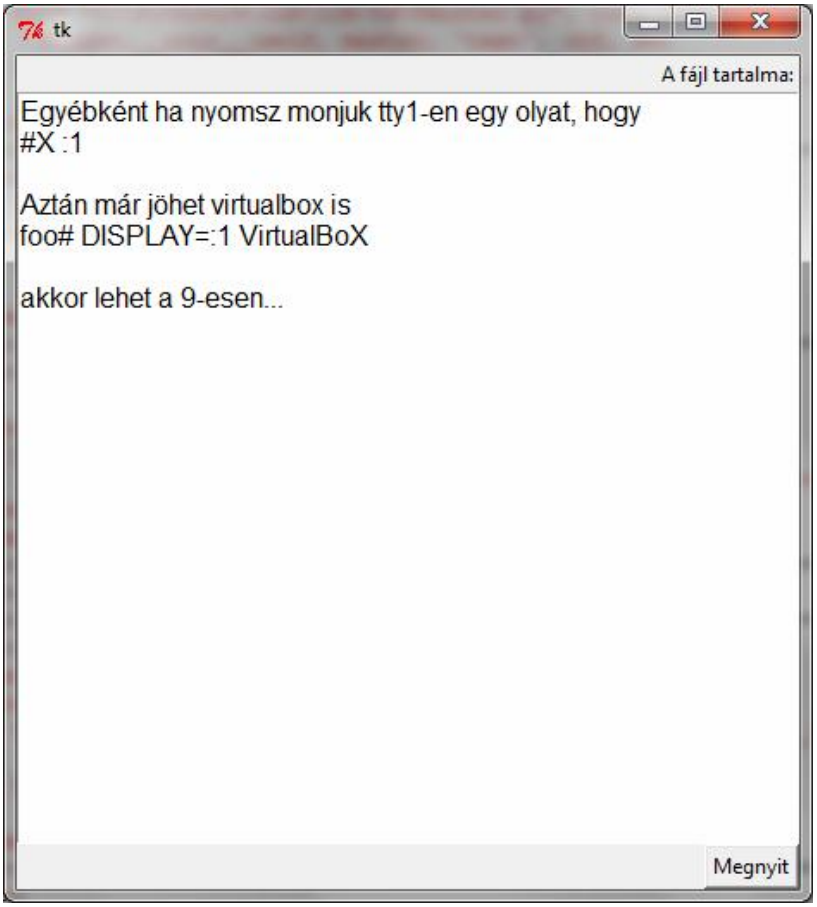
A ***close***() metódus lezárja a fájlt. Ettől kezdve az mindenféle használatra rendelkezésre áll.

## *Szekvenciális olvasás fájlból*

***Feladat – Szöveges fájl beolvasása***

Készítsünk olyan programot, amely egy szövegdoboz tartalmát egy szövegdobozba írja ki!



```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def megnyit():
    f = open('vboxvty9re.txt','r')
    sz = f.read()
    doboz1.delete('1.0', END)
    doboz1.insert('1.0', sz)
    f.close()

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='A fájl tartalma:')
gomb1 = Button(abl1, text='Megnyit', command=megnyit)
import tkFont
doboz1 = Text(abl1, width=50, font=tkFont.Font(size=12))

# laptördelés a'grid' metódus segítségével :
txt1.grid(row=1, sticky =E)
gomb1.grid(row =3, sticky =E)
doboz1.grid(row =2)

# indítás :
abl1.mainloop()
```

A ***read***() metódus kiolvassa a fájlbeli adatokat és egy *karakterlánc* (***string***) típusú változóba teszi. Ha argumentum nélkül használjuk ezt a metódust, akkor az egész fájl tartalmát beolvassa.

*Megjegyzések:*

A **vboxvty9re.txt** annak a fájlnak a neve, amit olvasni akarunk. A fájlt megnyitó utasításnak szükségszerűen hivatkozni kell erre a névre. Ha nem létezik a fájl, akkor egy hibaüzenetet kapunk. Példa :

***IOError: [Errno 2] No such file or directory: 'vboxvty9re.txt'***

Viszont semmilyen kikötést sem tettünk a ***fájl objektum*** nevének megválasztására vonatkozóan. Ez egy tetszőleges változónév. Így az első utasításunkban egy *f* nevű ***fájl objektumot*** hoztunk létre, ami az olvasásra (**'r'** argumentum) megnyitott **vboxvty9re.txt** valódi fájlra hivatkozik.

A ***read***() metódust argumentummal is használhatjuk. Az argumentum azt adja meg, hogy hány karaktert kell beolvasni a fájlban már elért pozíciótól. Ha a fájlban nincs annyi karakter hátra, mint amennyit az argumentum megad, akkor az olvasás a fájl végén egyszerűen leáll. Ha a fájl végén vagyunk, akkor a ***read***() metódus egy üres ***stringet*** küld vissza.

## *Előre elkészített párbeszédes ablakok*

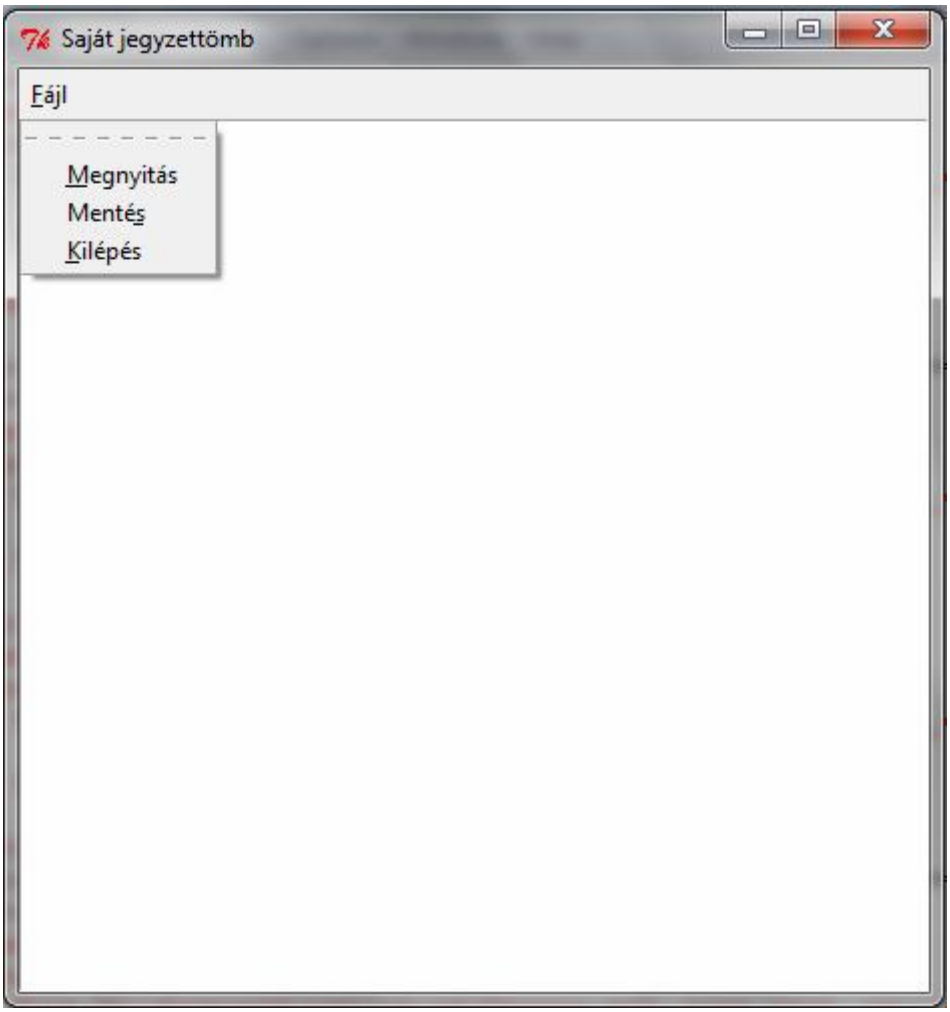
A ***tkFileDialog*** modul két különböző előugró ablakot tartalmaz, amit arra tudunk használni, hogy létező fájlokat találjunk meg, vagy új fájlokat hozzunk létre.

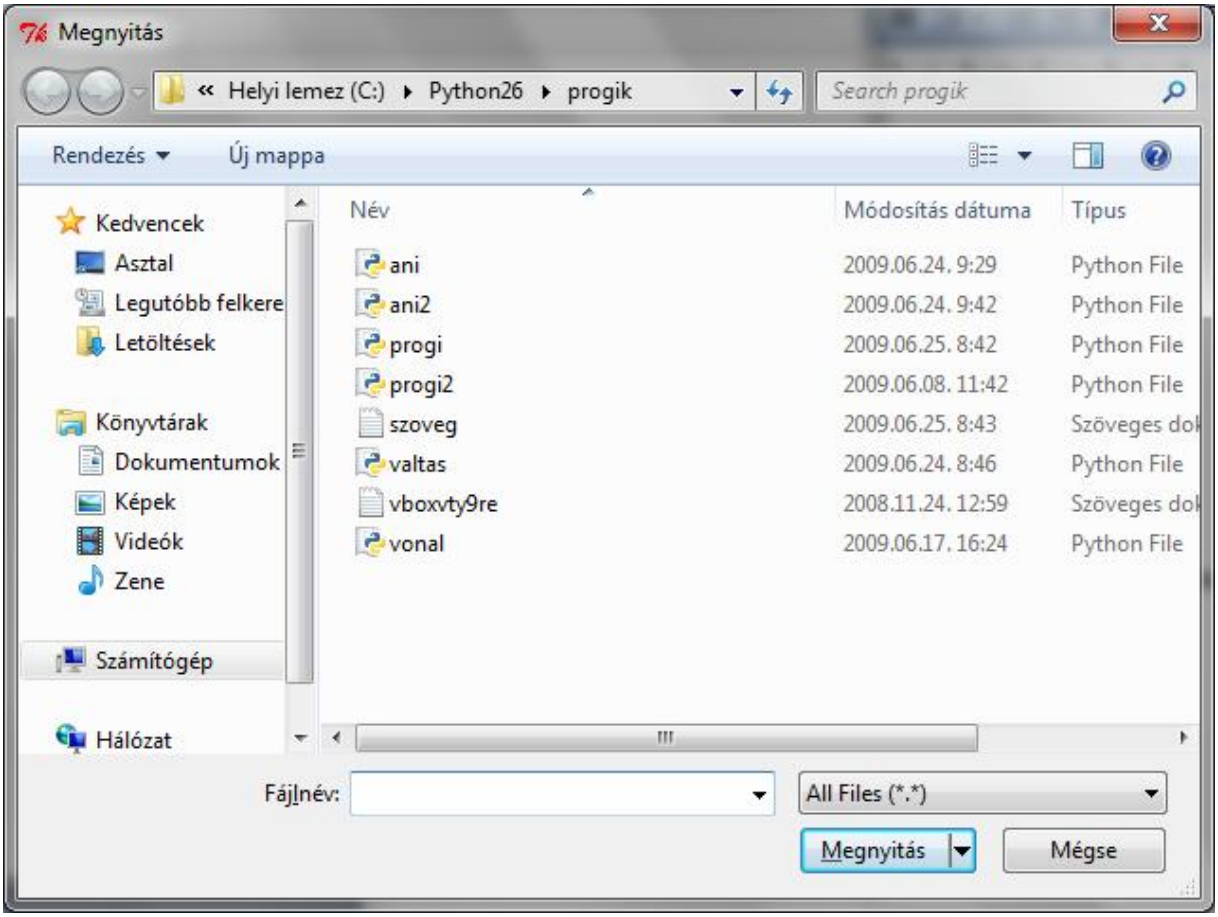
***askopenfilename*** : Akkor használatos, amikor a felhasználó ki akar választani egy létező fájlt. Ha a felhasználó egy nem létező fájlt választ ki, akkor egy ablak fog megjelenni, ami tájékoztat arról, hogy a kiválasztott fájl nem létezik.

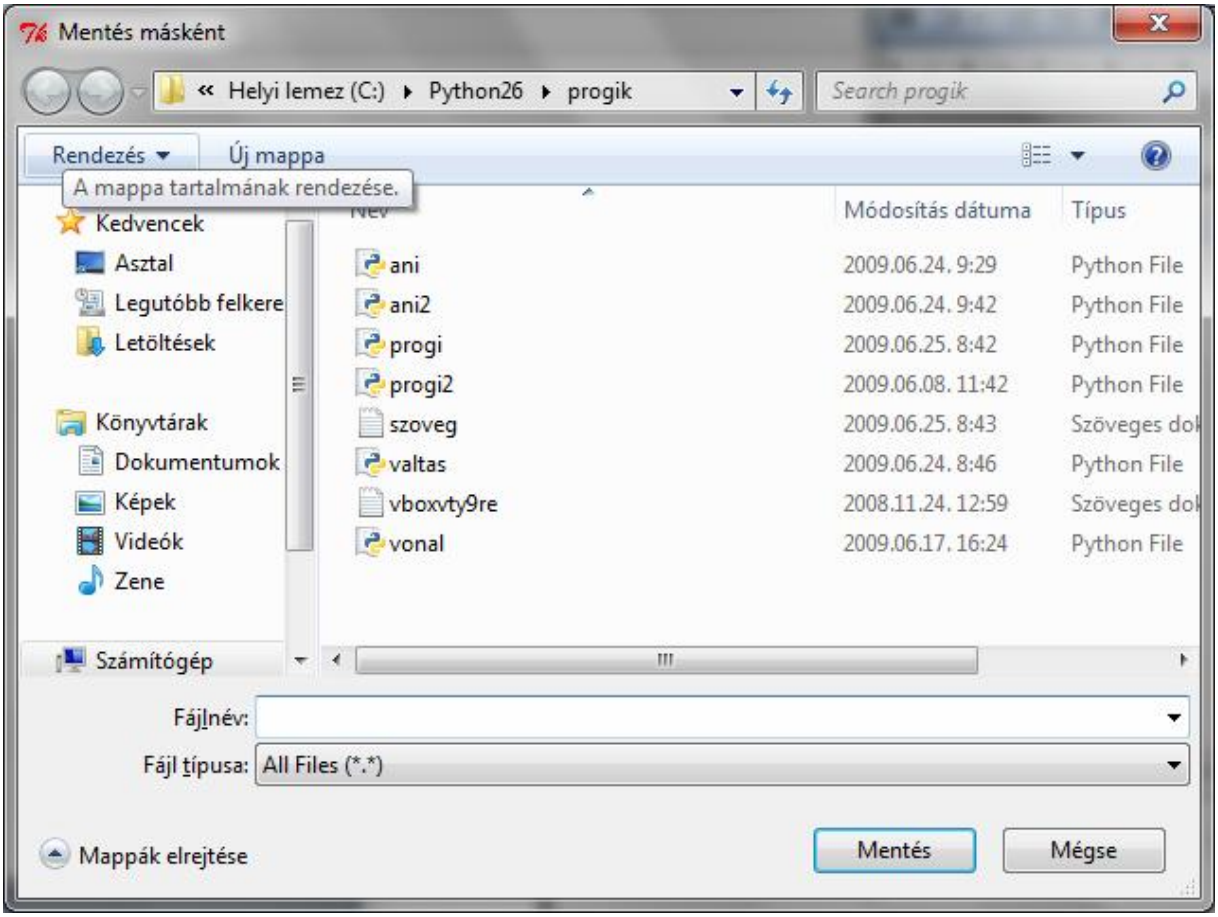
***asksaveasfilename*** : Akkor használatos, amikor a felhasználó létre akar hozni egy új fájlt vagy le akar cserélni egy létező fájlt. Ha a felhasználó egy létező fájlt választ ki, akkor egy ablak fog megjelenni, ami tájékoztat arról, hogy a fájl már létezik, és megkérdezi, hogy tényleg felül akarjuk-e írni.

***Feladat – Szövegszerkesztő***

Készítsünk olyan programot, amely egy szövegdobozba szöveget tud betölteni fájlból, a szövegdoboz tartalmát fájlba tudja menteni, a műveletekhez használja a megfelelő fájlablakokat!




Megnyitás
« Helyi lemez (C:) ▸ Python26 ▸ progik
Search progik
Rendezés
Új mappa
Kedvencek
Asztal
Legutóbb felkere
Letöltések
Könyvtárak
Dokumentumok
Képek
Videók
Zene
Számítógép
Hálózat
Név
Módosítás dátuma
Típus
ani 2009.06.24. 9:29 Python File
ani2 2009.06.24. 9:42 Python File
progi 2009.06.25. 8:42 Python File
progi2 2009.06.08. 11:42 Python File
szoveg 2009.06.25. 8:43 Szöveges dok
valtas 2009.06.24. 8:46 Python File
vboxvty9re 2008.11.24. 12:59 Szöveges dok
vonal 2009.06.17. 16:24 Python File
Fájlnév:
All Files (*.*)
Megnyitás
Mégse


Mentés másként
« Helyi lemez (C:) ▸ Python26 ▸ progik
Search progik
Rendezés
Új mappa
A mappa tartalmának rendezése.
Kedvencek
Asztal
Legutóbb felkere
Letöltések
Könyvtárak
Dokumentumok
Képek
Videók
Zene
Számítógép
Módosítás dátuma
Típus
ani 2009.06.24. 9:29 Python File
ani2 2009.06.24. 9:42 Python File
progi 2009.06.25. 8:42 Python File
progi2 2009.06.08. 11:42 Python File
szoveg 2009.06.25. 8:43 Szöveges dok
valtas 2009.06.24. 8:46 Python File
vboxvty9re 2008.11.24. 12:59 Szöveges dok
vonal 2009.06.17. 16:24 Python File
Fájlnév:
Fájl típusa: All Files (*.*)
Mappák elrejtése
Mentés
Mégse

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def megnyit():
    import tkFileDialog
    filenev = tkFileDialog.askopenfilename()
    f = open(filenev,'r')
    sz = f.read()
    doboz1.delete('1.0', END)
    doboz1.insert('1.0', sz)
    f.close()

def mentes():
    import tkFileDialog
    filenev = tkFileDialog.asksaveasfilename()
    f = open(filenev,'w')
    sz = doboz1.get('1.0', END+'-1c')
    f.write(sz.encode("utf-8"))
    f.close()

abl1 = Tk()
abl1.title('Saját jegyzettömb')

# a widgetek létrehozása:
menusor = Frame(abl1)
menusor.pack(side =TOP, fill =X)
menu1 = Menubutton(menusor, text ='Fájl', underline =0)
menu1.pack(side = LEFT )
fajl = Menu(menu1)
fajl.add_command(label ='Megnyitás', command = megnyit, underline =0)
fajl.add_command(label ='Mentés', command = mentes, underline =5)
fajl.add_command(label ='Kilépés', command = abl1.destroy, underline =0)
menu1.config(menu = fajl)

import tkFont
doboz1 = Text(abl1, width=50, font=tkFont.Font(size=12))
doboz1.pack()

# indítás :
abl1.mainloop()
```

## A ciklusból való kilépésre szolgáló break utasítás

Magától értetődik, hogy programhurkokra van szükségünk, amikor egy olyan fájlt kell kezelnünk, aminek nem ismerjük előre a tartalmát. Az alapelképzelés az, hogy részletekben olvassuk a fájt mindaddig, amíg el nem érjük a fájl végét.

A következő függvény illusztrálja ezt az elképzelést. Az egész fájlt – bármekkora is legyen a mérete – átmásolja egy másik fájlba 50 karakteres részletekben :

```
def fajltMasol(forras, cel):
    fs = open(forras, 'r')
    fd = open(cel, 'w')
    while 1:
            txt = fs.read(50)
            if txt =="":
                    break
            fd.write(txt)
    fs.close()
    fd.close()
    return
```

Ha ellenőrizni akarjuk a függvény működését, két argumentumot kell megadni : az első az eredeti fájl neve, a második a másolat neve. Példa:

```
fajltMasol('szoveg1.txt','szoveg2.txt')
```

Megjegyezhető, hogy a **while** ciklus ebben a függvényben máshogyan van megszerkesztve, mint amilyen formában az előzőekben találkoztunk vele. Tudjuk, hogy a **while** utasítást mindig egy kiértékelendő feltétel követi. Amíg ez a feltétel igaz, addig fog a **while** utasítást követő utasításblokk végrehajtódni. Itt pedig a kiértékelendő feltételt egy állandó helyettesíti. Azt is tudjuk, hogy a Python minden nullától különböző numerikus értéket igaznak tekint.

Az így megalkotott **while** ciklus végtelen ciklus, mert a folytatásának a feltétele mindig igaz. Ez azonban megszakítható a **break** utasítás hívásával, ami többféle kilépési mechanizmus elhelyezését teszi lehetővé ugyanabba a programhurokba :

```
while <feltétel 1> :
   --- különböző utasítások ---
   if <feltétel 2> :
      break
   --- különböző utasítások ---
   if <feltétel 3>:
      break
   stb.
```

Könnyű belátni, hogy a ***fajltMasol***() függvényünkben a **break** utasítás csak akkor fog végrehajtódni, ha elértük a fájl végét.

## *Szövegfájlok*

A szövegfájl egy olyan fájl, ami nyomtatható karaktereket és betűközöket tartalmaz egymást követő sorokba rendezve. A sorokat egy nem nyomtatható speciális karakter, a sorvége karakter választja el egymástól.

A stringekbe sorvége jelzéseket ( \n ) kell beszúrni azokra a helyekre, ahol el akarjuk egymástól választani a szövegsorokat. E nélkül a marker nélkül a karaktereket egymás után írnánk ki a fájlba.

Az olvasási műveletek alatt a szövegfájl sorai külön-külön nyerhetők vissza. A ***readline***() metódus például egyszerre csak egy sort olvas (beleértve a sorvége karaktert is), a ***readlines***() metódus az összes maradék sort egy stringekből álló listába teszi .

*Megjegyzések:*

A ***readlines***() metódus lehetővé teszi, hogy egyetlen utasítással olvassunk el egy egész fájlt. Ez azonban csak akkor lehetséges, ha az olvasandó fájl nem túl nagy. (Mivel teljes egészében be fogja másolni egy változóba, vagyis a számítógép operatív memóriájába, ezért a memória méretének megfelelően nagynak kell lennie.) Ha nagy fájlokat kell kezelnünk, inkább a ***readline***() metódust használjuk egy programhurokban, ahogyan azt a következő példa mutatja.

Jól jegyezzük meg, hogy a ***readline***() metódus egy karakterláncot ad vissza, míg a ***readlines***() metódusa egy listát. A fájl végén a ***readline***() egy üres stringet, míg a ***readlines***() egy üres listát ad vissza.

## *Kivételkezelés. A try – except – else utasítások*

A kivételek (*exceptions*) azok a műveletek, amiket az *interpreter* vagy a *compiler* akkor hajt végre, amikor a programvégrehajtás során hibát detektál. Általános szabályként a programvégrehajtás megszakad és egy többé-kevésbé explicit hibaüzenet jelenik meg.

A hibaüzenet két részből áll, amit : választ el. Elöl van a hiba típusa, utána egy - a hibára vonatkozó - specifikus információ következik.

Számos esetben előre lehet látni, hogy bizonyos hibák léphetnek fel a program egyik vagy másik részében. Ezekbe a programrészekbe beépíthetünk olyan speciális utasításokat, amik csak akkor aktiválódnak, ha ezek a hibák fellépnek. Az olyan magasszintű nyelvekben, mint amilyen a Python, lehetőség van arra, hogy egy felügyelő mechanizmust kössünk egy egész utasításcsoporthoz és így egyszerűsítsük azoknak a hibáknak a kezelését, melyek ezen utasítások bármelyikében felléphetnek.

Az ilyen mechanizmust általánosan kivételkezelő mechanizmusnak nevezik. A Python kivételkezelő mechanizmusa a **try** - **except** – **else** utasításcsoportot használja, ami lehetővé teszi egy hiba elfogását és egy - erre a hibára nézve specifikus - programrész végrehajtását. Ez a következő módon működik:

A **try** -t követő utasításblokkot a Python feltételesen hajtja végre. Ha az egyik utasítás végrehajtásakor hiba lép fel, akkor a Python törli a hibás utasítást és helyette az **except** -et követő kódblokkot hajtja végre. Ha semmilyen hiba sem lép fel a **try** utáni utasításokban, akkor az **else** -et követő kódblokkot hajtja végre (ha ez az utasítás jelen van). A program végrehajtása mindegyik esetben a későbbi utasításokkal folytatódhat.

Tekintsünk például egy programot, ami arra kéri a felhasználót, hogy adja meg egy fájl nevét, amit olvasásra kell megnyitni. Nem akarjuk, hogy a program „elszálljon”, ha a fálj nem létezik. Azt akarjuk, hogy írjon ki egy figyelmeztetést és a felhasználó esetleg megpróbálhasson beírni egy másik fájlnevet.

```
filename = raw_input("Írjon be egy fáljnevet : ")
try:
   f = open(filename, "r")
except:
   print "A fájl", filename, "nem létezik"
```

Ha úgy látjuk, hogy ez a fajta teszt alkalmas arra, hogy a program más részein is felhasználjuk, akkor egy függvénybe ágyazhatjuk:

```
def letezike(fname):
    try:
        f = open(fname,'r')
```

```
        f.close()
        return 1
    except:
        return 0
filename = raw_input("Írjon be egy fájlnevet: ")
if letezike(filename):
    print "Ez a file létezik."
else:
    print "A fájl", filename, "nem létezik."
```

Az is lehetséges, hogy a **try** -t több **except** blokk kövesse, melyek mindegyike egy specifikus hibatípust kezel.

**FELADATOK**

**1. Írjunk egy programot, ami lehetővé teszi egy szövegfájl kényelmes olvasását. A program először kérje a felhasználótól a fájl nevét. Ezután ajánlja föl a következő választást: vagy új szövegsorokat rögzít, vagy kiírja a fájl tartalmát.**

**2. Tegyük fel, hogy rendelkezésére áll egy szövegfájl, ami különböző hosszúságú mondatokat tartalmaz. Írjunk egy programot, ami megkeresi és kiírja a leghosszabb mondatot.**

**3. Írjunk egy programot, ami automatikusan létrehoz egy szövegfájlt, ami a 2 – 30 -as szorzótáblákat tartalmazza (mindegyik szorzótábla csak 20 tagot tartalmazzon).**

**4. Írjunk egy programot, ami úgy másol át egy fájlt, hogy a szavak között megháromszorozza a szóközök számát.**

**5. Rendelkezésünkre áll egy szövegfájl, aminek minden sora egy valós típusú numerikus érték reprezentációja (exponens nincs). Például:**
**14.896**
**7894.6**
**123.278**
**stb.**
**Írjunk egy programot, ami ezeket az értékeket egész számra kerekítve egy másik fájlba másolja (a kerekítésnek korrektnek kell lenni).**

**7. Írjunk egy programot, ami lehetővé teszi egy olyan szövegfájl rögzítését, mely különböző személyek vezetéknevét, keresztnevét, címét, postai irányítószámát és telefonszámát fogja tartalmazni (gondoljunk például arra, hogy egy klub tagjairól van szó).**

**8. Írjunk egy programot, ami az előző gyakorlatban használt fájlt másolja át úgy, hogy a személyek születési dátumát és nemét hozzáfűzi (a számítógépnek egyesével kell kiírni a sorokat és a felhasználótól kérni minden egyes kiegészítő adat beírását).**

**9. Tegyük fel, hogy elvégeztük az előző gyakorlatot, és most van egy olyan fájlunk, ami bizonyos számú személy adatait tartalmazza. Írjunk egy programot, ami lehetővé teszi, hogy kiszedjük ebből a fájlból azokat a sorokat, amik egy adott postai irányítószámnak felelnek meg.**

**10. Módosítsuk az előző gyakorlat programját úgy, hogy azokat a sorokat találja meg, melyek azoknak a személyeknek felelnek meg, akik nevének kezdő betűje az F és M között van az ABC-ben.**

# 2. Programozási tételek

A programozási feladatok megoldásuk szempontjából nagy csoportokba, osztályokba sorolhatók. A programozási tételek ilyen csoportok tipikus megoldásaival foglalkoznak.

Olyan feladatokkal foglalkozunk, amelyekben

- egy sorozathoz egy értéket,
- egy sorozathoz egy sorozatot,
- sorozatokhoz sorozatot vagy
- sorozathoz sorozatokat kell rendelni.

A vizsgálni kívánt sorozatot megadhatjuk elemei felsorolásával vagy elemei kiszámítási módjával.

***Összegzés***: (a sorozat elemeinek összege, szorzata, uniója stb.),
***Maximum-kiválasztás:*** (a sorozat maximális, vagy minimális értékű elemének kiválasztása),
***Megszámolás:*** (adott tulajdonságú elemek száma a sorozatban),
***Kiválogatás:*** (sorozat adott tulajdonságú elemeinek megadása),
***Kiválasztás:*** (a vizsgált sorozat egy adott tulajdonságú elemének megadása, ilyen elem biztosan szerepel a sorozatban),
***Eldöntés:*** (van-e a vizsgált sorozatban adott tulajdonságú elem),
***Keresés*** (a vizsgált sorozat egy adott tulajdonságú elemének megadása, ilyen elem lehet, hogy nincs a sorozatban),
***Unió:*** (két sorozatként ábrázolt halmaz egyesítése),
***Metszet:*** (két sorozatként ábrázolt halmaz közös elemei),
***Összefuttatás:*** (két rendezett sorozat egyesítése),
***Szétválogatás:*** (egy sorozat kétféle tulajdonságú elemeinek különválasztása),
***Rendezés:*** (rendezetlen sorozat elemeinek átrendezése növekvő vagy csökkenő sorrendbe),
***Visszalépéses keresés*** (sorozatokból egy-egy elem meghatározása).

## *Összegzés*

Ez a feladattípus egy sorozathoz egy értéket rendel. A sorozat elemeinek összegét, szorzatát, unióját stb. kell előállítani. A feladatok egy részénél az így kiszámított értékkel esetleg még egyszerű műveletet kell végezni (pl. osztani az átlagszámításnál).

***Feladat – Összegzés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számok összegét számítja ki!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')

#Az összeg kezdőértékét nullára állítjuk
osszeg = 0
while 1:
   sor = f.readline()
   if sor == "":
      break
   #Az összeget növeljük a soron következő értékkel
   osszeg = osszeg + eval(sor)
f.close()
```

```
print "A számok összege: ", osszeg
```

## Maximum-kiválasztás

Ebben a feladatkörben adott egy *N* elemű sorozat. A feladat ezen sorozat legnagyobb elemének meghatározása (néha csupán az értékére van szükség). Hasonló feladat - csupán a < relációt kell >-ra cserélni - a minimum-kiválasztás.

***Feladat – Maximum-kiválasztás***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számok közül a legnagyobbat választja ki!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')

#Az első érték kiemelése
sor = f.readline()
maximum = eval(sor)
while 1:
   sor = f.readline()
   if sor == "":
      break
   ertek = eval(sor)
   #Ha a soron következő érték nagyobb, akkor ő legyen a maximális
   if maximum < ertek :
      maximum = ertek
f.close()
print "A számok maximuma: ", maximum
```

## Megszámolás

A következő feladatok megoldása során egy sorozathoz egy számot rendelünk hozzá. Rendelkezésünkre áll egy *N* elemű sorozat és egy, a sorozat elemein értelmezett *T* tulajdonság. Az a feladatunk, hogy a T tulajdonsággal rendelkező elemeket számoljuk meg. Ügyeljünk arra, hogy a megszámolás céljára szolgáló változó értékét kezdetben 0-ra kell állítani!

***Feladat – Megszámolás***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számok közül megszámolja a párosakat!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
parosak = 0
while 1:
   sor = f.readline()
   if sor == "":
      break
```

```
   ertek = eval(sor)
   #Ha az adott tulajdonság teljesül, akkor növeljük a darabszámot
   if ertek % 2 == 0 :
      parosak = parosak + 1
f.close()
print "A párosak száma: ", parosak
```

## Kiválogatás

Ennél a feladattípusnál egy *N* elemű *A* sorozat összes *T* tulajdonsággal rendelkező elemét kell meghatározni. Az eredmény egy *B*(*M*) sorozat lesz, ahol 0<=*M*<=*N*.

*M*=0, ha az *A*(*N*) sorozat egy eleme sem, *M*=*N*, ha az *A*(*N*) sorozat minden eleme *T* tulajdonságú.

***Feladat – Kiválogatás***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számok közül a párosakat egy másik szövegfájlba gyűjti!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
g = open('parosak.txt','w')
while 1:
   sor = f.readline()
   if sor == "":
      break
   ertek = eval(sor)
   if ertek % 2 == 0 :
      g.write(sor)
f.close()
g.close()
print "A kiválogatás megtörtént."
```

## Kiválasztás

Adott egy *N* elemű sorozat és egy, a sorozat elemein értelmezett *T* tulajdonság. Azt is tudjuk, hogy a sorozatban van legalább egy *T* tulajdonságú elem. A feladat az első ilyen elem meghatározása.

***Feladat – Kiválasztás***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számok közül kiírja az első párosat!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
while 1:
   sor = f.readline()
   ertek = eval(sor)
   if ertek % 2 == 0:
```

```
        print "Az első páros szám: ", ertek
        break
f.close()
```

## Eldöntés

Ez a feladattípus egy sorozathoz logikai értéket rendel. A logikai érték "igaz"-at vesz fel, ha a sorozatban létezik adott (*T*) tulajdonságú elem, és "hamis"-at vesz fel, ha ilyen tulajdonságú elem nincs a sorozatban.

***Feladat – Kiválasztás***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számokról eldönti, hogy van-e közöttük páros!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
while 1:
    sor = f.readline()
    if sor == "":
        print "Páros szám nem található"
        break
    ertek = eval(sor)
    if ertek % 2 == 0:
        print "Van közöttük páros"
        break
f.close()
```

## Keresés

A keresés feladattípus egy sorozathoz egy elemet rendel. A feladat lényege, hogy a sorozatban egy adott (*T*) tulajdonságú elemet kell meghatároznunk. Nem biztos, hogy a sorozatban van ilyen elem.

A keresés hatékonyságát lényegesen befolyásolhatja, hogy rendezett vagy rendezetlen sorozatban keresünk: logaritmikus, illetve lineáris keresést használunk. Amelyik feladatban lehet, használjuk ki a sorozat rendezettségét!

***Feladat – Lineáris keresés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számok közül kiírja a keresett szám sorszámát!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
x = int(raw_input("A keresett érték: "))
i = 1
while 1:
    sor = f.readline()
    if sor == "":
        print "Nem található"
```

```
        break
    ertek = eval(sor)
    if ertek == x:
        print "A sorszáma: ", i
        break
    i = i + 1
f.close()
```

***Feladat – Logaritmikus keresés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található, nagyság szerint növekvő sorrendbe rendezett számok közül kiírja a keresett szám sorszámát!

```
# -*- coding: ISO-8859-2 -*-

szamok = []
f = open('rendezett.txt','r')
x = int(raw_input("A keresett érték: "))

while 1:
    sor = f.readline()
    if sor == "":
        break
    szamok.append(sor)

f.close()

alsoH = 0
felsoH = len(szamok)

while 1:
    kozep = (alsoH + felsoH) / 2
    if x < eval(szamok[kozep]):
        felsoH = kozep - 1
    if x > eval(szamok[kozep]):
        alsoH = kozep + 1
    if x == eval(szamok[kozep]):
        print "A sorszám: ", kozep + 1
        break
    if alsoH > felsoH :
        print "Nem található."
        break
```

## *Unió*

Eddig olyan feladattípusokkal foglalkoztunk, ahol egy sorozathoz egy értéket, vagy egy sorozatot rendeltünk hozzá. Ebben a részben viszont több sorozat adott, amelyből egy sorozatot kell valamilyen szempont szerint előállítani.

A matematikából ismerjük két halmaz egyesítésének (uniójának) fogalmát. A két halmaz egyesítéséből származó halmazba azok az elemek tartoznak, amelyek legalább az egyikben szerepelnek. Ebben, és a következő alfejezetben a halmazt speciálisan ábrázoljuk: az

elemeit felsoroljuk egy sorozatban. Ezt a sorrendet felhasználjuk a halmaz elemeinek bejárására.

Például, ha
az "*A*" sorozat elemei: e, b, a, f, d, c és
a "*B*" sorozat elemei: j, f, b, h, d,
akkor a két sorozat uniójába ("*C*") az e, b, a, f, d, c, j, h
elemek tartoznak.

***Feladat – Unió***
Készítsünk olyan programot, amely két szövegfájl soraiban található sorokat egyesít egy harmadik szövegfájlba!

```
# -*- coding: ISO-8859-2 -*-

f = open('egyik.txt','r')
g = open('masik.txt','r')
h = open('eredmeny.txt','w')

while 1:
   sor = f.readline()
   if sor == "":
      break
   h.write(sor)

while 1:
   sor = g.readline()
   if sor == "":
      break
   h.write(sor)

f.close()
g.close()
h.close()

print "Az egyesítés megtörtént."
```

## Metszet

A metszetképzés fogalmával találkoztunk már a matematika órán a halmazműveletek tanulása során. Két halmaz metszetébe azok az elemek tartoznak, amelyek mindkettőben szerepelnek.

Például, ha
az "*A*" halmaz elemei: e, b, a, f, d, c és
a "*B*" halmaz elemei: a, j, f, b, h, d,
akkor a két halmaz metszetébe ("*C*") a b, f, d elemek tartoznak.

***Feladat – Metszet***
Készítsünk olyan programot, amely két szövegfájl soraiban található számok közül a mindkettőben egyidejűleg szereplőket egy harmadik szövegfájlba írja!

```
# -*- coding: ISO-8859-2 -*-
```

```
f = open('egyik.txt','r')
g = open('masik.txt','r')
h = open('eredmeny.txt','w')
egyik = []
masik = []

while 1:
   sor = f.readline()
   if sor == "":
      break
   egyik.append(eval(sor))

while 1:
   sor = g.readline()
   if sor == "":
      break
   masik.append(eval(sor))

for i in range(len(egyik)):
   j = 1
   while (j < len(masik)) and (egyik[i] != masik[j]) :
      j = j + 1
   if j < len(masik):
      h.write(str(egyik[i])+'\n')

f.close()
g.close()
h.close()

print "A metszet létrehozva."
```

## *Összefuttatás*

Az alapelve megegyezik az unióképzéssel, de feltételezzük, hogy a két sorozat rendezett, és azt szeretnénk, hogy az egyesítés során keletkezett sorozat is rendezett legyen.

Például, ha
az *A* sorozat elemei: A, B, C, D, E, F és
a *B* sorozat elemei: B, D, F, H, J,
akkor a két sorozat összefuttatásával keletkezett sorozatba az A, B, C, D, E, F, H, J elemek tartoznak.

***Feladat – Összefuttatás***
Készítsünk olyan programot, amely két szövegfájl soraiban rendezetten található számokat egy harmadik szövegfájlba egyesít úgy, hogy a rendezettség is megmarad!

```
# -*- coding: ISO-8859-2 -*-

f = open('egyik.txt','r')
g = open('masik.txt','r')
```

```
h = open('eredmeny.txt','w')
egyik = []
masik = []

#Az egyik fájl elemeinek listába olvasása
while 1:
   sor = f.readline()
   if sor == "":
      break
   egyik.append(eval(sor))

#A másik  fájl elemeinek listába olvasása
while 1:
   sor = g.readline()
   if sor == "":
      break
   masik.append(eval(sor))

i = 0
j = 0

while 1:
   #Ha a soron következő elemek közül az elsőben kisebb az érték
   if egyik[i] < masik[j] :
      h.write(str(egyik[i])+'\n')
      i = i + 1
   #Ha a soron következő elemek közül a másodikban kisebb az érték
   elif egyik[i] > masik[j] :
      h.write(str(masik[j])+'\n')
      j = j + 1
   #Ha a soron következő elemek egyenlők
   else :
      h.write(str(egyik[i])+'\n')
      i = i + 1
      j = j + 1
   #Ha valamelyik sorozatnak elértük a végét
   if (i >= len(egyik)) or (j >= len(masik)) :
      break

#Ha a sorozatokból kimaradtak a túl nagy elemek
while i < len(egyik):
   h.write(str(egyik[i])+'\n')
   i = i + 1

while j < len(masik):
   h.write(str(masik[j])+'\n')
   j = j + 1

f.close()
g.close()
```

```
h.close()

print "Az összefuttatás létrehozva."
```

## Szétválogatás

Ebben a részben egy sorozat elemeit választjuk külön vagy egy sorozaton belül két részre, vagy két különböző sorozatba aszerint, hogy egy adott *T* tulajdonsággal rendelkeznek-e az elemek, vagy nem.

Például, ha az *A* sorozat elemei 1, 2, 3, 4, 5, 6, 7, 8, 9, és a *T* tulajdonság: a számok párossága, akkor a sorozat elemeinek szétválogatásával a 2, 4, 6, 8, és az 1, 3, 5, 7, 9 sorozatok keletkeznek.

***Feladat – Szétválogatás***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számokat szétválogat két új szövegfájlba aszerint, hogy párosak vagy nem!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
g = open('parosak.txt','w')
h = open('paratlanok.txt','w')

while 1:
    sor = f.readline()
    if sor == "":
        break
    szam = eval(sor)
    if szam % 2 == 0 :
        g.write(sor)
    else:
        h.write(sor)

f.close()
g.close()
h.close()

print "A szétválogatás megtörtént."
```

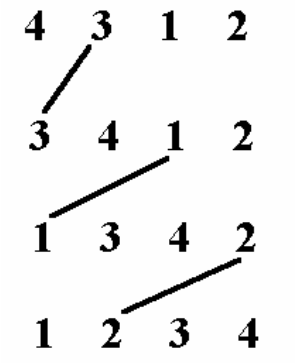
## Rendezések

Ebben a részben négyféle rendezést ismerünk meg, ezek a következők:

Közvetlen beszúrás
Közvetlen kiválasztás
Buborék-rendezés (közvetlen csere)
Keverő rendezés

## *Közvetlen beszúrás*

Ennél a rendezési módszernél minden lépésben a második elemtől egyesével kiemeljük a csökkenő sorozat szerinti első elemet, és beszúrjuk a megfelelő helyre

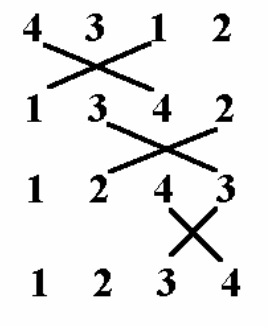



***Feladat – Közvetlen beszúrásos rendezés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számokat a közvetlen beszúrás módszerével rendezi, és egy új szövegfájlba helyezi!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
g = open('rendezett.txt','w')

szamok = []
while 1:
    sor = f.readline()
    if sor == "":
        break
    szamok.append(eval(sor))

i = 1
while i < len(szamok):
    elem = szamok[i]
    j = i - 1
    while (j >= 0) and (elem < szamok[j]) :
        szamok[j+1] = szamok[j]
        j = j - 1
    szamok[j+1] = elem
    i = i + 1

for i in range(len(szamok)):
    g.write(str(szamok[i])+'\n')

f.close()
g.close()

print "A rendezés megtörtént."
```

## *Közvetlen kiválasztás*

A rendezési eljárás a sorozat soron következő legkisebb elemét választja ki, majd beszúrja a megfelelő helyre.



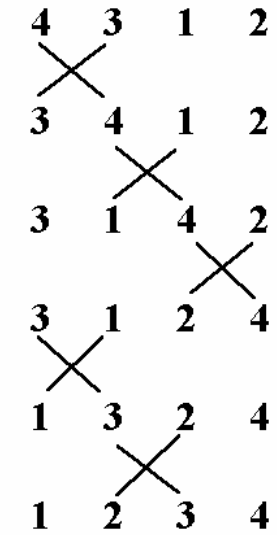

***Feladat – Közvetlen kiválasztásos rendezés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számokat a közvetlen kiválasztás módszerével rendezi, és egy új szövegfájlba helyezi!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
g = open('rendezett.txt','w')

szamok = []
while 1:
   sor = f.readline()
   if sor == "":
      break
   szamok.append(eval(sor))

i = 0
while i < len(szamok)-1:
   k = i
   elem = szamok[i]
   j = i + 1

   while j < len(szamok):
      if szamok[j] < elem:
         k = j
         elem = szamok[j]
      j = j + 1

   szamok[k] = szamok [i]
   szamok[i] = elem
   i = i + 1


for i in range(len(szamok)):
   g.write(str(szamok[i])+'\n')

f.close()
```

```
g.close()

print "A rendezés megtörtént."
```

## Közvetlen csere (buborék)

A egymást követő elempárok összehasonlítása és felcserélése során jutunk az összes elem rendezéséhez.



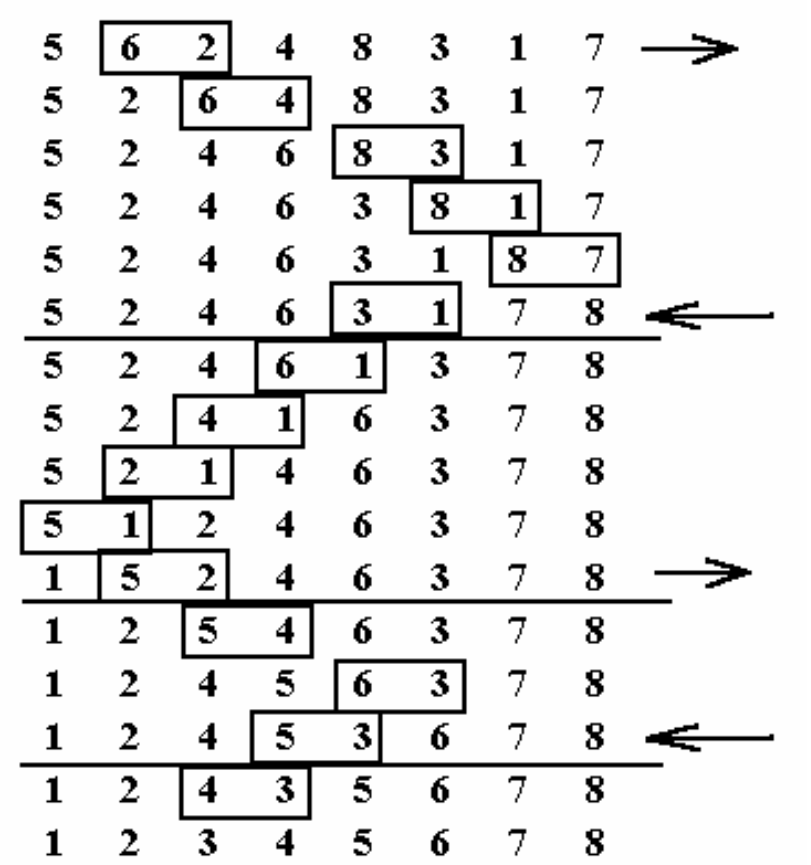

***Feladat – Buborék-rendezés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számokat a közvetlen csere módszerével rendezi, és egy új szövegfájlba helyezi!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
g = open('rendezett.txt','w')

szamok = []
while 1:
   sor = f.readline()
   if sor == "":
      break
   szamok.append(eval(sor))

i = 1
while i < len(szamok):
   j = len(szamok)-1

   while j >= i :
      if szamok[j-1] > szamok[j]:
         elem = szamok[j-1]
         szamok[j-1] = szamok[j]
         szamok[j] = elem
      j = j - 1
   i = i + 1
```

```
for i in range(len(szamok)):
    g.write(str(szamok[i])+'\n')

f.close()
g.close()

print "A rendezés megtörtént."
```

## Keverő rendezés

Az előző módszer elég rossz hatásfokú, ha kevés rossz helyen tartózkodó elem van, mert akkor is elvégzi az összes összehasonlítást. A keverő rendezés ezen úgy javít, hogy váltogatja az összehasonlító menetek irányát.




***Feladat – Keverő rendezés***
Készítsünk olyan programot, amely egy szövegfájl soraiban található számokat keverő rendezéssel rendezi, és egy új szövegfájlba helyezi!

```
# -*- coding: ISO-8859-2 -*-

f = open('szamok.txt','r')
g = open('rendezett.txt','w')

szamok = []
while 1:
    sor = f.readline()
```

```
   if sor == "":
      break
   szamok.append(eval(sor))

l = 1
r = len(szamok)-1
k = len(szamok)-1
while 1:
   j = r
   while j >= l:
      if szamok[j-1]>szamok[j]:
         elem = szamok[j-1]
         szamok[j-1] = szamok[j]
         szamok[j] = elem
         k = j
      j = j - 1
   l = k + 1
   j = l
   while j <= r:
      if szamok[j-1]>szamok[j]:
         elem = szamok[j-1]
         szamok[j-1] = szamok[j]
         szamok[j] = elem
         k = j
      j = j + 1
   r = k - 1
   if l > r:
      break

for i in range(len(szamok)):
   g.write(str(szamok[i])+'\n')

f.close()
g.close()

print "A rendezés megtörtént."
```

## *Visszalépéses keresés (backtrack)*

A visszalépéses keresés ( backtrack ) a problémamegoldás igen széles területén alkalmazható algoritmus, amelynek lényege a feladat megoldásának megközelítése rendszeres próbálgatással. Néha ez a legjobb megoldás.

Adott *N* sorozat, amelyek rendre *M*(1), *M*(2),...*M*(*N*) elemszámúak. Ki kell választani mindegyikből egy-egy elemet úgy, hogy az egyes sorozatokból való választások másokat befolyásolnak. Ez egy bonyolult keresési feladat, amelyben egy adott tulajdonsággal rendelkező szám *N*-est kell megadni úgy, hogy ne kelljen az összes lehetőséget végignézni.

Először megpróbálunk az első sorozatból kiválasztani egy elemet, ezután a következőből, s ezt addig csináljuk, amíg választás lehetséges. *X*(*I*) jelölje az *I*. sorozatból kiválasztott elem sorszámát! Ha még nem választottuk, akkor értéke 0 lesz. Ha nincs jó választás, akkor visszalépünk az előző sorozathoz, és megpróbálunk abból egy másik elemet

választani. Visszalépésnél természetesen törölni kell a választást abból a sorozatból, amelyikből visszalépünk.

Az eljárás akkor ér véget, ha minden sorozatból sikerült választani, vagy pedig a visszalépések sokasága után már az első sorozatból sem lehet újabb elemet választani (ekkor a feladatnak nincs megoldása).

**FELADATOK**

**1. Az indiánok 1627-ben eladták Manhattan szigetét 24 dollárért. Mennyit érne ez az összeg ma 5 százalékos kamat mellett? Mikor ér 1 millió dollárt?**

**2. Határozzuk meg, hogy egy mondatban hány e betű van!**

**3. Adott szöveg hány szótagból áll?**

**4. Adott szöveg hány szóból áll?**

**5. Pénzérmét dobunk fel 1000-szer, hány fej és hány írás?**

**6. Kockával dobunk 1000-szer, mi a dobások eloszlása?**

**7. Adottak a tanulók átlag szerinti csökkenő sorrendben. Bekérjük az átlagot kiírja a hozzá tartozó nevet.**

**8. Keverő rendezés**
**Adott 20 egész szám, írd fel közülük az 5 legnagyobbat csökkenő sorrendben! Számítsuk ki az összegüket!**

**9. Buborék rendezés**
**Adott 20 egész szám. írd fel közülük az 5 legkisebbet növekvő sorrendben! Számítsuk ki az átlagukat!**

**10. Közvetlen kiválasztásos rendezés**
**Adott 20 egész szám, írd fel közülük a párosakat növekvő sorrendben! Számítsuk ki az összegüket!**

**11. Közvetlen beszúrásos rendezés**
**Adott 20 egész szám, írd fel közülük a páratlanokat csökkenő sorrendben! Számítsuk ki az átlagukat!**

**12. Szétválogatás**
**Adott 30 irányítószám, válogasd szét a Budapestieket és a nem Budapestieket!**

**13. Metszet**
**Adott két szó. Írd ki azokat a betűket, amelyek mindkét szóban szerepelnek!**

**14. Logaritmikus keresés**
**Egy 20 irányítószámból álló sorozatban rendezés után írjuk ki, hogy hányadik Kiskunlacháza irányítószáma!**

**15. Lineáris keresés**
**Egy 20 irányítószámból álló sorozatban rendezés után írjuk ki, hogy hányadik Ráckeve irányítószáma!**

# 3. Adatbázis-kezelés

Az adatbázisok egyre gyakrabban alkalmazott eszközök. Nagymennyiségű adat tárolását teszik lehetővé egyetlen jól strukturált halmazban. Relációs adatbázisok esetén egyebek között lehetőség van a duplikált adatok elkerülésére is.

A Python számos rendszer alkalmazását megengedi, mi a MySQL –t fogjuk használni.

## *Adatbázisok*

Számos adatbázistípus létezik. Például már elemi adatbázisnak tekinthetünk egy nevek és címek listáját tartalmazó fájlt.

Ha a lista nem túl hosszú, és ha nem akarunk benne összetett feltételektől függő kereséseket végrehajtani, akkor magától értetődik, hogy ehhez az adattípushoz olyan egyszerű utasításokkal férhetünk hozzá, mint amilyeneket korábban tárgyaltunk.

A helyzet azonban nagyon gyorsan komplikálttá válik, ha kiválasztásokat és rendezéseket hajtunk végre, különösen, ha ezek száma megnő. A nehézségek még tovább nőnek, ha az adatok különböző táblákban vannak, amiket hierarchikus relációk kapcsolnak össze, és ha több felhasználónak egyidejűleg kell tudni hozzájuk férni.

## *Relációs adatbázis-kezelő rendszerek – A kliens/szerver modell*

Az ilyen komplex adatok hatékony kezelésére alkalmas programok maguk is szükségszerűen összetettek. Ezeket a programokat relációs adatbázis kezelő rendszereknek hívják (RDBMS = Relational Database Management Systems). Közülük egyesek speciálizálódott vállalatok (IBM, Oracle, Microsoft, Informix, Sybase...) termékei és általában nagyon drágák. Másokat (PostgreSQL, MySQL ...) kutatási központokban vagy az egyetemi oktatásban fejlesztettek ki; ezek általában teljesen ingyenesek.

E rendszerek mindegyike sajátos jellemzőkkel és teljesítménnyel rendelkezik, azonban a többségük működése a kliens/szerver modellen alapul. Ez azt jelenti, hogy az alkalmazás legnagyobb része (valamint az adatbázis) egyetlen helyre, elvileg egy nagyteljesítményű gépre van telepítve (ez az együttes alkotja a szervert), míg a másik jóval egyszerűbb rész meghatározatlan számú munkaállomásra van telepítve (ezeket hívjuk klienseknek).

A kliensek különböző eljárások vagy protokollok (esetleg az internet) révén állandóan vagy ideiglenesen a szerverhez vannak kapcsolva. Mindegyikük hozzáférhet az adatok több-kevesebb részéhez, egyes adatokat engedéllyel vagy anélkül módosíthatnak, újakat vihetnek be, törölhetnek a jól meghatározott hozzáférési jogoktól függően. (Ezeket a jogosultságokat egy adatbázis adminisztrátor definiálja).

A szerver és kliensei különálló alkalmazások, amik információt cserélnek. Képzeljük például el, hogy egyike vagyunk egy rendszer felhasználóinak. Ahhoz, hogy hozzáférjünk az adatokhoz, valamelyik munkaállomáson el kell indítanunk egy kliensalkalmazást. A kliensalkalmazás azzal kezdi, hogy kapcsolatot létesít a szerverrel és az adatbázissal. Amikor létrejött a kapcsolat, a kliensalkalmazás megfelelő formájú kérés (request) küldésével lekérdezheti a szervert. Például egy meghatározott információt kell megkeresni. A szerver a megfelelő adatok adatbázisban történő keresésével végrehajtja a kérést, majd valamilyen

választ küld vissza a kliensnek.

Ez a válasz lehet a kért információ, vagy hiba esetén egy hibaüzenet.

A kliens és a szerver közötti kommunikációt tehát kérések és válaszok alkotják. A kérések a klienstől a szervernek küldött valódi utasítások, amik nemcsak adatok kinyerésére, hanem adatok beszúrására, törlésére, módosítására, stb. is szolgálnak.

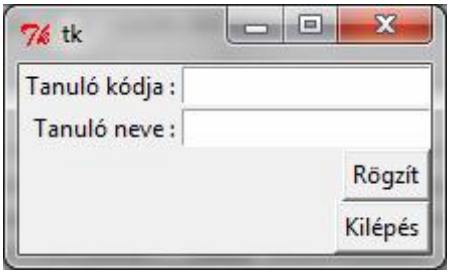
## *MySQL kliensprogram*

A feladatok elvégzése során feltételezzük, hogy a *root* felhasználó jelszava *pingvin*, és a táblák a *próba* adatbázisban vannak.

***Feladat – Tábla létrehozása***
Készítsünk olyan programot, amely létrehoz egy Tanulo nevű táblát (Tankod Integer, TanNev Text(20)) típusú mezőkkel!

```
# -*- coding: ISO-8859-2 -*-

import MySQLdb, sys

#A kapcsolat kiépítése
adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost', port=3306)

#A parancs tárolása sztringben
req = "create table Tanulo(Tankod integer, TanNev Text(20))"

#A parancs futtatása
adatBazis.cursor().execute(req)

print "A tábla létrejött."
```

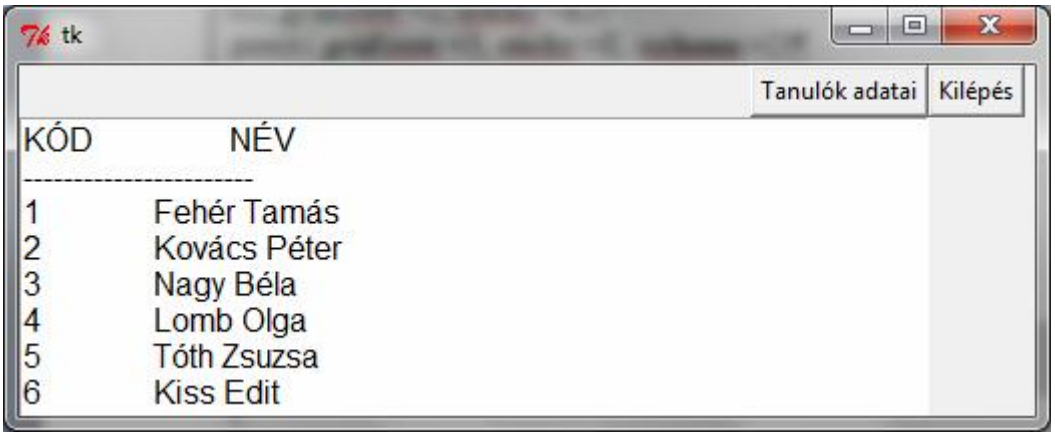
***Feladat – Adatbeviteli űrlap készítése***
Készítsünk adatbeviteli űrlapot az előző táblához az adatok rögzítésére!




```
# -*- coding: ISO-8859-2 -*-

from Tkinter import *
import MySQLdb, sys

#A kapcsolat kiépítése
```

```
adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost', port=3306)

abl1 = Tk()

def rogzit():
    kod = mezo1.get()
    nev = mezo2.get()
    #A parancs tárolása sztringben
    req = "insert into Tanulo values('"+kod+"','"+nev+"')"
    #A parancs futtatása
    adatBazis.cursor().execute(req)
    #Kiírás a lemezre
    adatBazis.commit()
    mezo1.delete(0,END)
    mezo2.delete(0,END)

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Tanuló kódja :')
txt2 = Label(abl1, text ='Tanuló neve :')
gomb1 = Button(abl1, text='Rögzít', command=rogzit)
gomb2 = Button(abl1, text='Kilépés', command=abl1.destroy)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
gomb1.grid(row =3, sticky =E, column =2)
gomb2.grid(row =4, sticky =E, column =2)
mezo1.grid(row =1, column =2)
mezo2.grid(row =2, column =2)

# indítás :
abl1.mainloop()
```

***Feladat – Lekérdezés végrehajtása***
Kérdezd le az összes tanuló összes adatát!



```
# -*- coding: ISO-8859-2 -*-

from Tkinter import *
import MySQLdb, sys

#A kapcsolat kiépítése
adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost',
port=3306)
cursor = adatBazis.cursor()

abl1 = Tk()

def lekerdez():
   #A parancs tárolása sztringben
   req = "select * from tanulo"
   #A parancs futtatása
   cursor.execute(req)
   #Az eredmény tárolása
   records = cursor.fetchall()
   #A lekérdezés eredményének fájlba írása
   f = open('lekerdezes.txt','w')
   f.write('KÓD              NÉV'+'\n')
   f.write('----------------------'+'\n')
   for rec in records:
      for item in rec:
         f.write(str(item)+'            ')
      f.write('\n')
   f.close()
   #Megjelenítés a szövegdobozban
   f = open('lekerdezes.txt','r')
   sz = f.read()
   doboz1.delete('1.0', END)
   doboz1.insert('1.0', sz)
   f.close()

# a widgetek létrehozása:
gomb1 = Button(abl1, text='Tanulók adatai', command=lekerdez)
gomb2 = Button(abl1, text='Kilépés', command=abl1.destroy)
import tkFont
doboz1 = Text(abl1, width=50, font=tkFont.Font(size=12))

# laptördelés a'grid' metódus segítségével :
gomb1.grid(row =1, sticky =E, column =1)
gomb2.grid(row =1, sticky =E, column =2)
doboz1.grid(row =2, column =1)

# indítás :
abl1.mainloop()
```

***Feladat – Lekérdezés végrehajtása2***

Készíts olyan programot, amely lefuttat egy szövegbeviteli mezőbe beírt SELECT utasítást!


```
# -*- coding: ISO-8859-2 -*-

from Tkinter import *
import MySQLdb, sys

#A kapcsolat kiépítése
adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost',
port=3306)
cursor = adatBazis.cursor()

abl1 = Tk()

def lekerdez():
   #A parancs tárolása sztringben
   req = mezo1.get()
   #A parancs futtatása
   cursor.execute(req)
   records = cursor.fetchall()
   #A lekérdezés eredményének fájlba írása
   f = open('lekerdezes.txt','w')
   for rec in records:
      for item in rec:
         f.write(str(item)+'            ')
      f.write('\n')
   f.close()
   #Megjelenítés a szövegdobozban
   f = open('lekerdezes.txt','r')
   sz = f.read()
   doboz1.delete('1.0', END)
   doboz1.insert('1.0', sz)
   f.close()

# a widgetek létrehozása:
mezo1 = Entry(abl1, width=100)
gomb1 = Button(abl1, text='Lekérdezés', command=lekerdez)
gomb2 = Button(abl1, text='Kilépés', command=abl1.destroy)
import tkFont
doboz1 = Text(abl1, width=50, font=tkFont.Font(size=12))

# laptördelés a'grid' metódus segítségével :
mezo1.grid(row =1)
```

```
gomb1.grid(row =1, sticky =E, column =1)
gomb2.grid(row =1, sticky =E, column =2)
doboz1.grid(row =2)

# indítás :
abl1.mainloop()
```

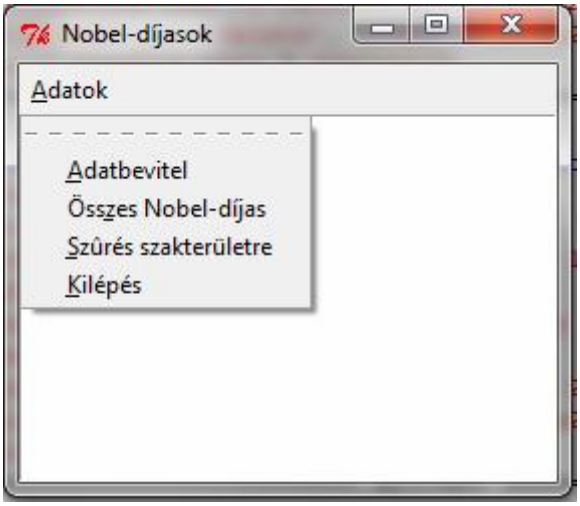
***Feladat – Nobel-díjasok***
A program tartsa nyilván a világ Nobel díjasait! Egy Nobel-díjhoz tartozó adatok: szakterület, évszám, név, ország.
A program sorban végezze el a következő feladatokat:

- Vegye nyilvántartásba a Nobel-díjasokat!
- Listázza ki az összes Nobel-díjast évszám szerinti rendezettségben!
- Készítse el egy adott szakterület listáját: kérje be a felhasználótól, hogy melyik szakterületre kíváncsi, majd listázza ki e szakterület Nobel-díjasait évszám szerint rendezetten!

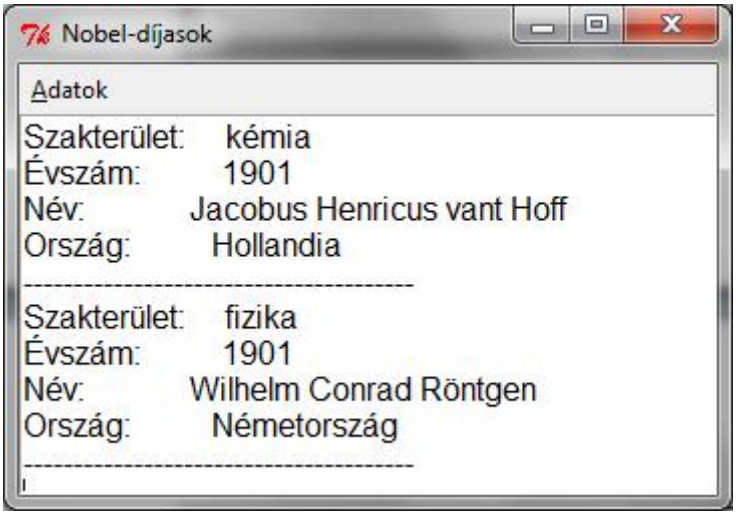
*Először hozzuk létre a táblát:*

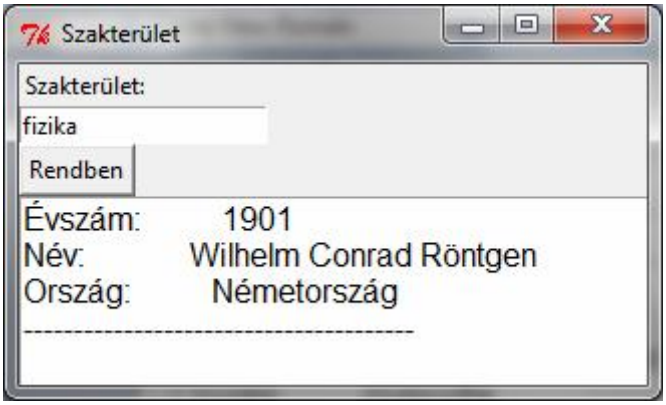
```
# -*- coding: ISO-8859-2 -*-

import MySQLdb, sys

#A kapcsolat kiépítése
adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost', port=3306)

#A parancs tárolása sztringben
req = "create table Nobel(szakterulet Text(20), evszam Integer, nev Text(30), orszag Text(20))"

#A parancs futtatása
adatBazis.cursor().execute(req)

print "A tábla létrejött."
```

*Készítsük el a menüvezérelt programot:*


Nobel-díjasok
Adatok
Adatbevitel
Összes Nobel-díjas
Szűrés szakterületre
Kilépés


Adatbevitel
Szakterület:
Évszám:
Név:
Ország:
Rögzít
Bezár


Nobel-díjasok
Adatok
Szakterület: kémia
Évszám: 1901
Név: Jacobus Henricus vant Hoff
Ország: Hollandia
---
Szakterület: fizika
Évszám: 1901
Név: Wilhelm Conrad Röntgen
Ország: Németország
---


Szakterület
Szakterület:
fizika
Rendben
Évszám: 1901
Név: Wilhelm Conrad Röntgen
Ország: Németország
---

```
# -*- coding: ISO-8859-2 -*-

from Tkinter import *
import MySQLdb, sys

adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost',
port=3306)

def bevitel():

    def rogzit():
        szak = m1.get()
        ev = m2.get()
        nev = m3.get()
        orsz = m4.get()
        #A parancs tárolása sztringben
        req = "insert into Nobel values('"+szak+"','"+ev+"','"+nev+"','"+orsz+"')"
        #A parancs futtatása
        adatBazis.cursor().execute(req)
        #Kiírás a lemezre
        adatBazis.commit()
        m1.delete(0,END)
        m2.delete(0,END)
        m3.delete(0,END)
        m4.delete(0,END)

    def bezar():
        abl2.destroy()

    abl2 = Toplevel(abl1)
    abl2.title('Adatbevitel')
    abl2.minsize(width =300, height=100)
    sz1 = Label(abl2, text ='Szakterület:')
    sz2 = Label(abl2, text ='Évszám:')
    sz3 = Label(abl2, text ='Név:')
    sz4 = Label(abl2, text ='Ország:')
    g1 = Button(abl2, text ='Rögzít', command=rogzit)
    g2 = Button(abl2, text ='Bezár', command=bezar)
    m1 = Entry(abl2)
    m2 = Entry(abl2)
    m3 = Entry(abl2)
    m4 = Entry(abl2)
    sz1.grid(row =1)
    sz2.grid(row =2)
    sz3.grid(row =3)
    sz4.grid(row =4)
    g1.grid(row =5, sticky =W)
    g2.grid(row =5, column =2, sticky =W)
    m1.grid(row =1, column =2, sticky =W)
    m2.grid(row =2, column =2, sticky =W)
```

```
    m3.grid(row =3, column =2, sticky =W)
    m4.grid(row =4, column =2, sticky =W)
    abl2.mainloop()

def osszegzes():
    cursor = adatBazis.cursor()
    #A parancs tárolása sztringben
    req = "select * from Nobel order by evszam"
    #A parancs futtatása
    cursor.execute(req)
    #Az eredmény tárolása
    records = cursor.fetchall()
    #A lekérdezés eredményének fájlba írása
    f = open('lekerdezes.txt','w')
    for rec in records:
        f.write('Szakterület:     '+str(rec[0])+'\n')
        f.write('Évszám:          '+str(rec[1])+'\n')
        f.write('Név:             '+str(rec[2])+'\n')
        f.write('Ország:          '+str(rec[3])+'\n')
        f.write('--------------------------------------'+'\n')
    f.close()
    #Megjelenítés a szövegdobozban
    f = open('lekerdezes.txt','r')
    sz = f.read()
    doboz1.delete('1.0', END)
    doboz1.insert('1.0', sz)
    f.close()

def szures():

    def megad():
        szt = m1.get()
        cursor = adatBazis.cursor()
        #A parancs tárolása sztringben
        req = "select * from Nobel where szakterulet='"+szt+"' order by evszam"
        #A parancs futtatása
        cursor.execute(req)
        #Az eredmény tárolása
        records = cursor.fetchall()
        #A lekérdezés eredményének fájlba írása
        f = open('lekerdezes.txt','w')
        for rec in records:
            f.write('Évszám:          '+str(rec[1])+'\n')
            f.write('Név:             '+str(rec[2])+'\n')
            f.write('Ország:          '+str(rec[3])+'\n')
            f.write('--------------------------------------'+'\n')
        f.close()
        #Megjelenítés a szövegdobozban
        f = open('lekerdezes.txt','r')
        sz = f.read()
```

```
        d1.delete('1.0', END)
        d1.insert('1.0', sz)
        f.close()

    #A szakterület bekérése
    abl3 = Toplevel(abl1)
    abl3.title('Szakterület')
    abl3.minsize(width =300, height=100)
    sz1 = Label(abl3, text ='Szakterület:')
    g1 = Button(abl3, text ='Rendben', command=megad)
    m1 = Entry(abl3)
    sz1.grid(row =1, sticky =W)
    m1.grid(row =2, sticky =W)
    g1.grid(row =3, sticky =W)
    import tkFont
    d1 = Text(abl3, width=100, font=tkFont.Font(size=12))
    d1.grid(row =4)
    abl3.mainloop()


abl1 = Tk()
abl1.title('Nobel-díjasok')

# a widgetek létrehozása:
menusor = Frame(abl1)
menusor.pack(side =TOP, fill =X)
menu1 = Menubutton(menusor, text ='Adatok', underline =0)
menu1.pack(side = LEFT )
fajl = Menu(menu1)
fajl.add_command(label ='Adatbevitel', command = bevitel, underline =0)
fajl.add_command(label ='Összes Nobel-díjas', command = osszegzes, underline =3)
fajl.add_command(label ='Szűrés szakterületre', command = szures, underline =0)
fajl.add_command(label ='Kilépés', command = abl1.destroy, underline =0)
menu1.config(menu = fajl)
import tkFont
doboz1 = Text(abl1, width=100, font=tkFont.Font(size=12))
doboz1.pack()

# indítás :
abl1.mainloop()
```

***Feladat – Lekérdezés végrehajtása + HTML***

Készíts olyan programot, amely lefuttat egy szövegbeviteli mezőbe beírt SELECT utasítást, az eredményt egy html állományba írja!


```
# -*- coding: ISO-8859-2 -*-

import MySQLdb, sys

#A kapcsolat kiépítése
adatBazis = MySQLdb.connect(db='proba',user='root', passwd='pingvin', host='localhost',
port=3306)
cursor = adatBazis.cursor()

#A parancs tárolása sztringben
req = "select * from Nobel"
#A parancs futtatása
cursor.execute(req)
#Az eredmény tárolása
records = cursor.fetchall()
#A lekérdezés eredményének fájlba írása
f = open('lekerdezes.html','w')
f.write('<HTML>')
f.write('<HEAD><TITLE>Nobel-díjasok</TITLE></HTML>')
f.write('<BODY>')
f.write('<H1>Nobel-díjasok</H1>')
f.write('<TABLE BORDER=1>')
f.write('<TR>')
f.write('<TD><B>SZAKTERÜLET</B></TD>')
f.write('<TD><B>ÉVSZÁM</B></TD>')
f.write('<TD><B>NÉV</B></TD>')
f.write('<TD><B>ORSZÁG</B></TD>')
f.write('</TR>')
```

```
for rec in records:
    f.write('<TR>')
    for item in rec:
        f.write('<TD>')
        f.write(str(item))
        f.write('</TD>')
    f.write('</TR>')
f.write('</TABLE>')
f.write('</BODY>')
f.write('</HTML>')
f.close()

print "Az exportálás sikerült."
```

FELADATOK

**1. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**KRONOLOGIA**
**Azonosito**
**Ev** **Az esemény évszáma**
**Helyseg** **Az esemény helyszíne**
**MiTortent** **Az esemény rövid leírása**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat évszám szerint rendezetten!**
**A program kérje be az évszázad sorszámát, és listázza az adott évszázad eseményeit!**

**2. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**EGESZSEG**
**Azonosito**
**Nev** **A sportoló neve**
**Testmagassag** **A sportoló testtömege**
**Testtomeg** **A sportoló testmagassága**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat azonosító szerint rendezetten!**
**A program számítsa ki a személyek testtömeg indexét**
**(Testtömeg(kg)/(testmagasság(m))$^2$), majd listázza a sportolók nevével együtt!**

**3. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**ENERGIA**
**Azonosito**
**Nev** **Az élelmiszer neve**
**Kaloria** **Az élelmiszer energiatartalma**
**Összetétel** **Rövid leírás az élelmiszer összetételéről**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat energiatartalom szerint rendezetten!**
**A program kérje be az adott élelmiszer nevét, majd írja ki az adatait!**

**4. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**KUTYA**
**Azonosito**

**Nev** **A talált kutya neve**
**Fajta** **A talált kutya fajtája**
**Mod** **Leírás arról, hogy hogyan találták**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat azonosító szerint rendezetten!**
**A program kérje be a megtalálás módját, és listázza az ahhoz tartozó kutyák adatait!**

**5. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**VIRAG**
**Azonosito**
**Nev** **A virág neve**
**Szin** **A virág jellemző színe**
**Fenyigeny** **Rövid leírás a növény fényigényéről**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat azonosító szerint rendezetten!**
**A program kérje be a jellemző színt, és listázza az ahhoz tartozó virágok adatait!**

**6. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**PARKETTA**
**Azonosito**
**Nev** **A parketta neve**
**Gyarto** **A parketta gyártója**
**Egysegar** **A parketta egységára**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat egységár szerint rendezetten!**
**A program kérjen be egy egységárat, és listázza az annál kisebb egységárú parketták adatait!**

**7. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**VAR**
**Azonosito**
**Helyseg** **A vár helye**
**Megye** **Melyik megyében van**
**Tortenet** **A vár történetének rövid leírása**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat azonosító szerint rendezetten!**
**A program kérje be a megye nevét, és listázza a megyében lévő várak adatait!**

**8. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**OPRENDSZER**
**Nev** **Az operációs rendszer neve**
**Gyarto** **Az operációs rendszer gyártója**
**Javido** **Átlagos javítási ideje**
**Jellemzok** **Rövid leírás a hardverkövetelményekről**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat név szerint rendezetten!**
**A program kérje be a gyártó nevét, és listázza a megfelelő operációs rendszerek adatait!**

**9. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**KEMIA**

**Azonosito**
**Nev** **Az elem neve**
**Vegyjel** **Az elem vegyjele**
**Rendszam** **Az elem rendszáma**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat rendszám szerint rendezetten!**
**A program kérje be a vegyjelet, és keresse ki az elem tulajdonságait!**

**10. Készíts menüvezérelt programot a következő adatok nyilvántartásához:**
**ALLOMANY**
**Azonosito**
**Kiterjesztes** **A fájl kiterjesztése**
**Program** **A fájl szerkesztésére használható program neve**
**Jellemzok** **Az adott fájltípus néhány jellemzője**
**A program biztosítson lehetőséget az adatbevitelre!**
**A program listázza az adatokat kiterjesztés szerint rendezetten!**
**A program kérje be a kiterjesztést, és keresse ki az állomány tulajdonságait!**

VÉGE A HARMADIK RÉSZNEK