BEVEZETÉS A PYTHON TKINTER PROGRAMOZÁSÁBA

(Informatikai szakközépiskola – 10. évfolyam)

Eddig a Pythont kizárólag „szövegmódban” használtuk. Azért jártunk így el, mert tisztázni kellett néhány elemi fogalmat és a nyelv alapszerkezetét, mielőtt komplexebb objektumokat (ablakok, képek, hangok, stb.) igényelő programokat készítünk. Most elkezdhetünk ismerkedni a grafikus interface-ek világával. Ez csak ízelítő lesz. Még sok alapvető dolog van, amit meg kell tanulni és közülük soknak a szöveges megközelítés a legmegfelelőbb.

1. Grafikus interface-ek (GUI)

A grafikus interface-ek (vagy *GUI : Graphical User Interface*) területe rendkívül komplex. Minden operációs rendszer több grafikus alap „függvénykönyvtárat” kínálhat, amihez gyakran társul még számos kiegészítő, többé-kevésbé speciális programozási nyelv. Általában ezek a komponensek objektum-osztályokként jelennek meg, amiknek az attribútumait és metódusait kell majd tanulmányozni.

A Pythonnal leginkább a Tkinter grafikus könyvtárat használják, ami a - kezdetben a Tcl nyelv számára fejlesztett - Tk könyvtár egy adaptációja. Sok más igen érdekes grafikus könyvtárat ajánlottak a Pythonhoz : wxPython, pyQT, pyGTK, stb. Lehetőség van a Java widget-ek és a Windows MFC -k használatára is.
Ebben a tananyagban a *Tkinter* -re fogunk szorítkozni, aminek szerencsére hasonló (és ingyenes) verziói vannak *Linux*, *Windows* és *Mac* platformokra.

2. Első lépések a Tkinter-rel

A használathoz a *Tkinter* modult telepíteni kell. Ahhoz, hogy a funkcióit használni tudjuk, a script valamelyik első sorának egy import utasítást kell tartalmaznia :

from Tkinter import *

Feladat – Mintaablak
Készítsük el az itt látható ablakot!


```
from Tkinter import *
ablak1 = Tk()
cimke1 = Label(ablak1, text='Döntsd el:', fg='red')
cimke1.pack()
gomb1 = Button(ablak1, text='OK')
gomb1.pack()
gomb2 = Button(ablak1, text='Nem OK')
gomb2.pack()
gomb3 = Button(ablak1, text='Kilépés', command = ablak1.destroy)
gomb3.pack()
ablak1.mainloop()
```

A program elemzése:

Könnyű olyan Python modulokat létrehozni, amik scripteket, függvénydefiníciókat, objektumosztályokat, stb. tartalmaznak. Ezeket a modulokat vagy teljes egészükben, vagy részben bármely programba importálhatjuk. Ezt tesszük példánk első sorában : a

from Tkinter import *

importálja a *Tkinter* modulban lévő valamennyi osztályt.
Egyre gyakrabban kell ***osztályokról*** (***class***-okról) beszélnünk. A programozásban így nevezzük az objektum generátorokat, amik újra felhasználható programrészletek.

Példánk második sorában :

ablak1 = Tk()

a Tkinter modul Tk() osztályát használjuk és annak egy példányát (egy objektumát) hozzuk létre az ablak1 -et.

Egy objektum létrehozása egy osztályból a mai programozási technikákban - melyek egyre gyakrabban folyamodnak az „objektum orientált programozás”-nak (vagy OOP : Object Oriented Programming) nevezett módszerhez - alapvető művelet.

Az osztály egy általános modell (vagy minta), amiből a gép kérésünkre egy speciális számítástechnikai objektumot hoz létre. Definíciókat és különböző opciókat tartalmaz, amiknek csak egy részét használjuk a belőle létrehozott objektumban. Így a Tkinter könyvtár egyik legalapvetőbb osztálya, a Tk() osztály mindent tartalmaz, ami különböző típusú, méretű és színű, menüsoros vagy anélküli, stb. alkalmazásablakok létrehozásához szükséges.

Ezt használjuk fel a grafikus alapobjektumunk, egy ablak létrehozására, ami a többi objektumot fogja tartalmazni. A Tk() zárójelében különböző opciókat adhatnánk meg, de ezekkel egy kicsit később foglalkozunk.

Az objektumot létrehozó utasítás egy egyszerű értékadáshoz hasonlít. Azonban itt két dolog történik egyszerre :

- egy új objektum jön létre, (ami összetett lehet és ezért tekintélyes mennyiségű memóriát foglalhat le)
- értéket adunk egy változónak, ami ettől fogva az objektum hivatkozásaként fog szolgálni.

A harmadik sorban :

cimke1 = Label(ablak1, text='Jó napot !', fg='red')

egy másik objektumot (egy *widget*-et) hozunk létre, ez alkalommal a **Label()** osztályból.

Amint a neve is jelzi, ez az osztály mindenféle *címkét* definiál. Ezek szövegtöredékek, amik arra használhatók, hogy különböző információkat írjunk az ablak belsejébe.

A korrekt kifejezésre törekedve azt fogjuk mondani, hogy a **cimke1** objektumot a **Label()** osztályból hoztuk létre (***instanciáltuk vagy példányosítottuk***).

Ugyanúgy hívunk egy osztályt, mint egy függvényt : vagyis zárójelben argumentumokat adunk meg. Majd meglátjuk, hogy egy osztály egyfajta konténer, amiben függvények és adatok vannak összegyűjtve.

Milyen argumentumokat adtunk meg ehhez a példányosításhoz ?

Az első argumentum (**ablak1**) azt adja meg, hogy a most létrehozott új *widget*-et ***egy már létező másik widget fogja tartalmazni***. Az utóbbit az új *widget* „master” *widgetjének* nevezzük, az **ablak1** *widget* a **cimke1** objektum tulajdonosa. (Azt is mondhatjuk, hogy a **cimke1** objektum az **ablak1** objektum ***slave*** *widget* -je).

A következő két argumentum a *widget* pontos alakjának meghatározására való. Ez két ***kezdőérték,*** mindegyikük string formában van megadva : első a címke szövege, második az előtér (f*oreground* rövidítve **fg**) színe. Esetünkben a kiíratni kívánt szöveg színét pirosnak definiáltuk.

Egyéb jellemzőket is meg tudnánk még határozni, például : a betűtípust, vagy a háttér színét. Ezeknek a jellemzőknek azonban van egy alapértelmezett értékük a **Label()** osztály belső definícióiban. Csak azoknak a jellemzőknek kell értéket adnunk, melyeket a standard modeltől eltérőnek akarunk.

A negyedik sorban :

cimke1.pack()

a cimke1 objektumhoz tartozó pack() metódust aktiváljuk. A metódus egy objektumba beépített (azt is fogjuk mondani, hogy egy objektumba bezárt) függvény. Egy objektum valójában egy programkód részlete, ami mindig tartalmaz :

- különböző típusú változókban tárolt adatokat (numerikusokat vagy másféléket) : ezeket az objektum attribútumainak (vagy tulajdonságainak nevezzük).
- eljárásokat vagy függvényeket (ezek tehát algoritmusok) : ezeket az objektum metódusainak nevezzük.

A pack() metódus része egy metóduscsoportnak, aminek a tagjai nemcsak a Label() osztály widget -jeire alkalmazhatók, hanem más Tkinter widget-ekre is. A widgetek ablakbeli geometriai elrendezésére hatnak. Ha egyesével írjuk be a példa parancsait, meggyőződhetünk róla, hogy a pack() metódus automatikusan akkorára csökkenti a „master” ablak méretét, hogy az elég nagy legyen az előre definiált „slave” widget -ek tartalmazásához.

Az kilencedik sorban :

gomb3 = Button(ablak1, text='Kilép', command = ablak1.destroy)

létrehozzuk negyedik „slave” widget -ünket : egy újabb nyomógombot.

Ahogyan az előző widget esetén tettünk, a Button() class-t hívjuk zárójelek között a paramétereket megadva. Mivel ez alkalommal egy interaktív objektumról van szó, ezért a command opcióval meg lehet adnunk, hogy mi történjen, amikor a felhasználó a nyomógombra kattint. Ebben az esetben az ablak1 objektumhoz tartozó destroy metódust hozzuk működésbe, ami törli az ablakot.

A tizedik sorban a pack() metódus az ablak geometriáját az imént beépített új objektumhoz igazítja.

Az utolsó sor :

abl1.mainloop()

nagyon fontos, mert ez indítja el az ablakhoz kapcsolt eseményfogadót, ami arra kell, hogy az alkalmazásunk figyelje az egérkattintásokat, billentyűleütéseket, stb. Ez az az utasítás, amelyik valamilyen módon elindítja az alkalmazást.

Mint a neve (mainloop = főhurok) mutatja, az ablak1 objektum egyik metódusáról van szó, ami egy programhurkot aktivál. Ez a programhurok háttérprogramként folyamatosan működik és várja az operációs rendszer által küldött üzeneteket. Folyamatosan lekérdezi a környezetét :a bemeneti perifériákat (egér, billentyűzet, stb.). Valamilyen esemény detektálásakor az illető eseményt leíró üzeneteket küld azoknak a programoknak, amiknek szükségük van arra, hogy tudjanak az illető esemény bekövetkezéséről. Nézzük meg ezt egy kicsit részletesebben.

3. Eseményvezérelt programok

Az első grafikus interface-ű programunkkal kísérleteztünk az imént. Az ilyen típusú program máshogyan van strukturálva, mint a korábban tanulmányozott „szöveges” scriptek.

Minden számítógépprogram működésének nagyjából három fő szakasza van :

I. inicializálási szakasz :

ez az elvégzendő munkára felkészítő utasításokat tartalmazza (a szükséges külső modulok hívása, fájlok megnyitása, adatbázis-szerverhez vagy az internethez kapcsolódás, stb.);

II. lényegi működési szakasz :

itt összpontosul a lényegi működés (vagyis mindaz, amit a program tesz : adatokat ír a képernyőre, számításokat végez, módosítja egy file tartalmát, nyomtat, stb.);

III. befejező szakasz :

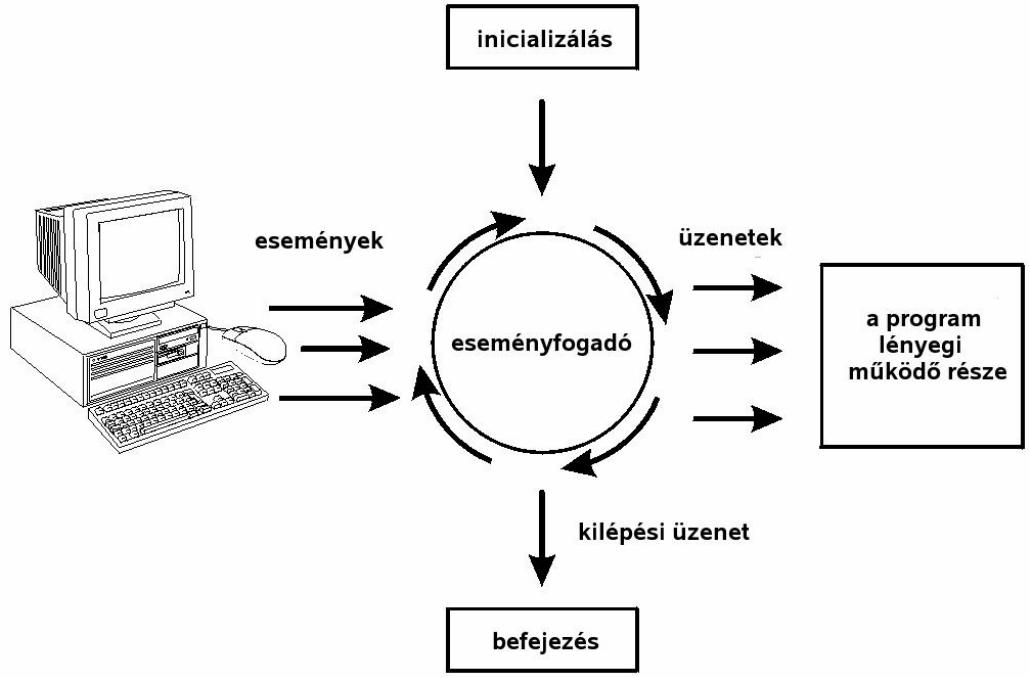
ez a műveletek megfelelő lezárására szolgál (vagyis a nyitva maradt fájlok lezárása, külső kapcsolatok bontása, stb.)

„Szövegmódú” programban ez a három szakasz egymást követi. Ennek az a következménye, hogy ezeket a programokat nagyon korlátozott interaktivitás jellemzi. A felhasználónak gyakorlatilag semmilyen szabadsága sincs : a program időről-időre megkéri, hogy írjon be adatokat, de mindig a programutasítások sorrendjének megfelelő, előre meghatározott sorrendben.

Egy grafikus interface-ű programnak ettől eltérő a belső szervezése. Azt mondjuk, hogy az ilyen program eseményvezérelt. Az inicializálási szakasz után az ilyen típusú program „várakozásba megy át” és rábízza magát egy másik, többé-kevésbé mélyen a számítógép operációs rendszerébe integrált és állandóan működő programra.

Ez az eseményfogadó állandón figyeli a perifériákat (billentyűzetet, egeret, órát, modemet, stb.) és azonnal reagál rá, ha egy eseményt érzékel.

Egy ilyen esemény lehet bármilyen felhasználói akció: az egér elmozdítása, billentyű lenyomása, stb., de egy belső esemény, vagy egy automatizmus is lehet (például órajel).



Amikor az eseményfogadó érzékel egy eseményt, akkor egy specifikus jelet küld a programnak, aminek azt fel kell ismerni, hogy reagáljon rá.

Egy grafikus user interface-ű program inicializálási szakasza olyan utasításokból áll, amik az interface különböző interaktív komponenseit helyezik el (ablakokat, gombokat, check

boxokat, stb.). Más utasítások a kezelendő üzeneteket definiálják : eldönthetjük, hogy a program csak bizonyos eseményekre reagáljon és a többit hagyja figyelmen kívül.

Míg egy „szöveges” programban a működési szakasz olyan utasítás sorozatból áll, ami előre leírja azt a sorrendet, ahogyan a gépnek a különböző feladatait végre kell hajtania (még ha különböző végrehajtási utakról gondoskodunk is válaszul a végrehajtás során adódó különböző feltételekre), addig egy grafikus interface-ű program működési fázisában csak független függvényeket találunk. Ezeket a program specifikusan akkor hívja, amikor egy meghatározott eseményt érzékel az operációs rendszer : vagyis azt a feladatot végzik el, amit erre az eseményre válaszul várunk a programtól és semmi mást.

Fontos, hogy megértsük : ez alatt az eseményfogadó folyamatosan működik és esetleges más események bekövetkezésére vár.

Ha más események következnek be, akkor megtörténhet, hogy egy második (vagy egy 3., egy 4.) függvény aktiválódik és az elsővel, ami még nem fejezte be működését, párhuzamosan kezd el működni. A modern operációs rendszerek és nyelvek lehetővé teszik ezt a párhuzamosságot, amit multitaskingnak is nevezünk.

A programszöveg struktúrája nem adja meg közvetlenül az utasítások végrehajtási sorrendjét. Ez méginkább igaz a grafikus interface-ű programokra, mivel a függvények hívásának sorrendje már sehol sincs leírva a programban. Az események vezérelnek !

Feladat – Számológép

Készítsük el az itt látható ablakot! Az ablak négy alapműveletes számológépként működjön!


```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *
def osszead():
    x = eval(mezo1.get())
    y = eval(mezo2.get())
    z = x + y
    mezo3.delete(0,END)
    mezo3.insert(0,'Eredmény: '+str(z))

def kivon():
    x = eval(mezo1.get())
    y = eval(mezo2.get())
    z = x - y
    mezo3.delete(0,END)
    mezo3.insert(0,'Eredmény: '+str(z))
```

```
def szoroz():
    x = eval(mezo1.get())
    y = eval(mezo2.get())
    z = x * y
    mezo3.delete(0,END)
    mezo3.insert(0,'Eredmény: '+str(z))

def oszt():
    x = eval(mezo1.get())
    y = eval(mezo2.get())
    z = x / y
    mezo3.delete(0,END)
    mezo3.insert(0,'Eredmény: '+str(z))

ablak1 = Tk()
mezo1 = Entry(ablak1)
mezo1.pack()
mezo2 = Entry(ablak1)
mezo2.pack()
gomb1 = Button(ablak1, text='Összeadás', command=osszead)
gomb1.pack()
gomb2 = Button(ablak1, text='Kivonás', command=kivon)
gomb2.pack()
gomb3 = Button(ablak1, text='Szorzás', command=szoroz)
gomb3.pack()
gomb4 = Button(ablak1, text='Osztás', command=oszt)
gomb4.pack()
mezo3 = Entry(ablak1)
mezo3.pack()
gomb5 = Button(ablak1, text='Kilépés', command = ablak1.destroy)
gomb5.pack()
ablak1.mainloop()
```

4. A Tkinter widget-osztályai

Apránként meg fogunk ismerkedni néhány widget használatával. Azokra a widget-ekre fogunk szorítkozni, amelyek módszertani szempontból a legérdekesebbnek tűnnek

A Tkinter widget-eknek 15 alaposztálya létezik :

Button	Klasszikus nyomógomb, valamilyen utasítás végrehajtásának az előidézésére használják.
Canvas	Különböző grafikus elemek elhelyezésére szolgáló felület. Rajzolásra, grafikus szerkesztők létrehozására és testre szabott widget-ek implementálására is használhatjuk.
Checkbutton	Egy jelölőnégyzet, aminek két különböző állapota lehet (a négyzet ki van jelölve vagy nincs kijelölve). Egy klikkelés a widgeten állapotváltozást idéz elő.
Entry	Adatbeviteli mező, amibe a felhasználó bármilyen szöveget beírhat.

Frame	Egy téglalap alakú felület az ablakban, ahova más widget-eket tehetünk. Ez a felület színes lehet. Szegélye is lehet.
Label	Valamilyen szöveg (vagy címke) (esetleg egy kép).
Listbox	A felhasználónak - általában valamilyen doboz formájában - felajánlott választéklista. A Listbox-ot úgy is konfigurálhatjuk, hogy vagy egy rádiógomb vagy egy jelölőnégyzet sorozatot tartalmazzon.
Menu	Menü. Lehet címsorhoz kapcsolódó legördülő menü, vagy egy kattintás után akárhol feljövő úszó popup menü.
Menubutton	Menügomb, amit legördülő menük implementálására használnak.
Message	Szöveg kiírását teszi lehetővé. A Label widget egy változata, ami lehetővé teszi, hogy a kiírt szöveg automatikusan egy bizonyos mérethez, vagy szélesség/magasság arányhoz igazodjon.
Radiobutton	(Egy fekete pont egy kis körben.) Egy változó lehetséges értékeit reprezentálja. Az egyik rádiógombra való kattintás az annak megfelelő értéket adja a változónak.
Scale	Egy kurzornak egy skála mentén való mozgatásával teszi láthatóvá egy változó értékének a változtatását.
Scrollbar	A görgető sort más widget-ekhez (Canvas, Entry, Listbox, Text) kapcsolva használhatjuk.
Text	Formázott szöveg kiírása. A felhasználónak is lehetővé teszi a kiírt szöveg formázását. Képeket is be lehet szúrni.
Toplevel	Egy külön, felülre kiírt ablak.

Mindegyik widget-osztály-nak számos beépített metódusa van. Kapcsolhatunk hozzájuk eseményeket is, amint azt az előző oldalakon láttuk. Többek között meg fogjuk tanulni, hogy ezeket a widget-eket a grid(), a pack() és a place() metódusok segítségével pozícionálhatjuk az ablakokban.

Ezeknek a módszereknek a haszna akkor válik nyilvánvalóvá, amikor megpróbálunk portábilis programokat írni (vagyis olyanokat, amik képesek különböző operációs rendszereken – mint a Unix, MacOS vagy Windows - futni), amiknek az ablakai átméretezhetőek.

5. A grid() metódus alkalmazása widget-ek pozícionálására

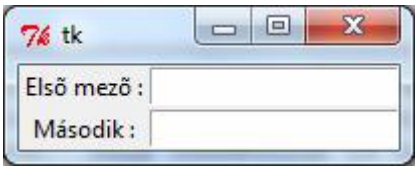
Eddig mindig a pack() metódus segítségével helyeztük el a widget-eket az ablakukban. Ennek a metódusnak a rendkívüli egyszerűség az előnye, azonban nem ad túl sok szabadságot a widget-ek kedvünk szerinti elrendezéséhez.

Feladat – Mintaablak2
Készítsük el az itt látható ablakot!

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

abl1 = Tk()
txt1 = Label(abl1, text = 'Első mező :')
txt2 = Label(abl1, text = 'Második :')
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
txt1.pack(side =LEFT)
txt2.pack(side =LEFT)
mezo1.pack(side =LEFT)
mezo2.pack(side =LEFT)

abl1.mainloop()
```

Tegyünk néhány kísérletet, hogy a pack() metódus nak különböző side = … argumentumokat adunk. Hogy jobban megértsük a pack() metódus működését, megpróbálhatjuk még a side =TOP, side=BOTTOM, side=RIGHT opciók különböző kombinációit a négy widget mindegyikére.

Feladat – Mintaablak3
Készítsük el az itt látható ablakot!



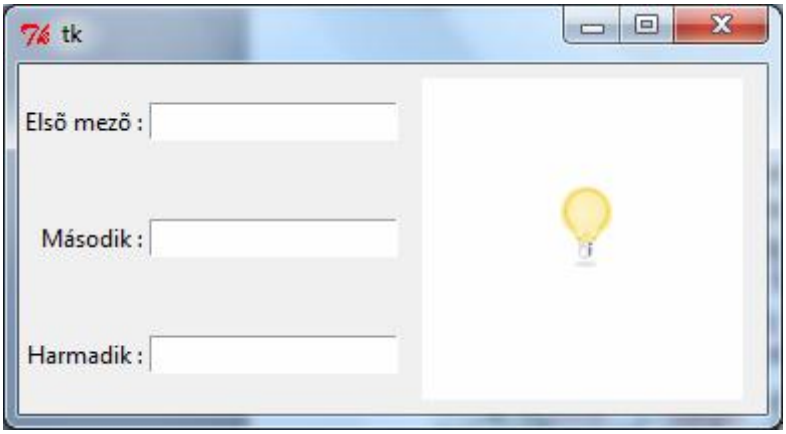

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

abl1 = Tk()
txt1 = Label(abl1, text = 'Első mező :')
txt2 = Label(abl1, text = 'Második :')
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
txt1.grid(row =0)
txt2.grid(row =1)
mezo1.grid(row =0, column =1)
mezo2.grid(row =1, column =1)
abl1.mainloop()
```

Ebben a scriptben a ***pack***() metódust a ***grid***()-del helyettesítettük. Látható, hogy az utóbbi használata igen egyszerű. Ez az ablakot egy táblázatnak (rácsnak) tekinti. Elég, ha megadjuk neki, hogy a táblázat melyik sorába (**row**) és melyik oszlopába (**column**) akarjuk elhelyezni a widget-eket. A sorokat és oszlopokat úgy számozhatjuk, ahogy akarjuk : kezdhetjük nullától, vagy egytől, vagy akármilyen számtól. A Tkinter figyelmen kívül fogja hagyni az üres sorokat és oszlopokat. *Jegyezzük meg, ha nem adunk meg semmilyen sor vagy oszlopszámot, akkor az alapértelmezett érték nulla lesz.*

A Tkinter automatikusan meghatározza a szükséges sorok és oszlopok számát. Láthatjuk, hogy az ablak baloldalán megjelenő karakterláncok középre igazítottak, ha mi jobbra igazítva szeretnénk őket, akkor elég a ***grid***() metódusnak egy argumentumot megadni. A **sticky** opció négy értéket vehet fel: N, S, W, E (a négy égtáj angol rövidítését). Ettől az értéktől függően lesznek a widget-ek föntre, lentre, balra, vagy jobbra igazítva. Helyettesítsük a scriptben a két első ***grid***() utasítást a következőkkel :

txt1.***grid***(**row** =0, **sticky** =E)
txt2.***grid***(**row** =1, **sticky** =E)

Feladat – Mintaablak4
Készítsük el az itt látható ablakot!



Az ablak 3 oszlopot tartalmaz: az elsőben 3 string, a másodikban 3 adatbeviteli mező, a harmadikban egy kép van. Az első két oszlopoban 3-3 sor van, viszont a harmadik oszlopban lévő kép valamilyen módon lefedi a három sort.

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

abl1 = Tk()

# a 'Label' és 'Entry' widgetek létrehozása:
txt1 = Label(abl1, text ='Első mező :')
txt2 = Label(abl1, text ='Második :')
txt3 = Label(abl1, text ='Harmadik :')
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)

# egy bitmap képet tartalmazó 'Canvas' widget létrehozása :
can1 = Canvas(abl1, width =160, height =160, bg ='white')
photo = PhotoImage(file ='light.gif')
item = can1.create_image(80, 80, image =photo)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
```

```
txt3.grid(row =3, sticky =E)
mezo1.grid(row =1, column =2)
mezo2.grid(row =2, column =2)
mezo3.grid(row =3, column =2)
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)

# indítás :
abl1.mainloop()
```

Ahhoz, hogy az olvasó futtatni tudja a scriptet, valószínűleg a képfile nevét (Martin_p.gif) a választása szerinti kép nevével kell helyettesítenie. Figyelem : a standard Tkinter könyvtár csak kevés fileformátumot fogad el. Válassza lehetőség szerint a GIF formátumot.

Néhány dolgot megjegyezhetünk a scriptből :

1) *A kép beszúrására alkalmazott technikát* :
A Tkinter nem teszi lehetővé, hogy közvetlenül szúrjunk be képet az ablakba. Először egy vásznat (canvas) kell az ablakba elhelyezni és a képet erre kell tenni. A vászonnak fehér színt választottunk azért, hogy meg lehessen különböztetni az ablaktól. Ha azt akarjuk, hogy a vászon láthatatlanná váljon, akkor a **bg** ='white' paramétert **bg** ='gray'-vel helyettesítsük. Mivel többféle képtípus létezik, ezért a ***PhotoImage***() class-ban a képobjektumot GIF típusú bitmap képnek kell deklarálnunk.

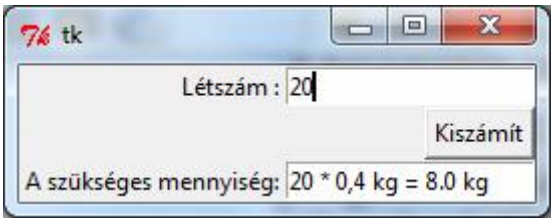
2) *A kép vászonra helyezésének módját* :
item = can1.***create_image***(80, 80, **image** =*photo*)
Pontos szóhasználattal azt mondjuk, hogy a can1 objektumhoz (amely objektum a Canvas osztály egy példánya) kötött ***create_image***() metódust használjuk. A két első argumentum (80, 80) adja meg a vászonnak azt az x és y koordinátáját, ahová a kép középpontját kell tenni. (Mivel a vászon 160x160-as méretű, ezért a képet a vászon közepére igazítja a koordináta választásunk).

3) *A sorok és oszlopok számozását a grid() metódusban* :
Megállapíthatjuk, hogy ez alkalommal a ***grid***() metódusban a sorok és oszlopok számozása 1-gyel kezdődik (nem pedig nullával, mint az előző scriptben). Amint már feljebb említettem, a számozás választása teljesen szabad. Számozhatnánk így is : 5, 10, 15, 20... mert a Tkinter figyelmen kívül hagyja az üres sorokat és oszlopokat. Az 1-től kezdődő számozás valószínűleg a kódunk olvashatóságát javítja.

4) *A grid()-del a vászon elhelyezésére használt argumentumokat* :
can1.***grid***(**row** =1, **column** =3, **rowspan** =3, **padx** =10, **pady** =5)
Az első kettő azt jelenti, hogy a vászon a harmadik oszlop első sorában lesz elhelyezve. A harmadik (**rowspan** =3) azt jelenti, hogy három sorra fog kiterjedni.
A két utolsó (**padx** =10, **pady** =5) annak a felületnek a méreteit jelenti, amit a widget körül kell lefoglalnunk (magasságban és szélességben).

Feladat – Krumpli
Egy cserkésztáborban készül a vacsora, de a konyhafőnök nem tudja, mennyi krumplit vegyen. A fejadagot ismeri az 0,4 kg, de a szükséges mennyiséget számítógép segítségével szeretné kiszámolni az aktuális létszámtól függően. Írjuk meg a programot a

konyhafőnöknek! A program kérje be a létszámot, majd írja ki a szükséges mennyiséget a következőképpen!




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

def kiszamit():
    letszam = eval(mezo1.get())
    mennyiseg = letszam * 0.4
    mezo2.delete(0,END)
    mezo2.insert(0,str(letszam)+' * 0,4 kg = '+str(mennyiseg)+' kg')

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Létszám :')
txt2 = Label(abl1, text ='A szükséges mennyiség:')
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

Feladat – Henger

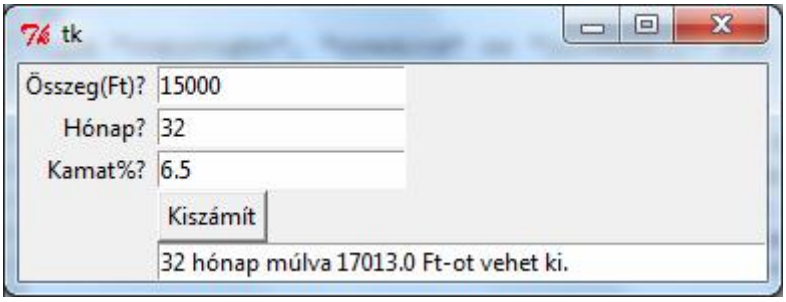
Kérjük be egy henger sugarát és magasságát cm-ben, majd
- írjuk ki a henger térfogatát!
- Írjuk ki a henger súlyát, ha ez tömör vashenger, és ha fahenger!

A kiírásokban a számokat kerekítsük 2 tizedesre!


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

def kiszamit():
    r = eval(mezo1.get())
    m = eval(mezo2.get())
    terf = r * r * pi * m
    vas = terf * 7.8
    fa = terf * 0.7
    mezo3.delete(0,END)
    mezo3.insert(0,str(round(terf,2))+' cm3')
    mezo4.delete(0,END)
    mezo4.insert(0,str(round(vas,2))+' g')
    mezo5.delete(0,END)
    mezo5.insert(0,str(round(fa,2))+' g')

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Sugár (cm):')
txt2 = Label(abl1, text ='Magasság (cm):')
txt3 = Label(abl1, text ='Térfogat :')
txt4 = Label(abl1, text ='Vashenger :')
txt5 = Label(abl1, text ='Fahenger :')
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1)
mezo5 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =4, sticky =E)
txt4.grid(row =5, sticky =E)
txt5.grid(row =6, sticky =E)
gomb1.grid(row =3, sticky =E, column =2)
mezo1.grid(row =1, column =2)
```

```
mezo2.grid(row =2, column =2)
mezo3.grid(row =4, column =2)
mezo4.grid(row =5, column =2)
mezo5.grid(row =6, column =2)

# indítás :
abl1.mainloop()
```

Feladat – Bank
Ha beteszünk a bankba egy adott összeget, adott éves kamatszázalékra, adott hónapra, mennyi pénzt vehetünk majd fel az idő lejártakor?




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

def kiszamit():
    osszeg = eval(mezo1.get())
    honap = eval(mezo2.get())
    kamat = eval(mezo3.get())
    ujosszeg = osszeg * pow(1+kamat/100,honap/12)
    mezo4.delete(0,END)
    mezo4.insert(0,str(honap)+' hónap múlva '+str(round(ujosszeg,0))+' Ft-ot vehet ki.')

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Összeg(Ft)? ')
txt2 = Label(abl1, text ='Hónap? ')
txt3 = Label(abl1, text ='Kamat%? ')
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1, width =50)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
gomb1.grid(row =4, sticky =W, column =2)
```

```
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =3, sticky =W, column =2)
mezo4.grid(row =5, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

FELADATOK:

1. Kérd be a felhasználó nevét, majd írd ki a következő jókívánságot:
Kedves <<X>>! Sikeres Tkinter programozást!

2. Kérd be egy téglatest három élének hosszúságát, majd írd ki a hasáb felszínét és térfogatát!

3. Tárold konstansokban a krumpli, a hagyma és a padlizsán egységárát! Írj olyan programot, amely bekéri, hogy miből mennyit óhajt a vásárló, majd készítsen egy számlát a következő formában:
Krumpli : 2.5 kg * 70 Ft/kg = 175 Ft
Hagyma : 3.0 kg * 98 Ft/kg = 294 Ft
Padlizsán : 10.0 kg * 200 Ft/kg = 2000 Ft

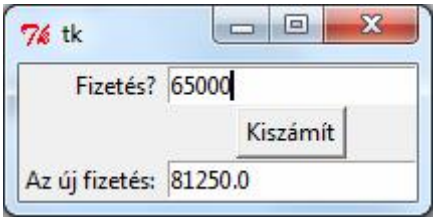
Összesen 2469 Ft

4. Kérd be a gömb sugarát, majd írd ki a gömb felszínét és térfogatát!

5. Ha a számla ÁFA összege a számla nettó értékének egy adott százaléka, akkor hány százalék ÁFÁ-t tartalmaz a számla bruttó összege? Készíts a problémára egy kisegítő programot! Például 25%-os ÁFA esetén a számla 20% ÁFÁ-t tartalmaz, 12%-os ÁFA esetén a számla ÁFA tartalma 10,71%.

6. Feri pénzt kap. Hogy mennyit, azt kérje be a program. A kifizetéshez 5000, 1000, 500 és 100 Ft-os címletek állnak rendelkezésre – a maradékot Feri nem kapja meg. Feltételezzük, hogy minden címletből van elég, és a lehető legkevesebb számú pénz kerül kiosztásra. Milyen címletből hányat kapott Feri, és mennyit hagyott ott ajándékba.

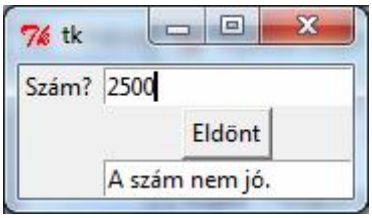
Feladat – Fizetés
Kérjük be egy alkalmazott fizetését! Ha ez a fizetés 100 000 forintnál nem nagyobb, akkor emeljük meg 25%-kal! Végül írjuk ki az alkalmazott fizetését!




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

```
def kiszamit():
    fiz = eval(mezo1.get())
    if (fiz < 100000) :
      fiz = fiz * 1.25
    mezo2.delete(0,END)
    mezo2.insert(0,str(fiz))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Fizetés? ')
txt2 = Label(abl1, text ='Az új fizetés: ')
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

Feladat – Jó szám
Kérjünk be egy valós számot! A szám akkor jó, ha 1000 és 2000 közötti páros egész (a határokat is beleértve). Írjuk ki, hogy a szám jó, vagy nem jó!



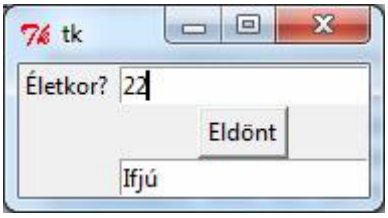

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def eldont():
    szam = eval(mezo1.get())
    if (szam >= 1000) and (szam <= 2000) and (szam % 2 == 0):
        mezo2.delete(0,END)
        mezo2.insert(0,'A szám jó.')
    else:
        mezo2.delete(0,END)
        mezo2.insert(0,'A szám nem jó.')

abl1 = Tk()
```

```
# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szám? ')
gomb1 = Button(abl1, text='Eldönt', command=eldont)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

Feladat – Kor

Olvassunk be egy nem negatív egész számot, valakinek az életkorát! Kortól függően írjuk ki a megfelelő szöveget:

0 – 13 évig: Gyermek
14 – 17 évig: Fiatalkorú
18 – 23 évig: Ifjú
24 – 59 évig: Felnőtt
60 évtől Idős!




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

def eldont():
    kor = eval(mezo1.get())
    if (kor <= 13):
        mezo2.delete(0,END)
        mezo2.insert(0,'Gyermek')
    elif (kor <= 17):
        mezo2.delete(0,END)
        mezo2.insert(0,'Fiatalkorú')
    elif (kor <= 23):
        mezo2.delete(0,END)
        mezo2.insert(0,'Ifjú')
    elif (kor <= 59):
        mezo2.delete(0,END)
        mezo2.insert(0,'Felnőtt')
    else:
        mezo2.delete(0,END)
```

```
        mezo2.insert(0,'Idős.')

    abl1 = Tk()

    # a widgetek létrehozása:
    txt1 = Label(abl1, text ='Életkor? ')
    gomb1 = Button(abl1, text='Eldönt', command=eldont)
    mezo1 = Entry(abl1)
    mezo2 = Entry(abl1)

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

    # indítás :
    abl1.mainloop()
```

FELADATOK:

1. Kérd be egy telek oldalait méterben! Írd ki a telek területét négyszögölben! (1négyszögöl = 3,6 m2). Ha a telek 100 négyszögölnél kisebb, akkor írja ki, hogy túl kicsi!

2. Van egy henger alakú hordónk, melybe nem tudjuk, hogy belefér-e a rendelkezésre álló bor. Kérd be a bor mennyiségét literben, majd a hordó összes szükséges adatát cm-ben. Adj tájékoztatást, hogy hány literes a hordó, és hogy belefér-e a hordóba a bor! Ha belefér, akkor add meg, hogy mennyi férne még bele! Írd ki százalékosan is a telítettséget! Az adatokat egészre kerekítve írd ki!

3. Kérj be egy évszámot! Ha a beütött szám negatív, akkor adj hibajelzést, ha nem, akkor állapítsd meg, hogy az évszám osztható-e 17-tel, vagy nem!

4. Kérd be Zsófi, Kati és Juli születési évét. Írd ki a neveket udvariassági sorrendben (előre az idősebbeket…)!

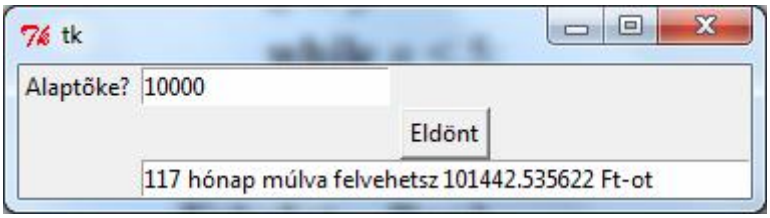
5. Kérj be egy egyjegyű, nem negatív számot! Írd ki a szám szöveges formáját (1=egy, 2=kettő stb.)

6. Kérj be egy egész óra értéket. Ha a szám nem 0 és 24 óra között van, akkor adjon hibaüzenetet, egyébként köszönjön el a program a napszaknak megfelelően! 4-9: Jó reggelt!, 10-17: Jó napot!, 18-21: Jó estét!, 22-3: Jó éjszakát!

7. Egy dolgozatra annak pontszámától függően a következő osztályzatot adják:
elégtelen (1): 0 – 29
elégséges (2): 30 – 37
közepes (3): 38 – 43
jó (4): 44 – 49
jeles (5): 50 – 55

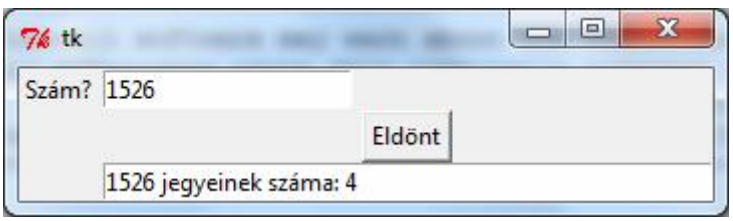
Kérd be a dolgozat pontszámát, majd írja ki az osztályzatot számmal és betűvel!

Feladat – Bank
Van egy kis megtakarított pénzem. Arra vagyok kíváncsi, hogy hány hónap múlva éri el ez az összeg a bankban a 100 000 Ft-ot, ha havi 2%-os kamattal számolhatok?




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

def kiszamit():
    penz = eval(mezo1.get())
    honap = 0
    while (penz < 100000):
        penz = penz * 1.02
        honap = honap + 1
    mezo2.delete(0,END)
    mezo2.insert(0,str(honap)+' hónap múlva felvehetsz '+str(penz)+' Ft-ot')

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Alaptőke? ')
gomb1 = Button(abl1, text='Eldönt', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1, width =50)

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

# indítás :
abl1.mainloop()
```

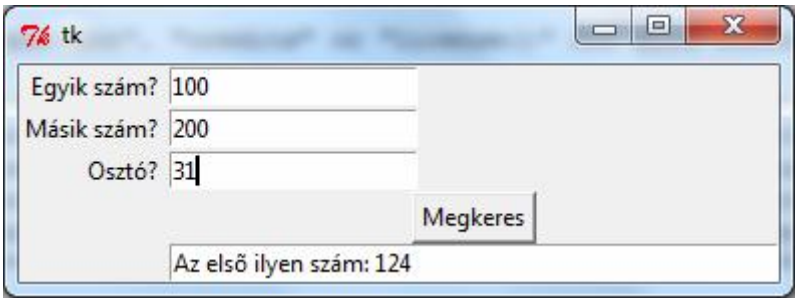
Feladat – Jegyek száma
Kérjünk be egy számot! Írjuk ki a jegyeinek a számát!



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

def kiszamit():
    szam = eval(mezo1.get())
    seged = szam
    jegySzam = 0
    while (seged != 0):
        seged = seged / 10
        jegySzam = jegySzam + 1
    mezo2.delete(0,END)
    mezo2.insert(0,str(szam)+' jegyeinek száma: '+str(jegySzam))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szám? ')
gomb1 = Button(abl1, text='Eldönt', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1, width =50)

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

# indítás :
abl1.mainloop()
```

Feladat – Oszthatóság

Két szám között határozzuk meg az első olyan számot, amelyik osztható egy megadott számmal!

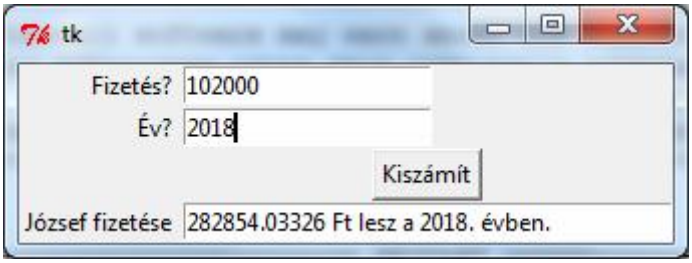



```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def megkeres():
    kezd = eval(mezo1.get())
    veg = eval(mezo2.get())
    oszto = eval(mezo3.get())
```

```
    if (kezd > veg):
        seged = kezd
        kezd =veg
        veg = seged
    while (kezd % oszto != 0) and (kezd <= veg):
        kezd = kezd + 1
    if (kezd > veg):
        mezo4.delete(0,END)
        mezo4.insert(0,'Nem található ilyen szám')
    else:
        mezo4.delete(0,END)
        mezo4.insert(0,'Az első ilyen szám: '+str(kezd))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Egyik szám? ')
txt2 = Label(abl1, text ='Másik szám? ')
txt3 = Label(abl1, text ='Osztó? ')
gomb1 = Button(abl1, text='Megkeres', command=megkeres)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1, width =50)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
gomb1.grid(row =4, column =2)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =3, sticky =W, column =2)
mezo4.grid(row =5, column =2)

# indítás :
abl1.mainloop()
```

Feladat – Fizetés

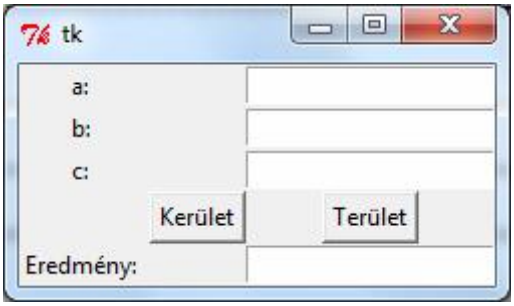
Most 2009-et írunk. Írjuk ki, hogy egy adott évben mennyi lesz József fizetése, ha évenként 12%-kal növekszik! József jelenlegi fizetését, és az évszámot kérjük be!



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

def kiszamit():
    fiz = eval(mezo1.get())
    ev = eval(mezo2.get())
    for i in range(2009,ev):
        fiz = fiz * 1.12
    mezo3.delete(0,END)
    mezo3.insert(0,str(fiz)+' Ft lesz a '+str(ev)+'. évben.')

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Fizetés? ')
txt2 = Label(abl1, text ='Év? ')
txt3 = Label(abl1, text ='József fizetése ')
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1, width=40)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =4, sticky =E)
gomb1.grid(row =3, column =2)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =4, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

Feladat – Háromszög

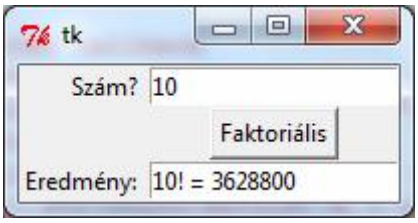
Írjunk olyan programot, amely egy háromszög három oldalából kiszámítja annak kerületét és területét!




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

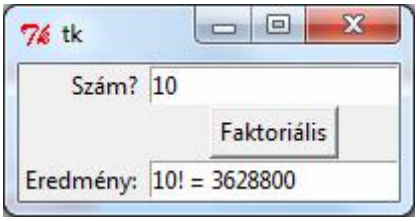
```
def kerulet(a,b,c):
    return a+b+c

def terulet(a,b,c):
    s = kerulet(a,b,c)/2
    return sqrt(s*(s-a)*(s-b)*(s-c))

def ker():
    x = eval(mezo1.get())
    y = eval(mezo2.get())
    z = eval(mezo3.get())
    mezo4.delete(0,END)
    mezo4.insert(0,str(round(kerulet(x,y,z),2)))

def ter():
    x = eval(mezo1.get())
    y = eval(mezo2.get())
    z = eval(mezo3.get())
    mezo4.delete(0,END)
    mezo4.insert(0,str(round(terulet(x,y,z),2)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='a: ')
txt2 = Label(abl1, text ='b: ')
txt3 = Label(abl1, text ='c: ')
txt4 = Label(abl1, text ='Eredmény: ')
gomb1 = Button(abl1, text='Kerület', command=ker)
gomb2 = Button(abl1, text='Terület', command=ter)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1)
txt2.grid(row =2)
txt3.grid(row =3)
txt4.grid(row =5)
gomb1.grid(row =4, column =1)
gomb2.grid(row =4, column =2)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =3, sticky =W, column =2)
mezo4.grid(row =5, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

Feladat – Faktoriális
Írjunk olyan programot, amely kiírja egy adott szám faktoriálisát!



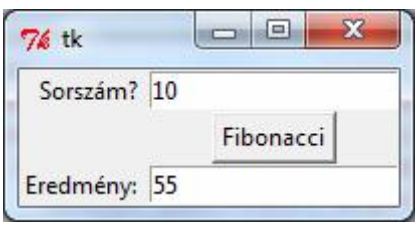

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def faktorialis(n):
    f = 1
    for i in range(1,n+1):
        f = f * i
    return f

def fakt():
    szam = eval(mezo1.get())
    mezo2.delete(0,END)
    mezo2.insert(0,str(szam)+'! = '+str(faktorialis(szam)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szám? ')
txt2 = Label(abl1, text ='Eredmény: ')
gomb1 = Button(abl1, text='Faktoriális', command=fakt)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

Feladat – Faktoriális2
Írjunk olyan programot, amely kiírja egy adott szám faktoriálisát! Alkalmazzunk rekurzív algoritmust!




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

def faktorialis(n):
    if (n == 0):
        return 1
    else:
        return n * faktorialis(n-1)

def fakt():
    szam = eval(mezo1.get())
    mezo2.delete(0,END)
    mezo2.insert(0,str(szam)+'! = '+str(faktorialis(szam)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szám? ')
txt2 = Label(abl1, text ='Eredmény: ')
gomb1 = Button(abl1, text='Faktoriális', command=fakt)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

Feladat – Fibonacci
Írjunk olyan programot, amely kiírja a Fibonacci-sorozat n-edik elemét!




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

```
from Tkinter import *

def fibonacci(n):
    if (n == 1) or (n == 2):
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)

def fib():
    szam = eval(mezo1.get())
    mezo2.delete(0,END)
    mezo2.insert(0,str(fibonacci(szam)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Sorszám? ')
txt2 = Label(abl1, text ='Eredmény: ')
gomb1 = Button(abl1, text='Faktoriális', command=fib)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

FELADATOK

1. Írj olyan függvényt, amely visszaadja egy szám kétszeresét!

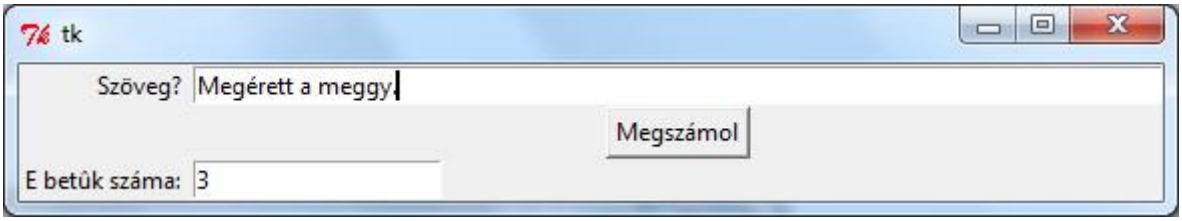
2. Írj olyan függvényt, amely egy kör sugarából visszaadja annak területét!

3. Írj olyan függvényt, amely egy gömb sugarából visszaadja annak térfogatát!

4. Írj olyan függvényt, amely visszatér a két paraméterében megadott egész szám közötti egész számok összegével, a határokat is beleértve!

5. Írj olyan függvényt, amely eldönti, hogy egy adott szám prímszám-e?

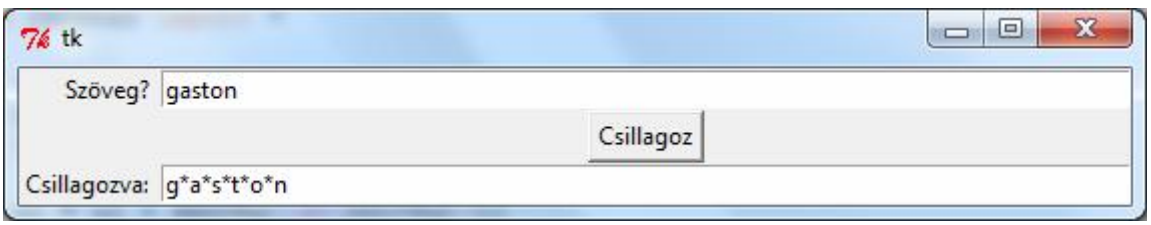
Feladat – E-betű
Írjunk olyan programot, amely egy stringben megszámolja az e betűket!




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

def ebetukSzama(szoveg):
    db = 0
    for i in range(len(szoveg)):
        if (szoveg[i] == 'e') or (szoveg[i] == 'E'):
            db = db + 1
    return db

def ebetuk():
    szov = mezo1.get()
    mezo2.delete(0,END)
    mezo2.insert(0,str(ebetukSzama(szov)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szöveg? ')
txt2 = Label(abl1, text ='E betűk száma: ')
gomb1 = Button(abl1, text='Megszámol', command=ebetuk)
mezo1 = Entry(abl1, width=80)
mezo2 = Entry(abl1)

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

# indítás :
abl1.mainloop()
```

Feladat – Csillagozás

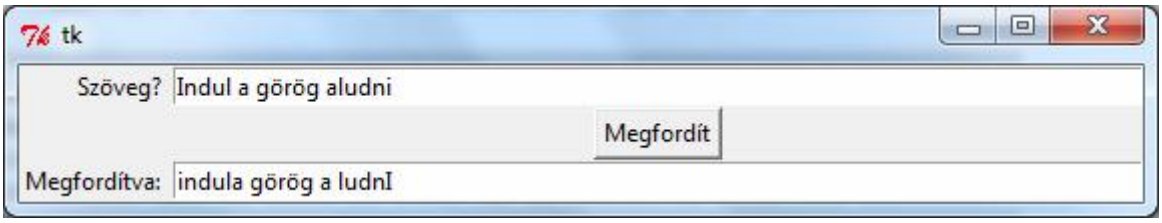
Írjunk egy programot, ami egy új változóba másol át egy karakterláncot úgy, hogy csillagot szúr be a karakterek közé. Így például, « gaston »-ból « g*a*s*t*o*n » lesz!



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

def csillagoz(szoveg):
    uj = ""
    for i in range(len(szoveg)-1):
        uj = uj + szoveg[i] + "*"
    uj = uj + szoveg[len(szoveg)-1]
    return uj

def csill():
    szov = mezo1.get()
    mezo2.delete(0,END)
    mezo2.insert(0,str(csillagoz(szov)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szöveg? ')
txt2 = Label(abl1, text ='Csillagozva: ')
gomb1 = Button(abl1, text='Csillagoz', command=csill)
mezo1 = Entry(abl1, width=80)
mezo2 = Entry(abl1, width=80)

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

# indítás :
abl1.mainloop()
```

Feladat – Megfordít

Írjunk egy programot, ami egy új változóba fordított sorrendben másolja át egy karakterlánc karaktereit. Így például « zorglub » -ból « bulgroz » lesz!

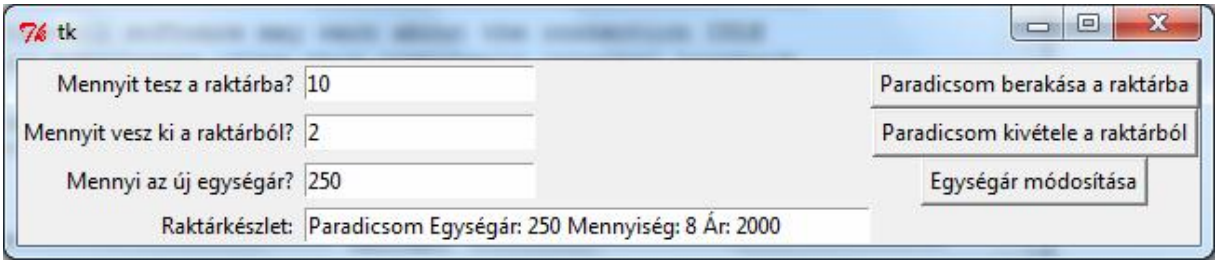



```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def fordit(szoveg):
```

```
    uj = ""
    for i in range(len(szoveg)-1,-1,-1):
        uj = uj + szoveg[i]
    return uj

def ford():
    szov = mezo1.get()
    mezo2.delete(0,END)
    mezo2.insert(0,str(fordit(szov)))

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Szöveg? ')
txt2 = Label(abl1, text ='Megfordítva: ')
gomb1 = Button(abl1, text='Megfordít', command=ford)
mezo1 = Entry(abl1, width=80)
mezo2 = Entry(abl1, width=80)

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

# indítás :
abl1.mainloop()
```

Feladat – Raktárprogram

Adott egy zöldségraktár, melyben pillanatnyilag egyetlen árut, paradicsomot raktározunk. A raktárba gyakran teszünk be, illetve veszünk ki onnan paradicsomot. A paradicsom pillanatnyi egységára 300 Ft, de ez változhat. Készítsünk olyan programot, amely segítségével rögzíteni tudjuk a megfelelő adatokat, és bármikor jelentést tudunk adni a paradicsom aktuális mennyiségéről, egységáráról, értékéről! Ki lehessen választani, hogy beteszünk-e, vagy kiveszünk paradicsomot, elletve, hogy emeljük, avagy csökkentjük a paradicsom egységárát! Minden művelet során adjunk jelentést!



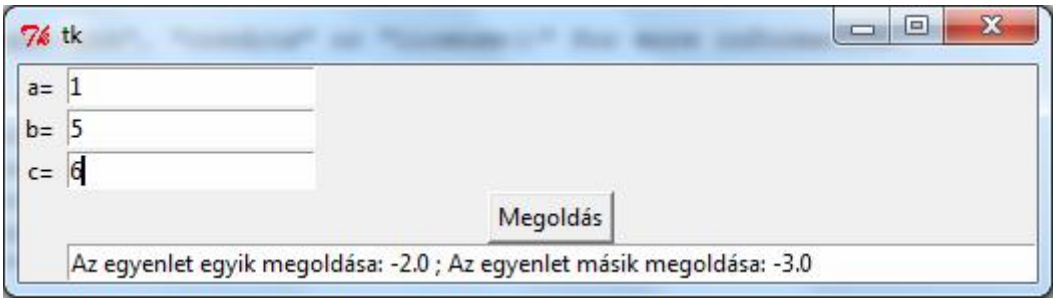

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

class Aru:
```

```
    def __init__(self,aruNev,aruEgysegar):
        self.nev = aruNev
        self.egysegar = aruEgysegar
        self.menny = 0

    def setEgysegar(self,aruEgysegar):
        if (aruEgysegar >= 0):
            self.egysegar = aruEgysegar

    def getAr(self):
        return self.menny * self.egysegar

    def hozzatesz(self,aruMenny):
        if (aruMenny > 0):
            self.menny = self.menny + aruMenny

    def elvesz(self,aruMenny):
        if (aruMenny > 0) and (aruMenny <= self.menny):
            self.menny = self.menny - aruMenny

    def __doc__(self):
        return self.nev+' Egységár: '+str(self.egysegar)+' Mennyiség: '+str(self.menny)+' Ár:
'+str(self.getAr())

aru = Aru("Paradicsom",300)

def be():
    menny = eval(mezo1.get())
    aru.hozzatesz(menny)
    mezo4.delete(0,END)
    mezo4.insert(0,aru.__doc__())

def ki():
    menny = eval(mezo2.get())
    aru.elvesz(menny)
    mezo4.delete(0,END)
    mezo4.insert(0,aru.__doc__())

def modosit():
    menny = eval(mezo3.get())
    aru.setEgysegar(menny)
    mezo4.delete(0,END)
    mezo4.insert(0,aru.__doc__())

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='Mennyit tesz a raktárba? ')
txt2 = Label(abl1, text ='Mennyit vesz ki a raktárból? ')
txt3 = Label(abl1, text ='Mennyi az új egységár? ')
```

```
txt4 = Label(abl1, text ='Raktárkészlet: ')
gomb1 = Button(abl1, text='Paradicsom berakása a raktárba', command=be)
gomb2 = Button(abl1, text='Paradicsom kivétele a raktárból', command=ki)
gomb3 = Button(abl1, text='Egységár módosítása', command=modosit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1, width=50)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
txt4.grid(row =4, sticky =E)
gomb1.grid(row =1, column =3)
gomb2.grid(row =2, column =3)
gomb3.grid(row =3, column =3)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =3, sticky =W, column =2)
mezo4.grid(row =4, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

Feladat – Másodfokú egyenlet

Írjunk olyan programot, amely kiszámítja egy másodfokú egyenlet gyökeit annak együtthatóiból!




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

class MasodfokuEgyenlet:
    def __init__(self,szam1,szam2,szam3):
        self.a = szam1
        self.b = szam2
        self.c = szam3

    def diszkriminans(self):
        return self.b * self.b - 4 * self.a * self.c
```

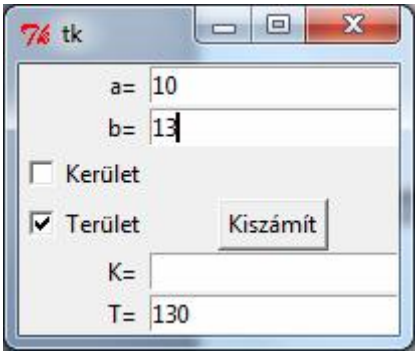
```
    def megoldas1(self):
        return (-self.b + sqrt(self.diszkriminans())) / (2 * self.a)

    def megoldas2(self):
        return (-self.b - sqrt(self.diszkriminans())) / (2 * self.a)

    def __doc__(self):
        if (self.a == 0):
            return 'Az egyenlet nem másodfokú.'
        elif (self.diszkriminans() < 0):
            return 'Az egyenletnek nincs valós megoldása.'
        else:
            return 'Az egyenlet egyik megoldása: '+str(self.megoldas1())+' ; Az egyenlet másik
megoldása: '+str(self.megoldas2())

def megold():
    a = eval(mezo1.get())
    b = eval(mezo2.get())
    c = eval(mezo3.get())
    egyenlet = MasodfokuEgyenlet(a,b,c)
    mezo4.delete(0,END)
    mezo4.insert(0,egyenlet.__doc__())

abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='a= ')
txt2 = Label(abl1, text ='b= ')
txt3 = Label(abl1, text ='c= ')
gomb1 = Button(abl1, text='Megoldás', command=megold)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1, width=80)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
gomb1.grid(row =4, column =2)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =3, sticky =W, column =2)
mezo4.grid(row =5, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

6. Widget-ek

A jelölőnégyzet

A következő két widget adategyüttesek kezelését teszi lehetővé:
– ***Checkbutton*** (jelölőnégyzet) többszörös választást tesz lehetővé,
– ***Radiobutton*** (rádiógomb) egyedi választást tesz lehetővé.
A két widget opciói :
– **command** egy metódus hozzárendelésére való, mint a ***Button*** esetén
– **variable** az érték tárolására és kinyerésére való.
A változót ***Tkinter*** változóként kell deklarálni. Ez lehetővé tesz a ***Tk*** és a Python között egy ***Tk*** által kezdeményezett interakciót (a callback elve): az ***IntVar*** és ***StringVar*** teszik lehetővé egészeket és karakterláncokat tároló objektumok létrehozását.

Feladat – Jelölőnégyzet
Írjunk olyan programot, amely attól függően számítja ki egy téglalap kerületét, illetve területét, hogy az adott jelölőnégyzetben van-e pipa!



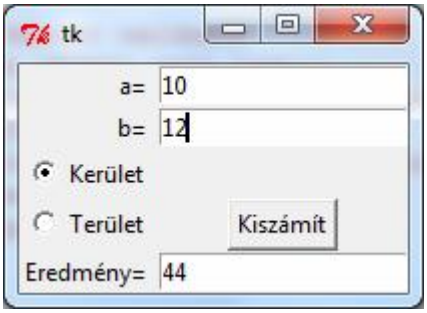

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def kiszamit():
    a = eval(mezo1.get())
    b = eval(mezo2.get())
    k = (a+b)*2
    t = a*b
    mezo3.delete(0,END)
    mezo4.delete(0,END)
    if var1.get() == 1:
        mezo3.insert(0,str(k))
    else:
        mezo3.insert(0,'')
    if var2.get() == 1:
        mezo4.insert(0,str(t))
    else:
        mezo4.insert(0,'')

abl1 = Tk()
```

```
# a widgetek létrehozása:
txt1 = Label(abl1, text ='a= ')
txt2 = Label(abl1, text ='b= ')
txt3 = Label(abl1, text ='K= ')
txt4 = Label(abl1, text ='T= ')
var1 = IntVar()
var2 = IntVar()
jelolo1 = Checkbutton(abl1, text='Kerület', variable = var1)
jelolo2 = Checkbutton(abl1, text='Terület', variable = var2)
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)
mezo4 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =5, sticky =E)
txt4.grid(row =6, sticky =E)
jelolo1.grid(row=3)
jelolo2.grid(row=4)
gomb1.grid(row =4, column =2)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =5, sticky =W, column =2)
mezo4.grid(row =6, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

A rádiógomb

Feladat – Rádiógomb

Írjunk olyan programot, amely attól függően számítja ki egy téglalap kerületét, illetve területét, hogy melyik rádiógomb van kijelölve!




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

def kiszamit():
```

```
    a = eval(mezo1.get())
    b = eval(mezo2.get())
    k = (a+b)*2
    t = a*b
    mezo3.delete(0,END)
    if var.get() == 'ker':
        mezo3.insert(0,str(k))
    if var.get() == 'ter':
        mezo3.insert(0,str(t))


abl1 = Tk()

# a widgetek létrehozása:
txt1 = Label(abl1, text ='a= ')
txt2 = Label(abl1, text ='b= ')
txt3 = Label(abl1, text ='Eredmény= ')
var = StringVar()
radio1 = Radiobutton(abl1, text='Kerület', value='ker', variable = var)
radio2 = Radiobutton(abl1, text='Terület', value='ter', variable = var)
gomb1 = Button(abl1, text='Kiszámít', command=kiszamit)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =5, sticky =E)
radio1.grid(row=3)
radio2.grid(row=4)
gomb1.grid(row =4, column =2)
mezo1.grid(row =1, sticky =W, column =2)
mezo2.grid(row =2, sticky =W, column =2)
mezo3.grid(row =5, sticky =W, column =2)

# indítás :
abl1.mainloop()
```

Listadoboz

A Listbox widget elemlisták kezelését teszi lehetővé. Elemek hozzáadására, törlésére és kiválasztására szolgáló metódusai vannak.
Listák manipulálása
— ***index*** egy listában:'0'-tól END-ig,
— ***curselection***() visszatérési értéke a kiválasztott index
— ***get***(*idx*) visszatérési értéke az *idx* indexen lévő elem
— ***get***(ACTIVE) visszatérési értéke a kiválasztott elem
— ***get***('1.0', END+'-1c') visszatérési értéke az egész szövegzóna tartalma,
— ***insert***(*idx*, *elem*) az *idx* pozícióba beszúrja az elemet,

— ***delete***(*kezdet*, [*vég*]) a *kezdet* és a *vég* indexek közötti elemeket törli (a vég paraméter opcionális).

Feladat – Listadoboz
Írjunk olyan programot, amely képes egy szövegbeviteli mezőből elemeket beszúrni egy listadobozba, illetve képes a kijelölt elemet törölni!


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

def torol():
    index = lista1.curselection()
    lista1.delete(index)

def hozzaad():
    a = mezo1.get()
    lista1.insert(0,a)

abl1 = Tk()

# a widgetek létrehozása:
lista1 = Listbox(abl1)
gomb1 = Button(abl1, text='Töröl', command=torol)
gomb2 = Button(abl1, text='Hozzáad', command=hozzaad)
mezo1 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
lista1.grid(row=1)
gomb1.grid(row =1, column =2)
gomb2.grid(row =2, column =2)
mezo1.grid(row =2, sticky =W)

# indítás :
abl1.mainloop()
```

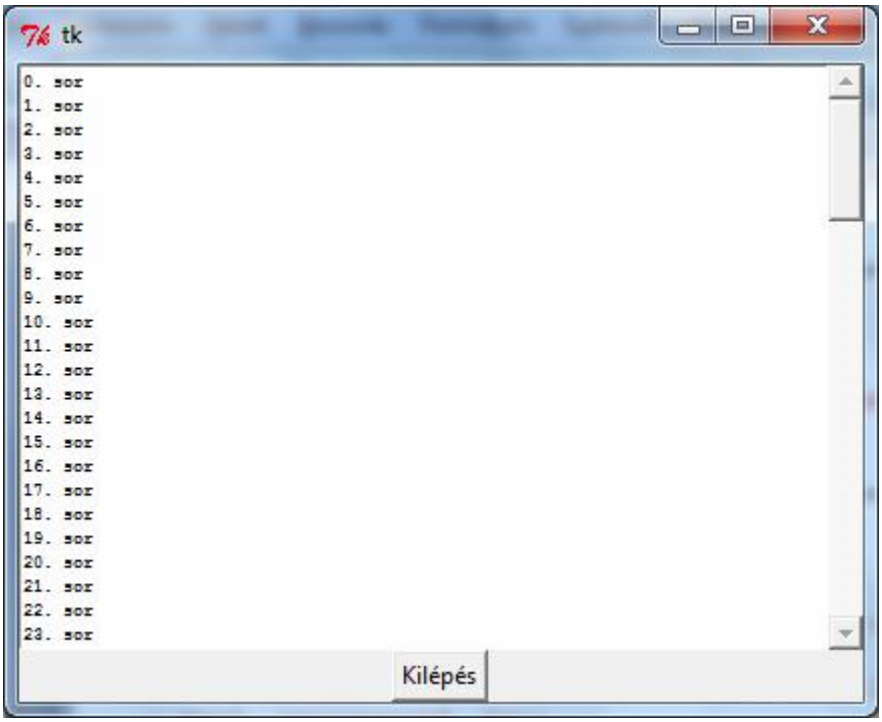
Szövegdoboz

A ***Text*** widget többsoros string formájú szövegbevitelt és szövegmanipulációt kínál. A szövegdoboz WRAP=WORD opciója meghatározza, hogy a szövegzóna tartalmának feldarabolása a szóhatárokon történik (amikor egy szó nem tartható egy soron belül, akkor a szövegzóna automatikusan sort vált).
Szöveg manipulációk
– *index* egy szövegzónán:1.0-tól (sor.oszlop) END-ig,
– az INSERT megadja a cursor beszúrási pozícióját,
– SEL_FIRST, SEL_LAST a kiválasztás elejének és végének pozícióját tartalmazza
– ***get***(*kezdet*, *vég*) visszatérési értéke az adott pozícióbeli szöveg,
– ***get***('1.0', END+'-1c') visszatérési értéke az egész szövegzóna tartalma,
– ***insert***('li.co', *txt*) a megadott pozícióba beszúrja a *txt* szöveget,
– ***delete***(*kezdet*, [*vég*]) a kezdet és a vég pozíciók közötti szöveget törli (a vég paraméter opcionális).
– ***search***(*string*, *kezdet*, *vég*) visszatérési értéke a *string* szövegzónabeli kezdő pozíciója
– ***see***('li.co') addig görgeti a kiírt szöveget, amíg a megadott szövegpozíció láthatóvá nem válik
– egy szövegzónabeli pozícióhoz viszonyított relatív elmozdulás :+/- bizonyos számú 'c' (karakter), 'w'szó (word), vagy 'l' sor (line).

Görgetősáv

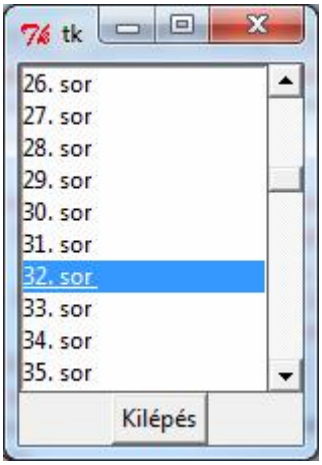
A ***ScrollBar*** widget a ***Text*** és a ***ListBox*** (vízszintes és függőleges) görgetését teszi lehetővé. Az utóbbi widgetek görgető metódusait használja. Először érdemes létrehozni egy ***frame***-et a lista és a görgetősáv csoportba foglalásához. Az utóbbiakat úgy érdemes konfigurálnuk, hogy a lista a görgetősáv használatának megfelelően váljon láthatóvá.

Feladat – Szövegdoboz + görgetősáv
Írjunk olyan programot, amely egy szövegdobozba beszúr 100 sort, és ellátja görgetősávval!



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

abl1 = Tk()

# a widgetek létrehozása:
frame1 = Frame(abl1)
szoveg1 = Text(frame1)
sbar = Scrollbar(frame1)
sbar.config(command = szoveg1.yview)
szoveg1.config(yscrollcommand = sbar.set)
gomb1 = Button( abl1, text ='Kilépés', command = abl1.destroy)

# laptördelés a'pack' metódus segítségével :
frame1.pack(side =TOP)
sbar.pack(side =RIGHT, fill =Y)
szoveg1.pack(side =LEFT, expand =YES, fill = BOTH)
gomb1.pack(side =BOTTOM)

# indítás :
idx = 0
for i in range(100):
    idx = idx+1
    szoveg1.insert(str(idx)+'.0', str(i) +'. sor\n')

abl1.mainloop()
```

Feladat – Listadoboz + görgetősáv
Írjunk olyan programot, amely egy listadobozba beszúr 100 sort, és ellátja görgetősávval!



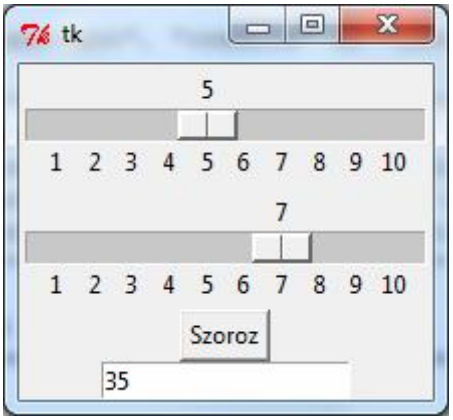

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

abl1 = Tk()

# a widgetek létrehozása:
frame1 = Frame(abl1)
```

```
lista1 = Listbox(frame1)
sbar = Scrollbar(frame1)
sbar.config(command = lista1.yview)
lista1.config(yscrollcommand = sbar.set)
gomb1 = Button( abl1, text ='Kilépés', command = abl1.destroy)

# laptördelés a'pack' metódus segítségével :
frame1.pack(side =TOP)
sbar.pack(side =RIGHT, fill =Y)
lista1.pack(side =LEFT, expand =YES, fill = BOTH)
gomb1.pack(side =BOTTOM)

# indítás :
idx = 0
for i in range(100):
    idx = idx+1
    lista1.insert(str(idx), str(i) +'. sor\n')

abl1.mainloop()
```

Scale

A skála widget lehetővé teszi a felhasználó számára, hogy számértékeket csúszka segítségével határozzon meg. Két típusa lehet vízszintes és függőleges.

Feladat – Szorzótábla
Írjunk olyan programot, amely két scale által meghatározott érték szorzatát írja ki egy szövegbeviteli mezőbe!



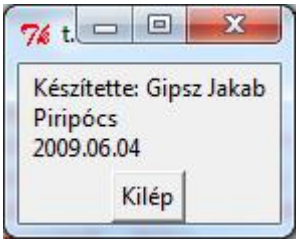

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

def szoroz():
    a = scale1.get()
    b = scale2.get()
    mezo1.delete(0,END)
    mezo1.insert(0,str(a*b))
```

```
abl1 = Tk()

# a widgetek létrehozása:
scale1 = Scale(abl1, from_ =1, to =10, tickinterval =1, orient=HORIZONTAL,
length=200)
scale2 = Scale(abl1, from_ =1, to =10, tickinterval =1, orient=HORIZONTAL,
length=200)
gomb1 = Button(abl1, text='Szoroz', command=szoroz)
mezo1 = Entry(abl1)

# laptördelés a'pack' metódus segítségével :
scale1.pack()
scale2.pack()
gomb1.pack()
mezo1.pack()

# indítás :
abl1.mainloop()
```

Message

A Label widgethez képest több sor tárolására alkalmas.

Feladat – Névjegy
Készítsünk a message widget segítségével olyan ablakot, amely a szerző legfontosabb adatait tartalmazza több sorban!

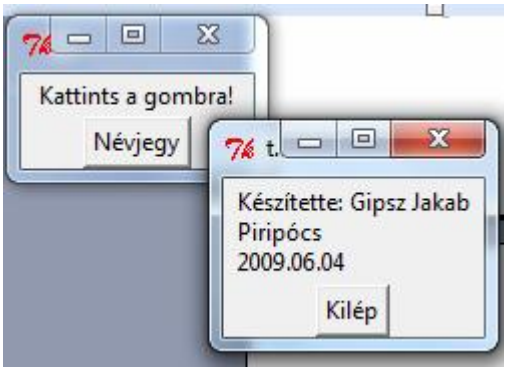



```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

abl1 = Tk()

# a widgetek létrehozása:
uz1 = Message(abl1, text='Készítette: Gipsz Jakab\nPiripócs\n2009.06.04', width=300)
gomb1 = Button(abl1, text='Kilép', command=abl1.destroy)

# laptördelés a'pack' metódus segítségével :
uz1.pack()
gomb1.pack()

# indítás :
abl1.mainloop()
```

7. Többablakos alkalmazás (Toplevel)

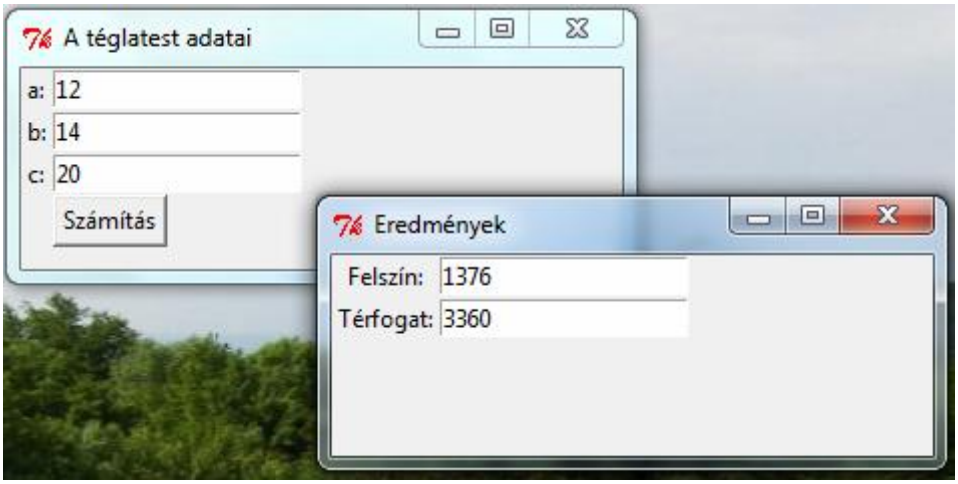
Többablakos alkalmazás akár Tk, akár Toplevel segítségével is készíthető.

Feladat – Több ablak
Készítsünk olyan ablakot, amelynek egy nyomógombjára kattintva egy másik ablak nyílik meg!




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

abl1 = Tk()

def ujablak():
    abl2 = Toplevel(abl1)
    uz2 = Message(abl2, text='Készítette: Gipsz Jakab\nPiripócs\n2009.06.04', width=300)
    gomb2 = Button(abl2, text='Kilép', command=abl2.destroy)
    uz2.pack()
    gomb2.pack()
    abl2.mainloop()

# a widgetek létrehozása
szoveg1 = Label(abl1, text ='Kattints a gombra!')
gomb1 = Button(abl1, text ='Névjegy', command=ujablak)

# laptördelés a'pack' metódus segítségével :
szoveg1.pack()
gomb1.pack()

# indítás :
abl1.mainloop()
```

Feladat – Több ablak2
Készítsünk olyan alkalmazást, amelyben bekérjük egy téglatest egy csúcsba futó éleit, majd a felszínt és a térfogatot egy új ablakban írjuk ki!




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

abl1 = Tk()
abl1.title('A téglatest adatai')
abl1.minsize(width =300, height=100)

def ujablak():
    abl2 = Toplevel(abl1)
    abl2.title('Eredmények')
    abl2.minsize(width =300, height=100)

    # a widgetek létrehozása
    sz1 = Label(abl2, text ='Felszín:')
    sz2 = Label(abl2, text ='Térfogat:')
    m1 = Entry(abl2)
    m2 = Entry(abl2)

    # laptördelés a'grid' metódus segítségével :
    sz1.grid(row =1)
    sz2.grid(row =2)
    m1.grid(row =1, column =2, sticky =W)
    m2.grid(row =2, column =2, sticky =W)

    a = eval(mezo1.get())
    b = eval(mezo2.get())
    c = eval(mezo3.get())

    felszin = 2*(a*b+a*c+b*c)
    terfogat = a*b*c

    m1.delete(0,END)
    m1.insert(0,str(felszin))
    m2.delete(0,END)
    m2.insert(0,str(terfogat))
```

```
    abl2.mainloop()


# a widgetek létrehozása
szoveg1 = Label(abl1, text ='a:')
szoveg2 = Label(abl1, text ='b:')
szoveg3 = Label(abl1, text ='c:')
gomb1 = Button(abl1, text ='Számítás', command=ujablak)
mezo1 = Entry(abl1)
mezo2 = Entry(abl1)
mezo3 = Entry(abl1)

# laptördelés a'grid' metódus segítségével :
szoveg1.grid(row =1)
szoveg2.grid(row =2)
szoveg3.grid(row =3)
gomb1.grid(row =4, column= 2, sticky =W)
mezo1.grid(row =1, column =2, sticky =W)
mezo2.grid(row =2, column =2, sticky =W)
mezo3.grid(row =3, column =2, sticky =W)

# indítás :
abl1.mainloop()
```

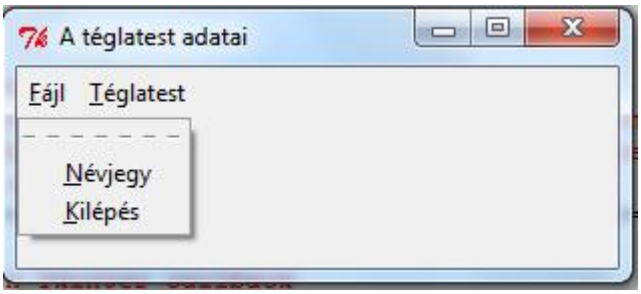
8. Menükészítés

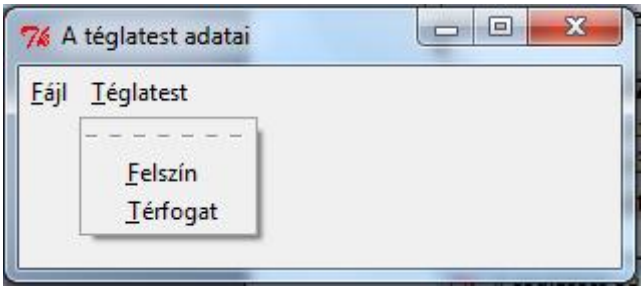
A menük definiálása egy ablakban, egy dedikált ***Frame*** segítségével történik. A ***Menubutton*** widget-tel lehet definiálni a menüsáv egy elemét és a ***Menu*** widget-tel lehet definiálni az elemhez kapcsolt parancsokat.

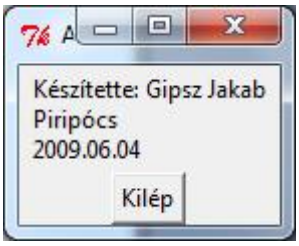
A sávban a menük szervezése a következő változók és függvények segítségével történik:

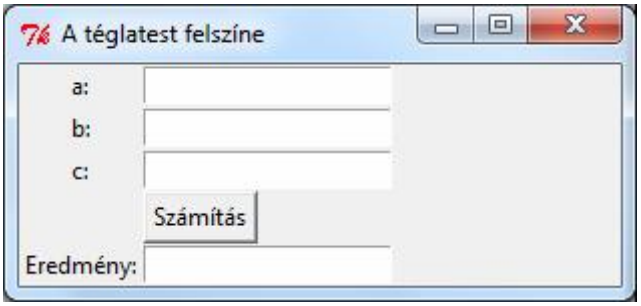
− **bd, refilef** a menüpontok száma és a domborzat (RAISED, SUNKEN, RIDGE),
− ***add_sepataror***() egy elválasztó vonalat illeszt be a menü két parancsa közé,
− ***add_cascade***() almenü létrehozása,
− ***entryconfig***(*num*, **state**=) beállítja az elérhető menüpontok (alapértelmezetten ENABLED), illetve a nem elérhető menüpontok (DISABLED) megjelenését.

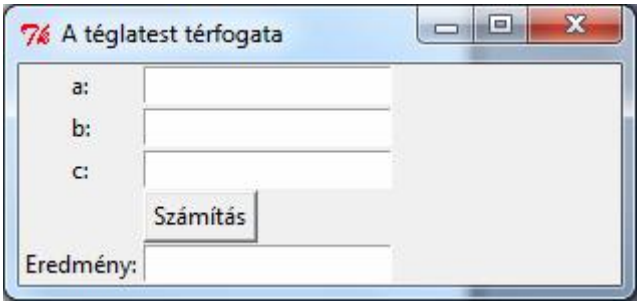
Feladat – Menü
Készítsünk olyan alkalmazást, amelyben menük segítenek annak a kiválasztásában, hogy egy téglatest egy csúcsba futó éleiből, annak felszínét vagy a térfogatát számítsuk ki, illetve, hogy a program készítőjének névjegyét tekintsük meg, vagy lépjünk ki a programból!











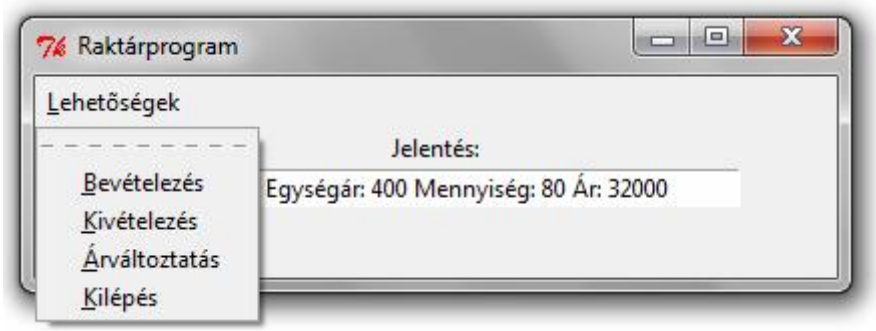

```
# -*- coding: ISO-8859-2 -*-
from Tkinter import *

#Névjegy ablak
def nevjegy():
    abl2 = Toplevel(foablak)
    uz2 = Message(abl2, text='Készítette: Gipsz Jakab\nPiripócs\n2009.06.04', width=300)
    gomb2 = Button(abl2, text='Kilép', command=abl2.destroy)
    uz2.pack()
    gomb2.pack()
    abl2.mainloop()
#Névjegy ablak vége

#Felszín ablak
def felszin():
```

```
    def szamit():
        a = eval(mezo1.get())
        b = eval(mezo2.get())
        c = eval(mezo3.get())
        felszin = 2*(a*b+a*c+b*c)
        mezo4.delete(0,END)
        mezo4.insert(0,str(felszin))

    abl3 = Toplevel(foablak)
    abl3.title('A téglatest felszíne')
    abl3.minsize(width =300, height=100)
    szoveg1 = Label(abl3, text ='a:')
    szoveg2 = Label(abl3, text ='b:')
    szoveg3 = Label(abl3, text ='c:')
    szoveg4 = Label(abl3, text ='Eredmény:')
    gomb1 = Button(abl3, text ='Számítás', command=szamit)
    mezo1 = Entry(abl3)
    mezo2 = Entry(abl3)
    mezo3 = Entry(abl3)
    mezo4 = Entry(abl3)
    szoveg1.grid(row =1)
    szoveg2.grid(row =2)
    szoveg3.grid(row =3)
    szoveg4.grid(row =5)
    gomb1.grid(row =4, column= 2, sticky =W)
    mezo1.grid(row =1, column =2, sticky =W)
    mezo2.grid(row =2, column =2, sticky =W)
    mezo3.grid(row =3, column =2, sticky =W)
    mezo4.grid(row =5, column =2, sticky =W)
    abl3.mainloop()
#Felszín ablak vége

#Térfogat ablak
def terfogat():

    def szamit():
        a = eval(mezo1.get())
        b = eval(mezo2.get())
        c = eval(mezo3.get())
        terfogat = a*b*c
        mezo4.delete(0,END)
        mezo4.insert(0,str(terfogat))

    abl3 = Toplevel(foablak)
    abl3.title('A téglatest térfogata')
    abl3.minsize(width =300, height=100)
    szoveg1 = Label(abl3, text ='a:')
    szoveg2 = Label(abl3, text ='b:')
    szoveg3 = Label(abl3, text ='c:')
    szoveg4 = Label(abl3, text ='Eredmény:')
```

```
    gomb1 = Button(abl3, text ='Számítás', command=szamit)
    mezo1 = Entry(abl3)
    mezo2 = Entry(abl3)
    mezo3 = Entry(abl3)
    mezo4 = Entry(abl3)
    szoveg1.grid(row =1)
    szoveg2.grid(row =2)
    szoveg3.grid(row =3)
    szoveg4.grid(row =5)
    gomb1.grid(row =4, column= 2, sticky =W)
    mezo1.grid(row =1, column =2, sticky =W)
    mezo2.grid(row =2, column =2, sticky =W)
    mezo3.grid(row =3, column =2, sticky =W)
    mezo4.grid(row =5, column =2, sticky =W)
    abl3.mainloop()
#Térfogat ablak vége

#Főablak
foablak = Tk()
foablak.title('A téglatest adatai')
foablak.minsize(width =300, height=100)

menusor = Frame(foablak)
menusor.pack(side =TOP, fill =X)

menu1 = Menubutton(menusor, text ='Fájl', underline =0)
menu1.pack(side = LEFT )
fajl = Menu(menu1)
fajl.add_command(label ='Névjegy', command = nevjegy, underline =0)
fajl.add_command(label ='Kilépés', command = foablak.destroy, underline =0)
menu1.config(menu = fajl)

menu2 = Menubutton(menusor, text ='Téglatest', underline =0)
menu2.pack(side = LEFT )
teglatest = Menu(menu2)
teglatest.add_command(label ='Felszín', command = felszin, underline =0)
teglatest.add_command(label ='Térfogat', command = terfogat, underline =0)
menu2.config(menu = teglatest)

foablak.mainloop()
```

Feladat – Raktárprogram

Adott egy zöldségraktár, melyben pillanatnyilag egyetlen árut, paradicsomot raktározunk. A raktárba gyakran teszünk be, illetve veszünk ki onnan paradicsomot. A paradicsom pillanatnyi egységára 300 Ft, de ez változhat Készítsünk olyan programot, amely segítségével rögzíteni tudjuk a megfelelő adatokat, és bármikor jelentést tudunk adni a paradicsom aktuális mennyiségéről, egységáráról, értékéről! Menüből lehessen kiválasztani, hogy beteszünk-e, vagy kiveszünk paradicsomot, elletve, hogy emeljük, avagy csökkentjük a paradicsom egységárát!




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

class Aru:
    def __init__(self,aruNev,aruEgysegar):
        self.nev = aruNev
        self.egysegar = aruEgysegar
        self.menny = 0

    def setEgysegar(self,aruEgysegar):
        if (aruEgysegar >= 0):
            self.egysegar = aruEgysegar

    def getAr(self):
        return self.menny * self.egysegar
```

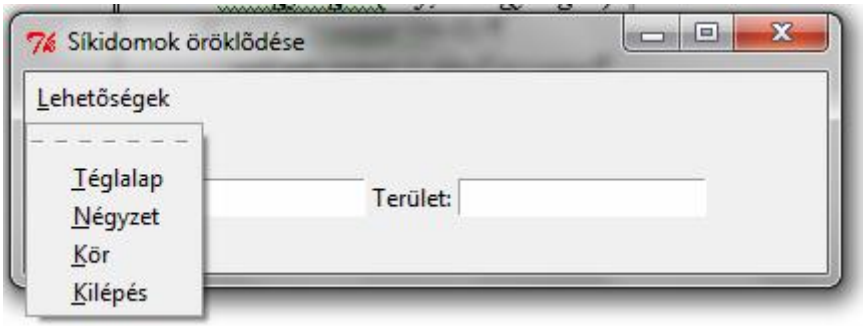
```
    def hozzatesz(self,aruMenny):
        if (aruMenny > 0):
            self.menny = self.menny + aruMenny

    def elvesz(self,aruMenny):
        if (aruMenny > 0) and (aruMenny <= self.menny):
            self.menny = self.menny - aruMenny

    def __doc__(self):
        mezo1.delete(0,END)
        mezo1.insert(0,self.nev+' Egységár: '+str(self.egysegar)+' Mennyiség: '+str(self.menny)+'
Ár: '+str(self.getAr()))

aru = Aru("Paradicsom",300)

#Bevételezés ablak

def bevetelezes():

    def rogzit():
        menny = eval(m1.get())
        aru.hozzatesz(menny)
        aru.__doc__()
        abl2.destroy()

    abl2 = Toplevel(foablak)
    abl2.title('Bevételezés')
    abl2.minsize(width =300, height=100)
    sz1 = Label(abl2, text ='Mennyiség:')
    g1 = Button(abl2, text ='Rögzít', command=rogzit)
    m1 = Entry(abl2)
    sz1.grid(row =1)
    g1.grid(row =2, column= 2, sticky =W)
    m1.grid(row =1, column =2, sticky =W)
    abl2.mainloop()

#Bevételezés ablak vége

#Kivételezés ablak

def kivetelezes():

    def rogzit():
        menny = eval(m1.get())
        aru.elvesz(menny)
        aru.__doc__()
        abl3.destroy()

    abl3 = Toplevel(foablak)
```

```
    abl3.title('Kivételezés')
    abl3.minsize(width =300, height=100)
    sz1 = Label(abl3, text ='Mennyiség:')
    g1 = Button(abl3, text ='Rögzít', command=rogzit)
    m1 = Entry(abl3)
    sz1.grid(row =1)
    g1.grid(row =2, column= 2, sticky =W)
    m1.grid(row =1, column =2, sticky =W)
    abl3.mainloop()

#Kivételezés ablak vége

#Árváltozás ablak

def arvaltozas():

    def rogzit():
        ujar = eval(m1.get())
        aru.setEgysegar(ujar)
        aru.__doc__()
        abl4.destroy()

    abl4 = Toplevel(foablak)
    abl4.title('Árváltozás')
    abl4.minsize(width =300, height=100)
    sz1 = Label(abl4, text ='Új ár:')
    g1 = Button(abl4, text ='Rögzít', command=rogzit)
    m1 = Entry(abl4)
    sz1.grid(row =1)
    g1.grid(row =2, column= 2, sticky =W)
    m1.grid(row =1, column =2, sticky =W)
    abl4.mainloop()

#Árváltozás ablak vége

#Főablak

foablak = Tk()
foablak.title('Raktárprogram')
foablak.minsize(width =400, height=100)

menusor = Frame(foablak)
menusor.pack(side =TOP, fill =X)

menu1 = Menubutton(menusor, text ='Lehetőségek', underline =0)
menu1.pack(side = LEFT )
fajl = Menu(menu1)
fajl.add_command(label ='Bevételezés', command = bevetelezes, underline =0)
fajl.add_command(label ='Kivételezés', command = kivetelezes, underline =0)
fajl.add_command(label ='Árváltoztatás', command = arvaltozas, underline =0)
```

```
fajl.add_command(label ='Kilépés', command = foablak.destroy, underline =0)
menu1.config(menu = fajl)

szoveg1 = Label(foablak, text ='Jelentés:')
mezo1 = Entry(foablak, width =50)
szoveg1.pack()
mezo1.pack()
aru.__doc__()

foablak.mainloop()
```

Feladat – Síkidomok öröklődése

Írjunk olyan programot, amely kiszámítja a négyzet, a téglalap és a kör kerületét, területét! Az egyes síkidomok osztályok legyenek, megfelelő metódusokkal, alkalmazzuk az öröklődést!



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

class Teglalap:
    def __init__(self,szam1,szam2):
        self.a = szam1
        self.b = szam2

    def kerulet(self):
        return 2*(self.a + self.b)

    def terulet(self):
        return self.a * self.b

    def __doc__(self):
        mezo1.delete(0,END)
        mezo1.insert(0,str(self.kerulet()))
        mezo2.delete(0,END)
        mezo2.insert(0,str(self.terulet()))

class Negyzet(Teglalap):
    def __init__(self,szam1):
        Teglalap.__init__(self,szam1,szam1)

    def __doc__(self):
        Teglalap.__doc__(self)

class Kor(Negyzet):
    def kerulet(self):
        return 2 * self.a * pi

    def terulet(self):
        return self.a * self.a * pi

    def __doc__(self):
        Negyzet.__doc__(self)

#Téglalap ablak
def tegla1():

    def szamit():
        a = eval(m1.get())
        b = eval(m2.get())
        teglalap = Teglalap(a,b)
        teglalap.__doc__()
        abl2.destroy()

    abl2 = Toplevel(foablak)
    abl2.title('Téglalap')
```

```
    abl2.minsize(width =300, height=100)
    sz1 = Label(abl2, text ='a:')
    sz2 = Label(abl2, text ='b:')
    g1 = Button(abl2, text ='Számít', command=szamit)
    m1 = Entry(abl2)
    m2 = Entry(abl2)
    sz1.grid(row =1)
    sz2.grid(row =2)
    g1.grid(row =3, column= 2, sticky =W)
    m1.grid(row =1, column =2, sticky =W)
    m2.grid(row =2, column =2, sticky =W)
    abl2.mainloop()

#Téglalap ablak
def negyz1():

    def szamit():
        a = eval(m1.get())
        negyzet = Negyzet(a)
        negyzet.__doc__()
        abl2.destroy()

    abl2 = Toplevel(foablak)
    abl2.title('Négyzet')
    abl2.minsize(width =300, height=100)
    sz1 = Label(abl2, text ='a:')
    g1 = Button(abl2, text ='Számít', command=szamit)
    m1 = Entry(abl2)
    sz1.grid(row =1)
    g1.grid(row =2, column= 2, sticky =W)
    m1.grid(row =1, column =2, sticky =W)
    abl2.mainloop()

#Kör ablak
def kor1():

    def szamit():
        r = eval(m1.get())
        kor = Kor(r)
        kor.__doc__()
        abl2.destroy()

    abl2 = Toplevel(foablak)
    abl2.title('Kör')
    abl2.minsize(width =300, height=100)
    sz1 = Label(abl2, text ='r:')
    g1 = Button(abl2, text ='Számít', command=szamit)
    m1 = Entry(abl2)
    sz1.grid(row =1)
    g1.grid(row =2, column= 2, sticky =W)
```

```
    m1.grid(row =1, column =2, sticky =W)
    abl2.mainloop()

#Főablak
foablak = Tk()
foablak.title('Síkidomok öröklődése')
foablak.minsize(width =400, height=100)

menusor = Frame(foablak)
menusor.pack(side =TOP, fill =X)

menu1 = Menubutton(menusor, text ='Lehetőségek', underline =0)
menu1.pack(side = LEFT )
fajl = Menu(menu1)
fajl.add_command(label ='Téglalap', command = tegla1, underline =0)
fajl.add_command(label ='Négyzet', command = negyz1, underline =0)
fajl.add_command(label ='Kör', command = kor1, underline =0)
fajl.add_command(label ='Kilépés', command = foablak.destroy, underline =0)
menu1.config(menu = fajl)

szoveg1 = Label(foablak, text ='Kerület:')
szoveg2 = Label(foablak, text ='Terület:')
mezo1 = Entry(foablak)
mezo2 = Entry(foablak)
szoveg1.pack(side =LEFT)
mezo1.pack(side =LEFT)
szoveg2.pack(side =LEFT)
mezo2.pack(side =LEFT)

foablak.mainloop()
```

9. Grafika a programban, Canvas

can1 = **Canvas**(*abl1*,**bg**='dark grey',**height**=400,**width**=400)
Festővászon:

- bg : háttérszín
- height : magasság
- width : szélesség

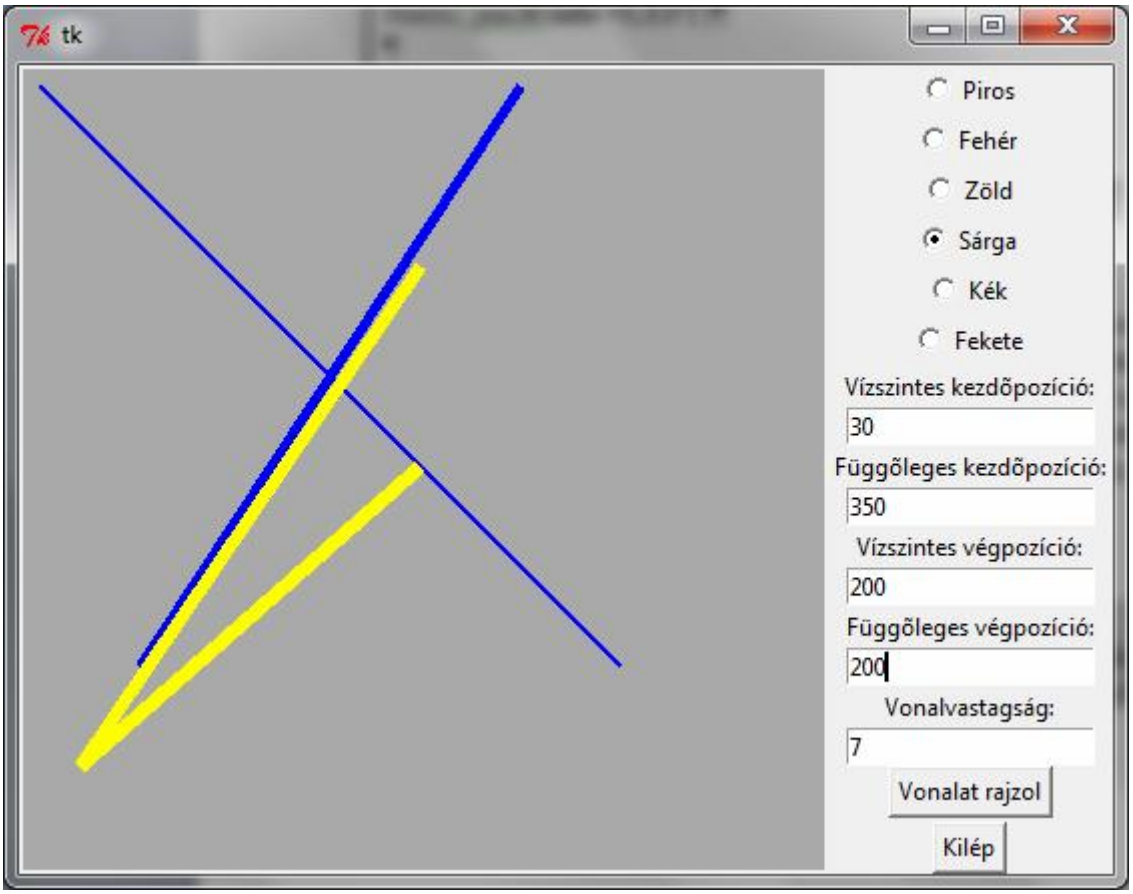
Vonal rajzolása

canvas.***create_line***(*x1*,*y1*,*x2*,*y2*, **width**=*vastagság*, **fill**=*color*)

- *x1* : a vonal egyik végének a festővászon bal szélétől mért távolsága
- *y1* : a vonal egyik végének a festővászon tetejétől mért távolsága
- *x2* : a vonal másik végének a festővászon bal szélétől mért távolsága
- *y2* : a vonal másik végének a festővászon tetejétől mért távolsága
- *vastagság* : a vonal vastagsága
- *color* : a vonal színe

Feladat – Vonalak rajzolása

Írjunk olyan programot, amely vonalakat rajzol, a vonalak végpontjainak koordinátái, a vonalak vastagsága, valamint a színe futás közben legyen beállítható!




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

def vonalatrajzol():
    x1 = eval(m1.get())
    y1 = eval(m2.get())
    x2 = eval(m3.get())
    y2 = eval(m4.get())
    vast = eval(m5.get())
    color = col.get()
    can1.create_line(x1,y1,x2,y2, width=vast, fill=color)

abl1 = Tk()
can1 = Canvas(abl1,bg='dark grey',height=400,width=400)
can1.pack(side=LEFT)

gomb1 = Button(abl1,text='Kilép',command=abl1.destroy)

col = StringVar()
radio1 = Radiobutton(abl1, text='Piros', value='red', variable = col)
radio2 = Radiobutton(abl1, text='Fehér', value='white', variable = col)
radio3 = Radiobutton(abl1, text='Zöld', value='green', variable = col)
```

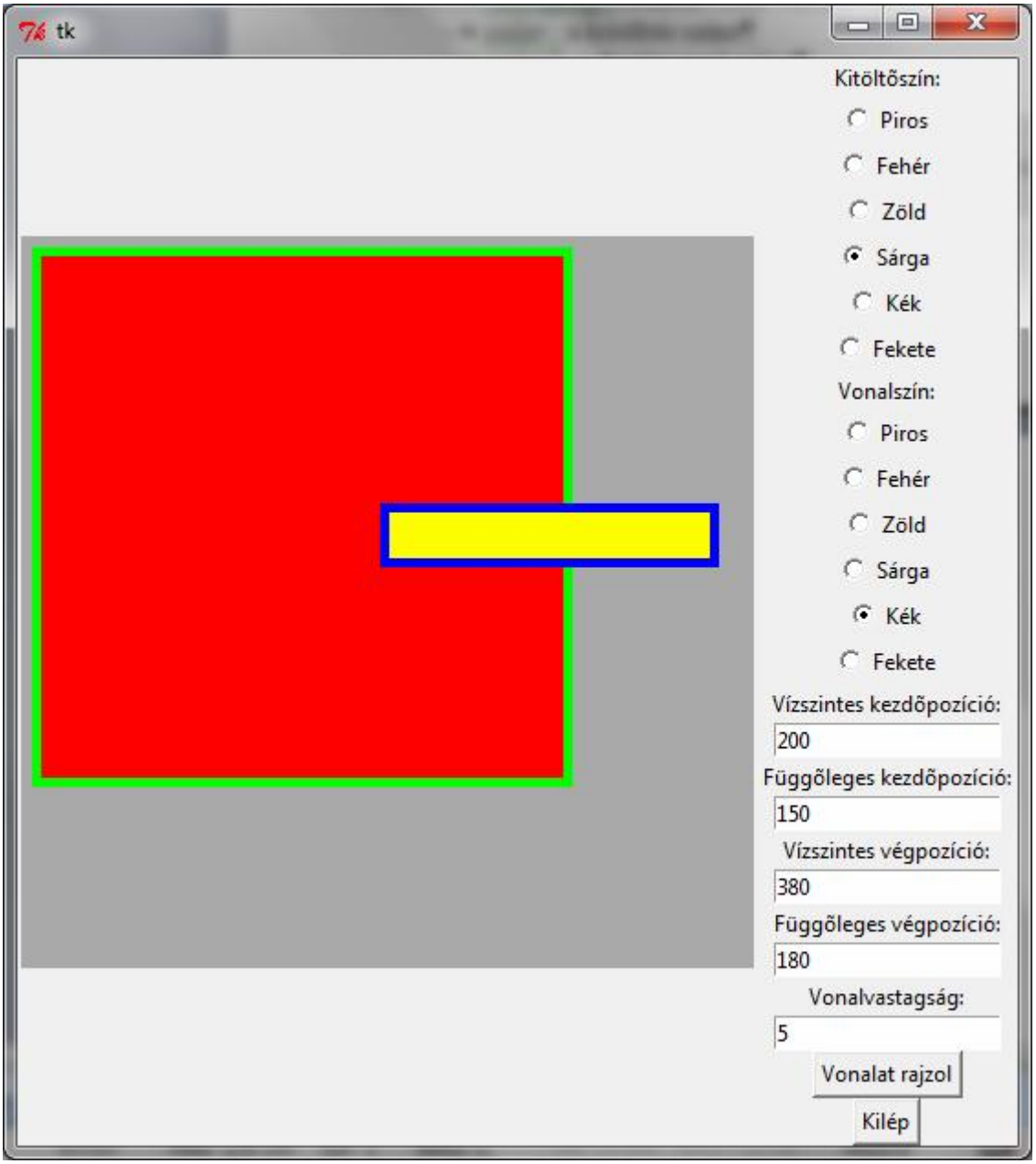
```
radio4 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col)
radio5 = Radiobutton(abl1, text='Kék', value='blue', variable = col)
radio6 = Radiobutton(abl1, text='Fekete', value='black', variable = col)
radio1.pack()
radio2.pack()
radio3.pack()
radio4.pack()
radio5.pack()
radio6.pack()

sz1 = Label(abl1, text ='Vízszintes kezdőpozíció:')
sz1.pack()
m1 = Entry(abl1)
m1.pack()
sz2 = Label(abl1, text ='Függőleges kezdőpozíció:')
sz2.pack()
m2 = Entry(abl1)
m2.pack()

sz3 = Label(abl1, text ='Vízszintes végpozíció:')
sz3.pack()
m3 = Entry(abl1)
m3.pack()
sz4 = Label(abl1, text ='Függőleges végpozíció:')
sz4.pack()
m4 = Entry(abl1)
m4.pack()

sz5 = Label(abl1, text ='Vonalvastagság:')
sz5.pack()
m5 = Entry(abl1)
m5.pack()

gomb1.pack(side=BOTTOM)
gomb2 = Button(abl1,text='Vonalat rajzol',command=vonalatrajzol)
gomb2.pack()

abl1.mainloop()
```

Téglalap rajzolása

canvas.***create_rectangle***(*x1*,*y1*,*x2*,*y2*, **width**=*vast*, **fill**=*color*, **outline**=*color2*)

- *x1* : a téglalap egyik csúcsának a festővászon bal szélétől mért távolsága
- *y1* : a téglalap egyik csúcsának a festővászon tetejétől mért távolsága
- *x2* : a téglalap átellenes csúcsának a festővászon bal szélétől mért távolsága
- *y2* : a téglalap átellenes csúcsának a festővászon tetejétől mért távolsága
- *vastagság* : a határvonal vastagsága
- *color* : a kitöltés színe
- *color2* : a határvonal színe

Feladat – Téglalapok rajzolása
Írjunk olyan programot, amely téglalapokat rajzol, a téglalapok adatai futás közben legyenek beállíthatók!




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

def vonalatrajzol():
    x1 = eval(m1.get())
    y1 = eval(m2.get())
    x2 = eval(m3.get())
    y2 = eval(m4.get())
    vast = eval(m5.get())
    color = col.get()
    color2 = col2.get()
```

```
   can1.create_rectangle(x1,y1,x2,y2, width=vast, fill=color, outline=color2)

abl1 = Tk()
can1 = Canvas(abl1,bg='dark grey',height=400,width=400)
can1.pack(side=LEFT)

gomb1 = Button(abl1,text='Kilép',command=abl1.destroy)

szov1 = Label(abl1, text ='Kitöltőszín:')
szov1.pack()
col = StringVar()
radio1 = Radiobutton(abl1, text='Piros', value='red', variable = col)
radio2 = Radiobutton(abl1, text='Fehér', value='white', variable = col)
radio3 = Radiobutton(abl1, text='Zöld', value='green', variable = col)
radio4 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col)
radio5 = Radiobutton(abl1, text='Kék', value='blue', variable = col)
radio6 = Radiobutton(abl1, text='Fekete', value='black', variable = col)
radio1.pack()
radio2.pack()
radio3.pack()
radio4.pack()
radio5.pack()
radio6.pack()

szov2 = Label(abl1, text ='Vonalszín:')
szov2.pack()
col2 = StringVar()
radio7 = Radiobutton(abl1, text='Piros', value='red', variable = col2)
radio8 = Radiobutton(abl1, text='Fehér', value='white', variable = col2)
radio9 = Radiobutton(abl1, text='Zöld', value='green', variable = col2)
radio10 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col2)
radio11 = Radiobutton(abl1, text='Kék', value='blue', variable = col2)
radio12 = Radiobutton(abl1, text='Fekete', value='black', variable = col2)
radio7.pack()
radio8.pack()
radio9.pack()
radio10.pack()
radio11.pack()
radio12.pack()

sz1 = Label(abl1, text ='Vízszintes kezdőpozíció:')
sz1.pack()
m1 = Entry(abl1)
m1.pack()
sz2 = Label(abl1, text ='Függőleges kezdőpozíció:')
sz2.pack()
m2 = Entry(abl1)
m2.pack()

sz3 = Label(abl1, text ='Vízszintes végpozíció:')
```

```
sz3.pack()
m3 = Entry(abl1)
m3.pack()
sz4 = Label(abl1, text ='Függőleges végpozíció:')
sz4.pack()
m4 = Entry(abl1)
m4.pack()

sz5 = Label(abl1, text ='Vonalvastagság:')
sz5.pack()
m5 = Entry(abl1)
m5.pack()

gomb1.pack(side=BOTTOM)
gomb2 = Button(abl1,text='Vonalat rajzol',command=vonalatrajzol)
gomb2.pack()

abl1.mainloop()
```

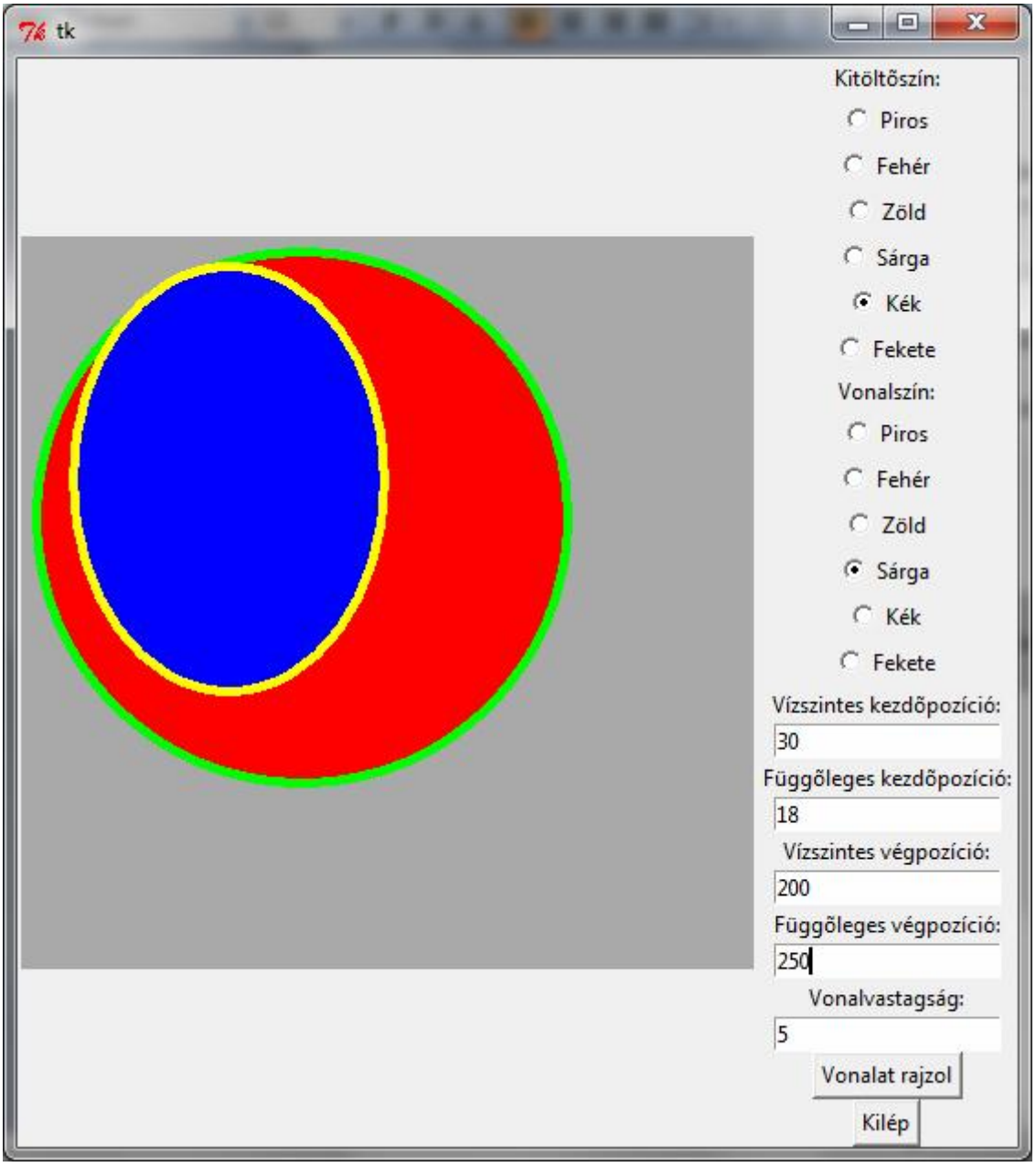
Ellipszis (és kör) rajzolása

canvas.*create_oval*(*x1*,*y1*,*x2*,*y2*, **width**=*vast*, **fill**=*color*, **outline**=*color2*)

- *x1* : az ellipszist „magába foglaló” téglalap egyik csúcsának a festővászon bal szélétől mért távolsága
- *y1* : az ellipszist „magába foglaló” téglalap egyik csúcsának a festővászon tetejétől mért távolsága
- *x2* : az ellipszist „magába foglaló” téglalap átellenes csúcsának a festővászon bal szélétől mért távolsága
- *y2* : az ellipszist „magába foglaló” téglalap átellenes csúcsának a festővászon tetejétől mért távolsága
- *vastagság* : a határvonal vastagsága
- *color* : a kitöltés színe
- *color2* : a határvonal színe

Feladat – Ellipszisek rajzolása

Írjunk olyan programot, amely ellipsziseket rajzol, az ellipszisek adatai futás közben legyenek beállíthatók!




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

def vonalatrajzol():
    x1 = eval(m1.get())
    y1 = eval(m2.get())
    x2 = eval(m3.get())
    y2 = eval(m4.get())
    vast = eval(m5.get())
    color = col.get()
    color2 = col2.get()
    can1.create_oval(x1,y1,x2,y2, width=vast, fill=color, outline=color2)

abl1 = Tk()
can1 = Canvas(abl1,bg='dark grey',height=400,width=400)
```

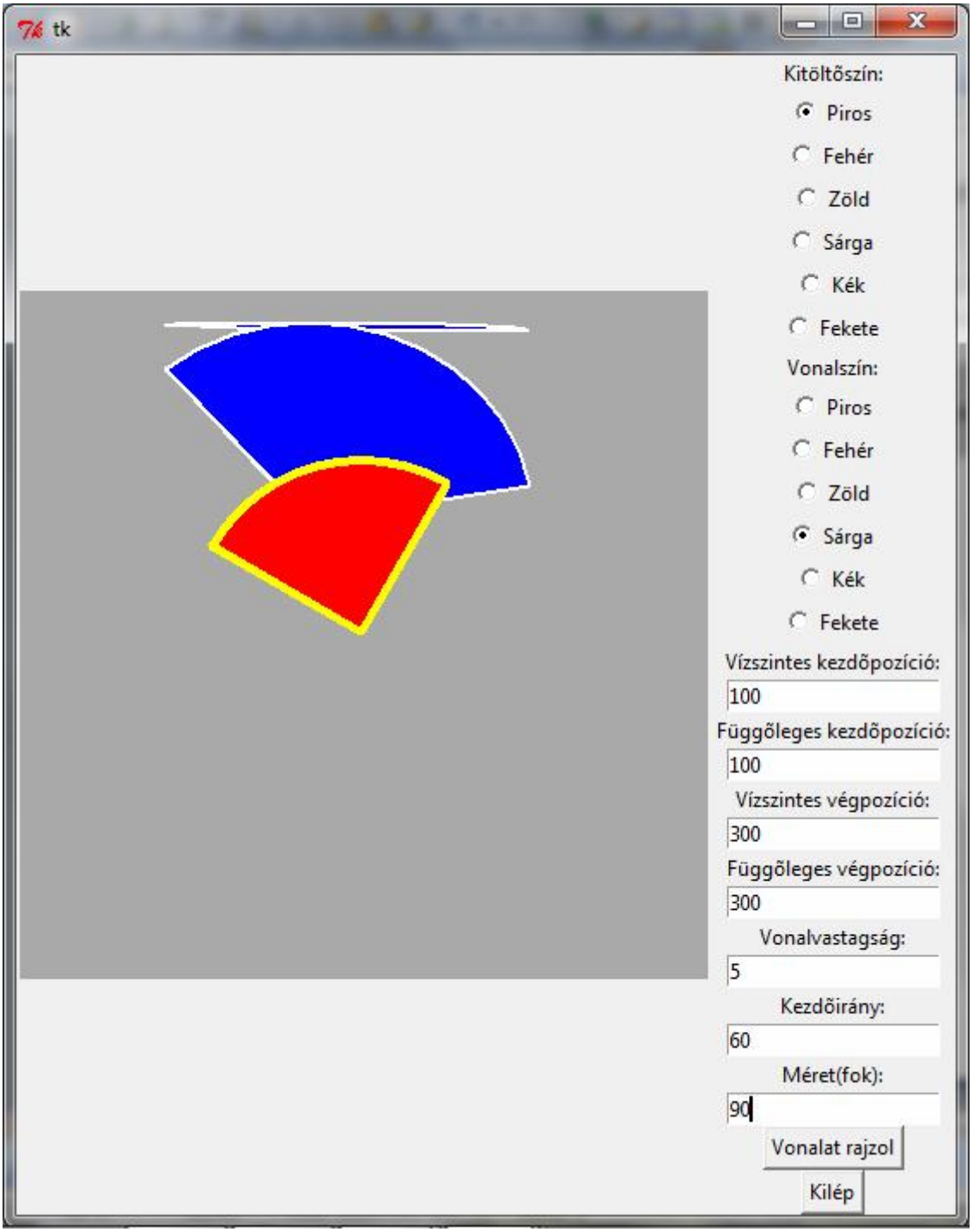
```
can1.pack(side=LEFT)

gomb1 = Button(abl1,text='Kilép',command=abl1.destroy)

szov1 = Label(abl1, text ='Kitöltőszín:')
szov1.pack()
col = StringVar()
radio1 = Radiobutton(abl1, text='Piros', value='red', variable = col)
radio2 = Radiobutton(abl1, text='Fehér', value='white', variable = col)
radio3 = Radiobutton(abl1, text='Zöld', value='green', variable = col)
radio4 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col)
radio5 = Radiobutton(abl1, text='Kék', value='blue', variable = col)
radio6 = Radiobutton(abl1, text='Fekete', value='black', variable = col)
radio1.pack()
radio2.pack()
radio3.pack()
radio4.pack()
radio5.pack()
radio6.pack()

szov2 = Label(abl1, text ='Vonalszín:')
szov2.pack()
col2 = StringVar()
radio7 = Radiobutton(abl1, text='Piros', value='red', variable = col2)
radio8 = Radiobutton(abl1, text='Fehér', value='white', variable = col2)
radio9 = Radiobutton(abl1, text='Zöld', value='green', variable = col2)
radio10 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col2)
radio11 = Radiobutton(abl1, text='Kék', value='blue', variable = col2)
radio12 = Radiobutton(abl1, text='Fekete', value='black', variable = col2)
radio7.pack()
radio8.pack()
radio9.pack()
radio10.pack()
radio11.pack()
radio12.pack()

sz1 = Label(abl1, text ='Vízszintes kezdőpozíció:')
sz1.pack()
m1 = Entry(abl1)
m1.pack()
sz2 = Label(abl1, text ='Függőleges kezdőpozíció:')
sz2.pack()
m2 = Entry(abl1)
m2.pack()

sz3 = Label(abl1, text ='Vízszintes végpozíció:')
sz3.pack()
m3 = Entry(abl1)
m3.pack()
sz4 = Label(abl1, text ='Függőleges végpozíció:')
```

```
sz4.pack()
m4 = Entry(abl1)
m4.pack()

sz5 = Label(abl1, text ='Vonalvastagság:')
sz5.pack()
m5 = Entry(abl1)
m5.pack()

gomb1.pack(side=BOTTOM)
gomb2 = Button(abl1,text='Vonalat rajzol',command=vonalatrajzol)
gomb2.pack()

abl1.mainloop()
```

Ív rajzolása

canvas.*create_arc*(*x1*,*y1*,*x2*,*y2*, **width**=*vast*, **fill**=*color*, **outline**=*color2*, **start=***kezdoirany*, **extent=***fok*)

- *x1* : az ellipszist „magába foglaló” téglalap egyik csúcsának a festővászon bal szélétől mért távolsága
- *y1* : az ellipszist „magába foglaló” téglalap egyik csúcsának a festővászon tetejétől mért távolsága
- *x2* : az ellipszist „magába foglaló” téglalap átellenes csúcsának a festővászon bal szélétől mért távolsága
- *y2* : az ellipszist „magába foglaló” téglalap átellenes csúcsának a festővászon tetejétől mért távolsága
- *vastagság* : a határvonal vastagsága
- *color* : a kitöltés színe
- *color2* : a határvonal színe
- *kezdoirany* : az ív kezdőpontjának irányszöge
- *fok* : az ív középponti szöge az óramutató járásával ellentétes irányban.



Feladat – Ívek rajzolása
Írjunk olyan programot, amely íveket rajzol, az ívek adatai futás közben legyenek beállíthatók!

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

def vonalatrajzol():
    x1 = eval(m1.get())
    y1 = eval(m2.get())
    x2 = eval(m3.get())
```

```
    y2 = eval(m4.get())
    vast = eval(m5.get())
    kezdo = eval(m6.get())
    meret = eval(m7.get())
    color = col.get()
    color2 = col2.get()
    can1.create_arc(x1,y1,x2,y2, width=vast, fill=color, outline=color2, start=kezdo,
extent=meret)

abl1 = Tk()
can1 = Canvas(abl1,bg='dark grey',height=400,width=400)
can1.pack(side=LEFT)

gomb1 = Button(abl1,text='Kilép',command=abl1.destroy)

szov1 = Label(abl1, text ='Kitöltőszín:')
szov1.pack()
col = StringVar()
radio1 = Radiobutton(abl1, text='Piros', value='red', variable = col)
radio2 = Radiobutton(abl1, text='Fehér', value='white', variable = col)
radio3 = Radiobutton(abl1, text='Zöld', value='green', variable = col)
radio4 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col)
radio5 = Radiobutton(abl1, text='Kék', value='blue', variable = col)
radio6 = Radiobutton(abl1, text='Fekete', value='black', variable = col)
radio1.pack()
radio2.pack()
radio3.pack()
radio4.pack()
radio5.pack()
radio6.pack()

szov2 = Label(abl1, text ='Vonalszín:')
szov2.pack()
col2 = StringVar()
radio7 = Radiobutton(abl1, text='Piros', value='red', variable = col2)
radio8 = Radiobutton(abl1, text='Fehér', value='white', variable = col2)
radio9 = Radiobutton(abl1, text='Zöld', value='green', variable = col2)
radio10 = Radiobutton(abl1, text='Sárga', value='yellow', variable = col2)
radio11 = Radiobutton(abl1, text='Kék', value='blue', variable = col2)
radio12 = Radiobutton(abl1, text='Fekete', value='black', variable = col2)
radio7.pack()
radio8.pack()
radio9.pack()
radio10.pack()
radio11.pack()
radio12.pack()

sz1 = Label(abl1, text ='Vízszintes kezdőpozíció:')
sz1.pack()
m1 = Entry(abl1)
```

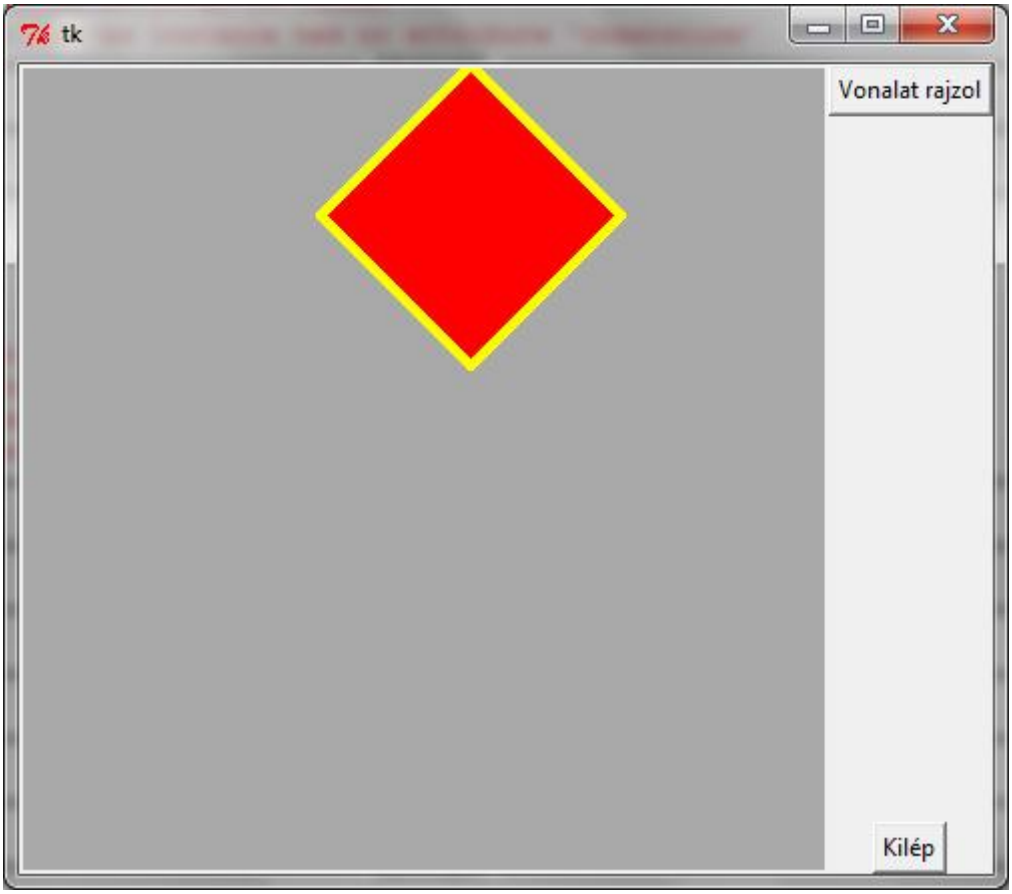
```
m1.pack()
sz2 = Label(abl1, text ='Függőleges kezdőpozíció:')
sz2.pack()
m2 = Entry(abl1)
m2.pack()

sz3 = Label(abl1, text ='Vízszintes végpozíció:')
sz3.pack()
m3 = Entry(abl1)
m3.pack()
sz4 = Label(abl1, text ='Függőleges végpozíció:')
sz4.pack()
m4 = Entry(abl1)
m4.pack()

sz5 = Label(abl1, text ='Vonalvastagság:')
sz5.pack()
m5 = Entry(abl1)
m5.pack()

sz6 = Label(abl1, text ='Kezdőirány:')
sz6.pack()
m6 = Entry(abl1)
m6.pack()
sz7 = Label(abl1, text ='Méret(fok):')
sz7.pack()
m7 = Entry(abl1)
m7.pack()

gomb1.pack(side=BOTTOM)
gomb2 = Button(abl1,text='Vonalat rajzol',command=vonalatrajzol)
gomb2.pack()

abl1.mainloop()
```

Polygon (sokszög) rajzolása

canvas.***create_polygon***(*x1*,*y1*,*x2*,*y2*, …, *xn*, *yn*, **width**=*vast*, **fill**=*color*, **outline**=*color2*)

- *x1*,*y1*,*x2*,*y2*, …, *xn*, *yn* : a sokszög csúcspontjainak koordinátái.
- *vastagság* : a határvonal vastagsága
- *color* : a kitöltés színe
- *color2* : a határvonal színe

Feladat – Sokszög rajzolása
Írjunk olyan programot, amely elkészíti a következő ábrát!




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

def vonalatrajzol():
    can1.create_polygon(150,75,225,0,300,75,225,150, width=5, fill='red', outline='yellow')

abl1 = Tk()
can1 = Canvas(abl1,bg='dark grey',height=400,width=400)
can1.pack(side=LEFT)

gomb1 = Button(abl1,text='Kilép',command=abl1.destroy)

gomb1.pack(side=BOTTOM)
gomb2 = Button(abl1,text='Vonalat rajzol',command=vonalatrajzol)
gomb2.pack()

abl1.mainloop()
```

FELADATOK

1. Írj olyan programot, amely egy gomb lenyomására kirajzolja az öt olimpiai karikát egy fehér hátterű vászonra! A programból egy másik nyomógomb segítségével lehessen kilépni!

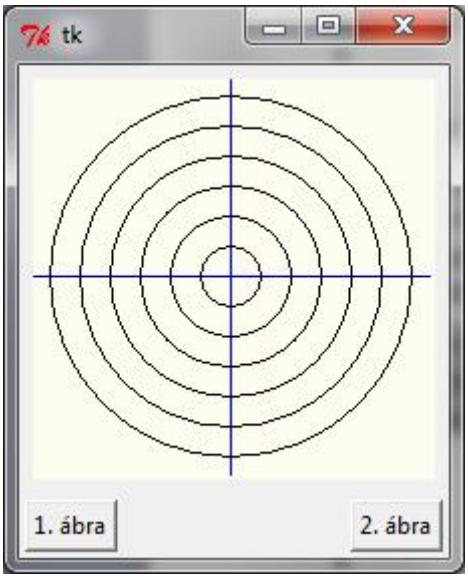
2. Alakítsd át az előző programot úgy, hogy a karikákat külön-külön gombhoz tartozó esemény rajzolja meg!

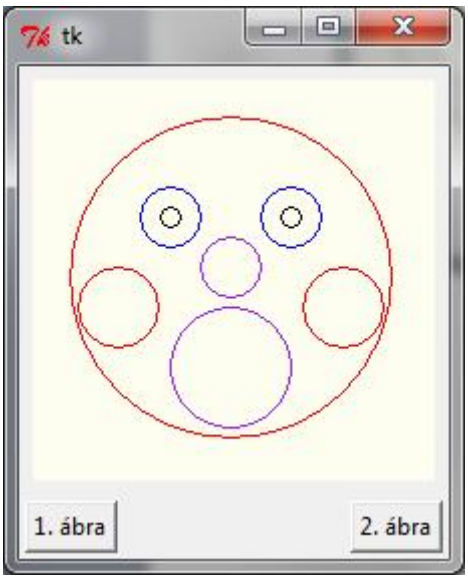
3. Írj olyan programot, amely két téglalapból egy piros keresztet állít elő gombnyomás hatására! A programból egy másik nyomógomb segítségével lehessen kilépni!

Rajzok közötti váltás

Egy példa segítségével azt mutatjuk be, hogyan lehet felhasználni a programhurkokról, listákról és függvényekről szerezett ismereteinket arra, hogy néhány kódsor segítségével több rajzot hozzunk létre. Az alkalmazásunk a választott gombtól függően fogja megjeleníteni az alábbi rajzok közül az egyiket.

Feladat – Rajzok közötti váltás
Írjunk olyan programot, amely két nyomógombot tartalmaz, az egyikre kattintva egy céltábla, a másikra kattintva egy arc jelenik meg!






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

```
from Tkinter import *

def kor(x, y, r, color ='black'):
    can.create_oval(x-r, y-r, x+r, y+r, outline=color)

def celtabla():
    # először a meglévő rajz törlése :
    can.delete(ALL)
    # a két egyenes rajzolása (függ. és vizsz.) :
    can.create_line(100, 0, 100, 200, fill ='blue')
    can.create_line(0, 100, 200, 100, fill ='blue')
    # több koncentrikus kör rajzolása :
    sugar = 15
    while sugar < 100:
        kor(100, 100, sugar)
        sugar += 15

def arc():
    "egyszerűsített arc rajzolása"
    # először minden meglévő rajz törlése :
    can.delete(ALL)
    # minden kör jellemzőjét listák listájába tesszük
    cc =[[100, 100, 80, 'red'],     # fej
        [70, 70, 15, 'blue'],      # szem
        [130, 70, 15, 'blue'],
        [70, 70, 5, 'black'],
        [130, 70, 5, 'black'],
        [44, 115, 20, 'red'],      # arc
        [156, 115, 20, 'red'],
        [100, 95, 15, 'purple'],   # orr
        [100, 145, 30, 'purple']]  # száj
    # az összes kört egy ciklus segitségével rajzoljuk meg :
    i =0
    while i < len(cc):      # a lista bejárása
        elem = cc[i]        # minden elem maga is lista
        kor(elem[0], elem[1], elem[2], elem[3])
        i += 1

#Főprogram
window = Tk()
can = Canvas(window, width =200, height =200, bg ='ivory')
can.pack(side =TOP, padx =5, pady =5)
b1 = Button(window, text ='1. ábra', command =celtabla)
b1.pack(side =LEFT, padx =3, pady =3)
b2 = Button(window, text ='2. ábra', command =arc)
b2.pack(side =RIGHT, padx =3, pady =3)
window.mainloop()
```

Kezdjük az alkalmazás végén lévő főprogram elemzésével : Létrehozunk a ***Tk***() osztályból egy ablak-objektumot a *window* változóban.

Utána létrehozunk ebben az ablakban 3 widgetet : egy vásznat (***canvas***) a *can* és két gombot (***button***) a *b1* és *b2* változókba. A widget-eket - mint az előző scriptben - a ***pack***() metódusukkal pozícionáljuk az ablakban, de most a metódust opciókkal használjuk :

- a **side** opció a TOP, BOTTOM, LEFT és RIGHT értékeket fogadja el. A widget-et a megfelelő oldalra teszi az ablakban.
- a **padx** és **pady** opciók egy sávot tartanak szabadon a widget körül. Ezt a sávot pixelekben adjuk meg : a **padx** a widget bal- és jobboldalán, a **pady** a widget alatt és fölött foglalja le az említett sávot.

A gombok vezérlik a két ábra kirajzolását a ***celtabla***() és az ***arc***() függvények hívásával. Mivel több kört kell rajzolni a vászonra, ezért úgy gondoltuk, hasznos lenne először egy specializált ***kor***() függvényt írni. Mint már tudjuk, a ***Tkinter canvas***-nak van egy ***create_oval***() metódusa, amivel akármilyen ellipszist (így köröket is) rajzolhatunk. Azonban ezt a metódust négy argumentummal kell hívni. Az argumentumok egy fiktív téglalap balfelső és jobbalsó sarkának a koordinátáit adják meg, amibe bele fogjuk rajzolni az ellipszist. A kör speciális esetében ez nem túl praktikus. Természetesebbnek tűnik a rajzolást a kör középpontjának és sugarának megadásával vezérelni. Ezt a ***kor***() függvényünkkel érjük el, ami a koordináták konverzióját elvégezve hívja a ***create_oval***() metódust. Vegyük észre, hogy a függvény egy opcionális argumentumot vár, ami a rajzolandó kör színére vonatkozik (ez alapértelmezetten fekete).

A ***celtabla***() függvényben világosan szembetűnik ennek a megközelítésnek a hatékonysága. Egy programhurok van benne, ami az azonos középpontú, növekvő sugarú körök sorozatának kirajzolására szolgál. Figyeljük meg a += incrementáló operátor használatát (példánkban az r változó értékét növeli 15 egységgel minden iterációban).

A második rajz egy kicsit összetettebb, mert változó méretű, különböző középpontú körökből van összeállítva. Az összes kört ugyanígy, egyetlen programhurok segítségével rajzolhatjuk meg, ha felhasználjuk a listákról szerzett tudásunkat.

Négy jellemző különbözteti meg egymástól a megrajzolandó köröket :a középpont x és y koordinátái, a sugár és a szín. Mindegyik körnek ezt a négy jellemzőjét összegyűjthetjük egy-egy listában és ezeket a listákat egy másik listában tárolhatjuk. Így egy egymásba ágyazott listánk lesz, amit elég lesz egy ciklus segítségével bejárnunk a megfelelő rajzok elkészítéséhez.

FELADATOK

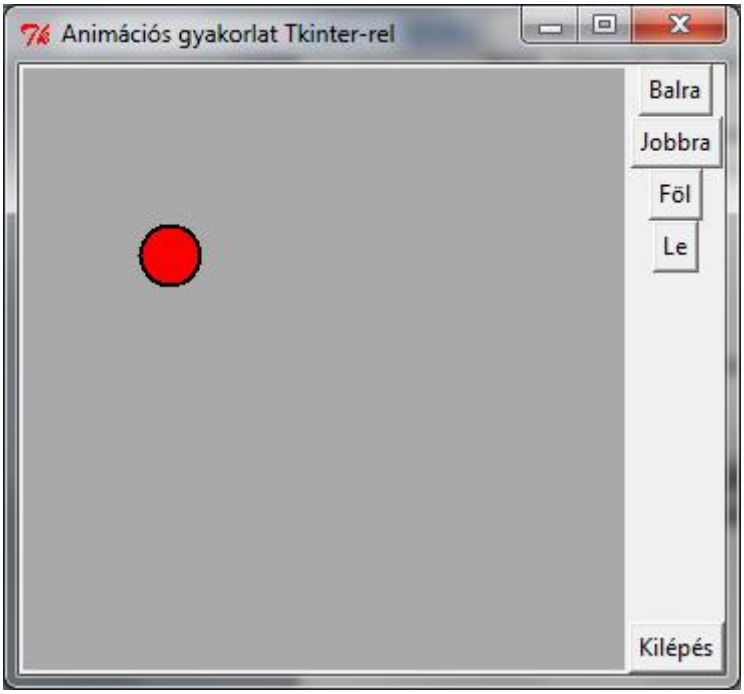
1. Írjunk egy programot, ami egy sakktáblát (fehér alapon fekete négyzeteket) jelenít meg, amikor egy gombra kattintunk!

2. Az előző programa építsünk be még egy gombot, ami korongokat jelenít meg véletlenszerűen a sakktáblán (a gomb minden megnyomására egy új korong jelenjen meg)!

Animáció

A végéhez közeledve ebben a részben olyan programot fogunk készíteni, amely egy rajzot mozgat a vásznon.

Feladat – Kör mozgatása
Írjunk olyan programot, amely nyomógombok segítségével egy kört mozgat a vásznon!




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

# általános mozgatóeljárás :
def mozog(gd, hb):
    global x1, y1
    x1, y1 = x1 + gd, y1 + hb
    can1.coords(oval1, x1, y1, x1+30, y1+30)

# eseménykezelők :
def mozdit_balra():
    mozog(-10, 0)

def mozdit_jobbra():
    mozog(10, 0)

def mozdit_fel():
    mozog(0, -10)

def mozdit_le():
    mozog(0, 10)

#------ Főprogram -------

# a következő változók globálisak :
x1, y1 = 10, 10                 # kiindulási koordináták

# A fő ("master") widget létrehozása :
abl1 = Tk()
abl1.title("Animációs gyakorlat Tkinter-rel")
```

```
# "slave" widget-ek létrehozása :
can1 = Canvas(abl1,bg='dark grey',height=300,width=300)
oval1 = can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill='red')
can1.pack(side=LEFT)
Button(abl1,text='Kilépés',command=abl1.destroy).pack(side=BOTTOM)
Button(abl1,text='Balra',command=mozdit_balra).pack()
Button(abl1,text='Jobbra',command=mozdit_jobbra).pack()
Button(abl1,text='Föl',command=mozdit_fel).pack()
Button(abl1,text='Le',command=mozdit_le).pack()

# eseményfigyelő (főhurok) indítása :
abl1.mainloop()
```

A program számos ismert elemet tartalmaz: létrehozunk egy *abl1* ablakot, ebben elhelyezünk egy vásznat, amin egy színes kör, és öt vezérlőgomb van. Vegyük észre, hogy a nyomógomb widget-eket nem rendeljük változókhoz (ez felesleges lenne, mert nem fogunk rájuk hivatkozni a későbbiekben). A ***pack***() metódust rögtön ezeknek az objektumoknak a létrehozásakor kell alkalmaznunk.

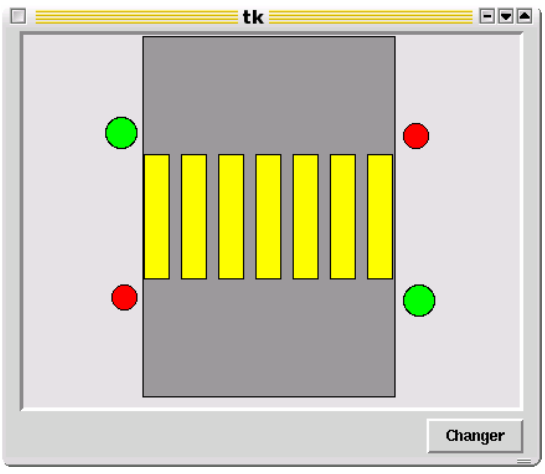
Az igazi újdonság a script elején definiált ***mozog***() függvényben van. A függvény minden egyes hívásakor át fogja definiálni a vászonra helyezett színes kör objektum koordinátáit és ez idézi elő az objektum animációját.

Ez az eljárás jellemző az objektumorientált programozásra: Létrehozunk egy objektumot, majd metódusok segítségével módosítjuk a tulajdonságaikat.

A régimódi procedurális (azaz az objektumok használatát nélkülöző) programozásban úgy animáljuk az ábrákat, hogy töröljük őket egy adott helyen, majd egy kicsit távolabb újra rajzoljuk őket. Ezzel szemben az objektumorientált programozásban ezeket a feladatokat azok az osztályok végzik el automatikusan, melyekből az objektumok származnak és ezért nem kell az időnket vesztegetni ezek újraprogramozására.

FELADAT

1. Írjunk olyan programot, ami megjelenít egy ablakot egy vászonnal és egy gombbal. A vásznon rajzoljunk egy sötétszürke téglalapot, ami egy utat jelöl, fölé pedig sárga téglalapokat, amik egy gyalogátkelőhelyet reprezentálnak. Helyezzünk el négy színes kört, amik a gyalogosok és a járművek közlekedési lámpáit reprezentálják. A nyomógombra történő minden egyes kattintásra a lámpáknak színt kell váltani!



Automatikus animáció - Rekurzivitás

A ***Tkinter*** grafikus interface-szel való első találkozásunk összegzéseként nézzük meg az utolsó animációs példát, ami ez alkalommal az elindítása után autonóm módon fog működni.

Feladat – Kör mozgatása 2
Írjunk olyan programot, amely egy kört mozgat a vásznon automatikusan!


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

# eseménykezelők definiálása

def mozog():
    global x1, y1, dx, dy, jel
    x1, y1 = x1 +dx, y1 + dy
    if x1 >360:
        x1, dx, dy = 360, 0, 15
    if y1 >360:
        y1, dx, dy = 360, -15, 0
    if x1 <10:
        x1, dx, dy = 10, 0, -15
    if y1 <10:
        y1, dx, dy = 10, 15, 0
    can1.coords(oval1,x1,y1,x1+30,y1+30)
    if jel >0:
```

```
    abl1.after(50, mozog)          # 50 millisec után ciklus

def leallit():
   global jel
   jel =0

def elindit():
   global jel
   if jel ==0:      # azért, hogy csak egy ciklust indítsunk
     jel = 1
     mozog()

#========== Főprogram ============
# a következő változókat globális változókként fogjuk használni :
x1, y1 = 10, 10                # kezdő koordináták
dx, dy = 15, 0          # elmozdulás
jel =0                         # kapcsoló

# A fő-widget létrehozása ("master") :
abl1 = Tk()
abl1.title("Animációs gyakorlat Tkinter-rel")

# a "slave" widget-ek (canvas + oval, button) létrehozása:
can1 = Canvas(abl1,bg='dark grey',height=400,width=400)
can1.pack(side=LEFT)
oval1 = can1.create_oval(x1,y1,x1+30,y1+30,width=2,fill='red')
Button(abl1,text='Kilép',command=abl1.destroy).pack(side=BOTTOM)
Button(abl1,text='Elindít',command=elindit).pack()
Button(abl1,text='Leállít',command=leallit).pack()

# az eseményfogadó indítása (főciklus) :
abl1.mainloop()
```

A programban az egyetlen újdonság a ***mozog***() függvény definíciójának végén található ***after***() metódus használata. Ezt a metódust bármilyen ablakra alkalmazhatjuk. Egy függvényhívást fog kezdeményezni egy bizonyos idő eltelte után. Így például a *window*.***after***(200,***qqc***) a window widget-en 200 millimásodperc szünet után hívja a ***qqc***() függvényt.

Programunkban az ***after***() metódus önmagát a ***mozog***() függvényt hívja. Most először használjuk a rekurziónak nevezett igen hatékony programozási technikát. Az egyszerűség kedvéért azt fogjuk mondani, hogy akkor történik rekurzió, amikor egy függvény önmagát hívja. Nyilván így egy végtelen ciklust kapunk, hacsak nem gondoskodunk előre valamilyen eszközről, hogy megszakítsuk azt.

Nézzük, hogyan működik ez a példánkban:

A ***mozog***() függvényt először az Elindít gombra való kattintáskor hívja a program. A függvény elvégzi a feladatát (vagyis pozícionálja a labdát) majd egy kis szünet után önmagát hívja. Elvégzi feladatát, majd egy kis szünet után újra hívja önmagát és ez így megy tovább a végtelenségig.

Legalábbis ez történne, ha elővigyázatosságból nem tettünk volna valahová a hurokba egy kilépési utasítást. Egy egyszerű feltételvizsgálatról van szó: minden iterrációs ciklusban

megvizsgáljuk egy **if** utasítás segítségével a *jel* változó tartalmát. Ha a *jel* változó tartalma nulla, akkor a ciklus többet nem fut le, és az animáció leáll. Mivel a *jel* változó egy globális változó, ezért más függvények segítségével könnyen megváltoztathatjuk az értékét. Azokkal a függvényekkel, amiket az Elindít és a Leállít gombokhoz kapcsoltunk.

Így egy egyszerű mechanizmust kapunk az animáció indítására és leállítására :

Az első kattintás az Elindít gombra egy nem nulla értéket rendel a *jel* változóhoz, majd rögtön kiváltja a ***mozog***() függvény első hívását. Ez végrehajtódik, és ezután mindaddig hívja önmagát, amíg a *jel* nullává nem válik. Ha tovább kattintgatunk az Elindít gombra, a *jel* értéke nő, de semmi egyéb nem történik, mert a ***mozog***() hívására csak akkor kerül sor, ha a *jel* értke 1. Így elkerüljük több, konkurrens ciklus indítását.

A Leállít gomb visszaállítja a *jel* értékét nullára és a ciklus megszakad.

FELADATOK

1. Módosítsuk az előző programot úgy, hogy a kör minden fordulónál változtassa meg a színét!

2. Módosítsuk az előző programot úgy, hogy a kör ferde mozgást végezzen, és úgy változtasson irányt, mint az asztal széléről visszapattanó biliárdgolyó!

VÉGE A MÁSODIK RÉSZNEK

1. Grafikus interface-ek (GUI) 3
2. Első lépések a Tkinter-rel 3
3. Eseményvezérelt programok 6
4. A Tkinter widget-osztályai 8
5. A grid() metódus alkalmazása widget-ek pozícionálására 9
6. Widget-ek 34
A jelölőnégyzet 34
A rádiógomb 35
Listadoboz 36
Szövegdoboz 38
Görgetősáv 38
Scale 40
Message 41
7. Többablakos alkalmazás (Toplevel) 42
8. Menükészítés 44
9. Grafika a programban, Canvas 54
Vonal rajzolása 54
Téglalap rajzolása 56
Ellipszis (és kör) rajzolása 59
Ív rajzolása 62
Polygon (sokszög) rajzolása 65
Rajzok közötti váltás 67
Animáció 69
Automatikus animáció - Rekurzivitás 72