ЗАКАРПАТСЬКИЙ УГОРСЬКИЙ ІНСТИТУТ ІМЕНІ ФЕРЕНЦА РАКОЦІ ІІ
II. RÁKÓCZI FERENC KÁRPÁTALJAI MAGYAR FŐISKOLA

Кафедра математики та інформатики
Matematika és Informatika Tanszék

МОДУЛЬ TKINTER БІБЛОТЕКИ PYTHON/ PYTHON KÖNYTÁRI TKINTER MODUL

МЕТОДИЧНІ ВКАЗІВКИ ДО ПРАКТИЧНИХ ЗАНЯТЬ З ДИСЦІПЛІНИ «СУЧАСНІ МОВИ ПРОГРАМУВАННЯ»/ MÓDSZERTANI ÚTMUTATÓ "MODERN PROGRAMOZÁSI NYELVEK" TÁRGY GYAKORLATI FOGLALKOZÁSAIHOZ

Перший (бакалаврський) / Alapképzés (BSc)
(рівень вищої освіти / felsőoktatás szintje)
01 Освіта/Педагогіка / 01 Oktatás/Pedagógia
(галузь знань / képzési ág)
Середня освіта (Інформатика) / Középiskolai oktatás (Informatika)
(освітня програма / képzési program)

Берегове / Beregszász 2024 р.

Методичні вказівки «Модуль Tkinter бібліотеки Python» до практичних занять з дисципліни «Сучасні мови програмування» розроблені на основі Освітньої програми підготовки бакааврів з галузі знань «01 Освіта/Педагогіка» за напрямом «014 Середня освіта (Інформатика)» як для студентів денної, так і заочної форми навчання. Метою методичного посібника є поглиблення знань студентів з дисципліни « Сучасні мови програмування». У роботі описано модуль Tkinter мови програмування Python, наводяться приклади програм з використанням модуля Tkinter. Методичні вказівки можуть бути використані на практичних заняттях та при самостійній роботі по даному курсу.

Затверджено до використання у навчальному процесі
на засіданні кафедри математики та інформатики ЗУІ ім. Ф.Ракоці ІІ
(протокол № 1 від «13» серпня 2024 року)

Розглянуто та рекомендовано Навчально-методичною радою
Закарпатського угорського інституту імені Ференца Ракоці ІІ
(протокол №22 від «26» серпня 2024 року)

Рекомендовано до видання в електронній формі (PDF)
рішенням Вченої ради Закарпатського угорського інституту імені Ференца
Ракоці ІІ (протокол №7 від « 27» серпня 2024 року)

Підготовлено до видання в електронній формі (PDF) кафедрою математики та інформатики спільно з Видавничим відділом Закарпатського угорського інституту імені Ференца Ракоці ІІ

Розробник методичних вказівок:

Йожеф ГОЛОВАЧ – професор кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доктор технічних наук

Рецензенти:

Олександр МІЦА – завідувач кафедри інформаційних управляючих систем та технологій УжНУ, доктор технііч наук, професор (м. Ужгород)

Мирослав СТОЙКА – доцент кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, кандидат фізико-математичних наук, доцент

Відповідальні за випуск:

Каталін КУЧІНКА – завідувач кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доцент, кандидат фізико-математичних наук

Олександр ДОБОШ – начальник Видавничого відділу ЗУІ ім. Ф.Ракоці ІІ

За зміст методичних вказівок відповідальність несе розробник.

Видавництво: Закарпатський угорський інститут імені Ференца Ракоці ІІ (адреса: пл. Кошута 6, м. Берегове, 90202. Електронна пошта: foiskola@kmf.uz.ua)

A „Python könyvtári Tkinter modul” módszertani útmutató a „Modern programozási nyelvek” tárgy gyakorlati foglalkozásaihoz a BSc „Középiskolai oktatás (Informatika)” képzési programja alapján lett kidolgozva nappali és levelező tagozatos hallgatók részére. A módszertani kézikönyv célja, hogy elmélyítse a „Modern programozási nyelvek” tárgy t hallgatóinak ismereteit. Az útmutató leírja a Python programozási nyelv Tkinter modulját, példákat ad a Tkinter alkalamzására a programok keszitésében . A módszertani utasítások a gyakorlati órákon és a tantárgy önálló munkája során használhatók.

Javasolta a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola
Matematika és Informatika Tanszéke
(2024. 08.13, 1. számú jegyzőkönyv).

Megjelentetésre javasolta a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola
Oktatási és Módszertani Tanácsa
(2024. augusztus 26., 22. számú jegyzőkönyv).

Elektronikus formában (PDF fájlformátumban) történő kiadásra javasolta
a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Tudományos Tanácsa
(2024. augusztus 27., 7 . számú jegyzőkönyv).

A módszertani útmutató kidolgozója:

HOLOVÁCS József – professzor, műszaki tudományok doktora, a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszékének professzora

Szakmai lektorok:

MITSA Oleksandr – Az UNE Információs vezérlési Rendszerek és Technológiák Tanszékének vezetője, műszaki tudományok doktora, professzor

SZTOJKA Miroszláv – a fizikai és matematikai tudományok kandidátusa, docens, a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszékének docense

A kiadásért felelnek:

KUCSINKA Katalin – PhD, a fizikai és matematikai tudományok kandidátusa, a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Matematika és Informatika Tanszékének tanszékvezetője és docense

DOBOS Sándor – a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola Kiadói Részlegének vezetője

A segédlet tartalmáért kizárólag a módszertani útmutató kidolgozója felel.

Kiadó: II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola (cím: 90 202, Beregszász, Kossuth tér 6. E-mail: foiskola@kmf.uz.ua)

TARTALOMJEGYZÉK

ELŐSZÓ

A Python egy ingyenes, magas szintű, objektumorientált, általános célú programozási nyelv automatikus memóriakezeléssel, valamint a Pythonban írt kód hordozhatóságának biztosítása. Miután a hallgatók megismerkrdnek a Python alalpjaival és képesek létrehozni saját függvényeket, kezelni a fontosabb adatszerkezeteket, ajánlatos megismerkedniük a Tkinter könytárral és alkalmazni azt a feladatok megoldására.

A Tkinter egy Python szabványos könyvtári modul, amely interfészt biztosít grafikus felhasználói felületek (GUI) létrehozásához. A Tk könyvtáron alapul, amely több eszközt biztosít ablakok, widgetek és események létrehozásához és kezeléséhez egy grafikus alkalmazásban.

A Tkinter főbb jellemzői:

Könnyű használat: A Tkinter egyszerű és egyértelmű módot biztosít felhasználói felület létrehozására Pythonban. API-ja intuitív és könnyen megtanulható.

Cross-Platform: A Tkinterrel épített alkalmazások jelentősebb módosítások nélkül futhatnak különböző operációs rendszereken, például Windowson, macOS-en és Linuxon.

Beépített widgetek: A Tkinter szabványos widgetek széles skáláját kínálja, mint például gombok, szövegmezők, címkék, listák és egyebek, amelyek segítségével különféle interfész elemeket lehet létrehozni.

Eseménymodell: A Tkinter támogatja az olyan események kezelését, mint az egérkattintások, billentyűleütések és egyéb felhasználói műveletek, lehetővé téve interaktív alkalmazások létrehozását.

Rugalmasság és bővíthetőség: A szabványos widgetek mellett a Tkinter lehetővé teszi egyéni widgetek és elrendezések létrehozását és testreszabását, valamint integrálását a meglévő alkalmazásokba.

A Tkinter által gyorsan létrehozhatók egyszerű grafikus felületű alkalmazások, például adminisztrációs eszközök, adatkezelő programok, vizualizációs eszközök stb.

E módszertani útmutató tartalmaz több megoldot feladatott a Tkinter alkalmzásával. A módszertani útmutató célja, hogy a hallgatók miután tanulmányozták, elemezték a feladatok megoldására készitett programokat, képesek legyenek önállóan készitени bonyolultabb pr

ogramokat, amelyek alkalmazzák a grafikus felhasználói felületekek.

1. Bevezetés. GUI alkalmazások

A programozók által írt programok között nagyon fontosak a grafikus felhasználói felülettel (GUI, graphical user interface) rendelkező alkalmazások. Az ilyen programok készítésekor nem csak az adatfeldolgozó algoritmusok válnak fontossá, hanem a felhasználó számára programfejlesztés is egy kényelmes interfész, amellyel kölcsönhatásba lépve meghatározza az alkalmazás viselkedését.

A grafikus felhasználói felülettel rendelkező alkalmazásoknak nemcsak szépnek kell lenniük a képernyőn, hanem bizonyos műveleteket is végre kell hajtaniuk, ezáltal megvalósítva a felhasználó igényeit. A modern felhasználó elsősorban különféle gombok, menük, ikonok segítségével lép kapcsolatba a programmal, és információkat ír be speciális mezők, bizonyos értékek kiválasztásával listákból, stb. Ezek az eszközök, amelyeket **widgeteknek** hívják, bizonyos értelemben a GUI-t alkotják.

A Python esetében ez eszköz egy speciális könyvtárban található: a **tkinter-ben**. Ha importálja a programba(scriptbe), akkor grafikus felületet lehet létrehozni. A program felhasználóbarát legyen, és reagáljon a műveleteire (eseményekre).

A grafikus alkalmazás létrehozásakor a lépések sorrendjének megvannak a maga sajátosságai.

Javasolt lépések, amelyeken keresztül kell mennie a programozás során, hogy egy grafikus felhasználói felülettel rendelkező programot kapjon:

1. Könyvtár importálása
2. A főablak létrehozása
3. Hozzon létre egy widgetet
4. Tulajdonságuk beállítása
5. Az események meghatározása
6. Eseménykezelők meghatározása
7. A widget elhelyezése a főablakban
8. Jelenítse meg a főablakot

1. tkinter modul importálása

```
import tkinter
```

2. Fő ablak létrehozása

A modern operációs rendszerekben minden felhasználói alkalmazás egy ablakba van zárva, amelyet főnek nevezhetünk, mert benne az összes többi widget megtalálható.

A tkinter modul **Tk** osztályának meghívásakor egy legfelső szintű ablak objektum jön létre. Az ablak objektumhoz társított elemet általában root-nak hívják (bár egyértelmű, hogy bárminek nevezhető). Második kódsor:

```
root = Tk()
```

Példa. Üres ablak létrehozása

```
from tkinter import *
root = Tk() # ablak létrehozása
root.title("Üres ablak") # ablak fejléce
root.geometry("400x100") # ablak mérete
root.mainloop()
```

Az ablakot be lehet zárni a következő paranccsal

```
root.destroy()
```

3. Widgetek létrehozása

Tegyük fel, hogy csak egy nyomógomb van az ablakban. A nyomógomb a tkinter modul **Button** osztály meghívásával jön létre. Button objektum valamilyen változóhoz kapcsolódik. A Button osztálynak (mint minden más osztálynak, a Tk kivételével) van egy kötelező paramétere, az az objektum, amelyhez a nyomógomb tartozik (az nem lehet „nem birtokolható”). Amíg egyetlen ablakunk (root) van, a gombobjektumot oda kell elhelyezni:

```
but = Button(root)
```

4. Widget tulajdonságainak beállítása

Egy nyomógombnak számos tulajdonsága van: méret, háttérszín és címke stb. Egyelőre csak egy tulajdonságot telepítsünk, a felirat szövegét (text):

```
but["text"] = "Nyomtatás"
```

5-6. Eseményeket és azok kezelőinek meghatározása

Tegyük fel, hogy a nyomóngomb feladata valamilyen üzenet megjelenítése a nyomtatási függvény használatáról. És ezt akkor teszi meg, ha rákattint a bal egérgombbal. Az adott esemény során fellépő műveletek (algoritmusok) meglehetősen összetettek lehetnek. Ezért függvények formában alkalmazzák, majd meghívják, amikor szükség van rájuk.

```
def  printer(event):
     print ("Hello World!")
```

A függvényt célszerű (majdnem kötelező) a kód elejére tenni. Az **event** paraméter egy esemény. A bal egérgomb kattintási eseményhez egy kezelőt (pl., a printer függvényt) kell társítania. Kapcsolattartásért a **bind** metódust alkalmazzuk:

```
but.bind("<Button-1>",printer)
```

7. Widget elhelyezése

Meg kell jeleníteni a gombot az ablakban. A legegyszerűbb a **pack** metódus használata:

```
but.pack()
```

Ha nem szúrja be ezt a kódsort, a gomb nem jelenik meg az ablakban.

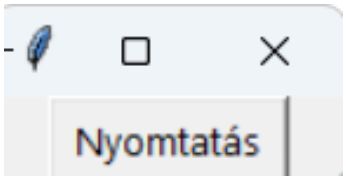
8. A fő ablak megjelenítése

És végül a főablak sem jelenik meg, amíg a speciális **mainloop** metódust meg nem hívják:

```
root.mainloop()
```

Ennek a kódsornak mindig a szkript végén kell lennie. Most a programkód így nézhet ki:

```
from tkinter import *

def printer(event):

     print ("Hello World!")
root = Tk()
but = Button(root)
but["text"] = " Nyomtatás "
but.bind("<Button-1>",printer)
but.pack()
root.mainloop()
```



Eredmény: Hello World!

A grafikus felhasználói felület programozásakor az *objektum-orientált* megközelítés hatékonyabb. Ezért ajánlatos a programot osztályként tervezni.

Minden widget osztálynak vannak bizonyos tulajdonságai, amelyeknek az értékei létrehozásakor beállíthatók, valamint beprogramozhatók a felhasználói műveletek során és a program végrehajtása során történő változásra.

Widgetek a Tkinterben

A widgetek a Python Tkinter GUI keretrendszerének magjai. Ezek azok az elemek, amelyeken keresztül a felhasználók kapcsolatba lépnek a programmal. A Tkinterben minden widgetet egy osztály határoz meg. Az alábbiakban felsoroljuk a Tkinter fontosabb widgetjeit:

Widget osztály	Leírás
Label	Szöveg megjelenítésére vagy kép beszúrására szolgál az alkalmazás ablakába.
Button	Olyan gomb, amelyen szöveg is lehet, és rákattintva végrehajt bizonyos műveleteket.

Entry	Beviteli widget egy sor szöveg beírásához. Egyenértékű az <input type="text"> HTML-ben.
Text	Szöveg widget nagy szöveg beviteléhez. Egyenértékű a <textarea>-val a HTML-ben.
Frame	Téglalap alakú terület, amely widgetek csoportosítására vagy a widgetek közötti térköz növelésére szolgál..

Widgetek javasolt elnevezésének szabályai

A widget létrehozásakor tetszőleges nevet adhatunk neki, feltéve, hogy a megadott nevet még nem regisztrálta a Python, mint például with, for, range stb.

Általában célszerű a widget osztálynevét belefoglalni a widget példányhoz rendelt változó nevében. Például, ha a Label widgetet egy felhasználó nevének megjelenítésére használják, elnevezheti a widgetet label_user_name. A felhasználó életkorának meghatározására használt Entry widgetet entry_age nevezhetjük.

Ha a változónévbe belefoglalja a widget osztálynevét, segít megérteni, hogy a változónév milyen típusú widgetre vonatkozik.

A teljesen minősített eszközprimitív osztálynév használata azonban hosszú változóneveket eredményezhet, ezért érdemes lehet rövidítést használni az egyes widgettípusokra való hivatkozáshoz.

Pl. *lbl_name (Label), btn_submit (Button), ent_age (Entry), txt_notes (Text), frm_cim (Frame).*

VISSZA

2. Label (cimke)

Ha már van egy ablak, akkor hozzáadhatunk widgeteket. Például, használjuk a **tk.Label** osztályt, és adjunk hozzá szöveget az ablakhoz. Készítsünk egy címke widgetet a "Hello, Tkinter!", és rendeljünk hozzá egy greeting nevű változóhoz:

```
greeting = tk.Label(text="Szia, Tkinter!")
```

A widget létrejött, de még nincs hozzáadva az ablakhoz. Többféleképpen is hozzáadhatunk widgeteket egy ablakhoz. Használhatjuk a **pack()** metódust:

```
greeting.pack()
```

Amikor a pack() metódussal widgetet helyez el egy ablakban, a Tkinter a lehető legkisebbre állítja az ablak méretét, amíg a widget el nem fér. Most tegyük a következőket:

```
window.mainloop()
```

Ha nem adja hozzá a **window.mainloop()**-ot a program végéhez a Python fájlban, akkor a Tkinter alkalmazás egyáltalán nem indul el, és semmi sem jelenik meg.

A Label widget szöveget és képeket jelenít meg. A Label widgeten lévő szöveget a felhasználó nem szerkesztheti. Az csak megjelenik a formon (ablakban). A címkék meglehetősen egyszerű widgetek, amelyek egy sort (vagy több sort) tartalmaznak szöveget, és elsősorban a felhasználó tájékoztatása.

```
lab = tk.Label(root, text="Ez cimke! \n Két soros.",
     font="Arial 18")
```

A Label widgetet a Label osztály példányosításával és egy karakterlánc átadásával lehet létrehozni a szöveges paraméternek:

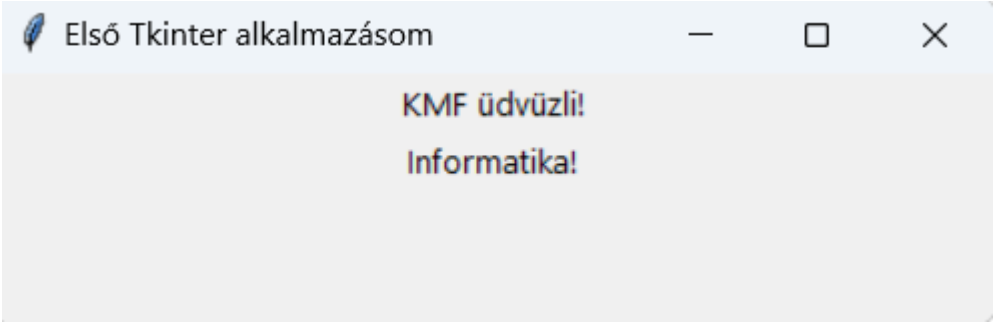
```
Label = tk.Label(text="Hello, Tkinter")
```

A címkemodulok az alapértelmezett rendszerszínnel és háttérrel jelenítenek meg szöveget. A színek általában fekete-fehérek. Ezért, ha az operációs rendszere más színeket ad meg, ezeket a színeket fogja látni.

Példa.

```
import tkinter as tk
# Fő ablak létrehozása
root = tk.Tk()
root.title("Első Tkinter alkalmazásom")
root.geometry("400x300") # ablak mérete
# Cimke (Label)
label = tk.Label(text="KMF üdvüzli!")
label.pack() # cimke az ablakra
label2 = tk.Label(root, text="Informatika!")
label2.pack()
```

```
# Fő ciklus inditása
root.mainloop()
```

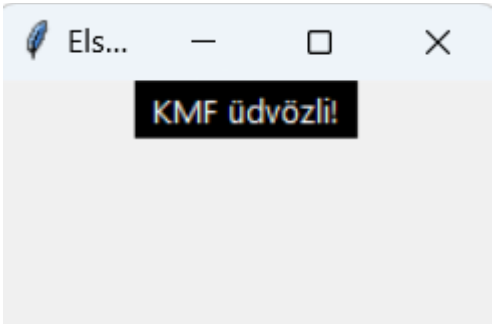


A widget szövegét és háttérszínét az előtér és a háttér paramétereivel módosíthatja:

```
label = tk.Label(
    text="Szia, Tkinter!",
    foreground="white", # Fehér szöveget állít be
    background="black" # A hátteret feketére állítja
)
```

Egy kis módositás után:

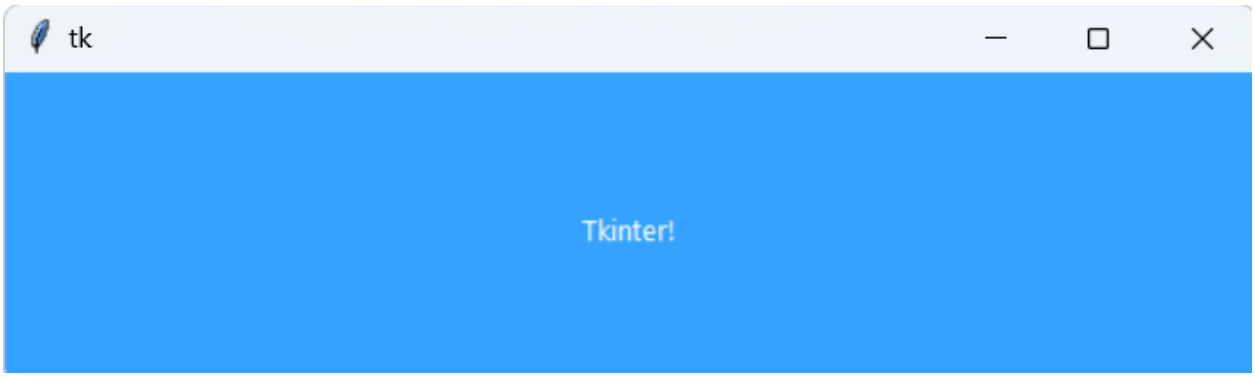
```
import tkinter as tk
root = tk.Tk()
root.title("Első Tkinter alkalmazásom")
root.geometry("200x100") # ablak mérete
label = tk.Label(text=" KMF üdvözli! ", fg="white",
bg="black")
label.pack() # cimke az ablakra
root.mainloop()
```



Néhány gyakran alkalmazott szín:

- *red – piros;*
- *orange – narancs;*
- *yellow – sárga;*
- *green – zöld;*
- *blue – kék;*
- *purple – lila.*

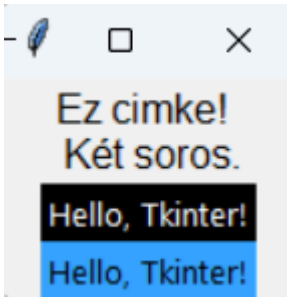
```
import tkinter as tk
window = tk.Tk()
label = tk.Label(
    text="Tkinter!",
    fg="white",    bg="#34A2FE",    width=80,    height=8)
label.pack()
window.mainloop()
```



Sok HTML-színnek ugyanaz a neve a Tkinterben is. A színt megadhatunk hexadecimális RGB-értékekkel is, amelyet gyakran használnak a CSS-ben a webhelyek stílusához:

```
import tkinter as tk
root=tk.Tk()
lab = tk.Label(root, text="Ez cimke! \n Két soros.",
font="Arial 12")
lab.pack()
lab1 = tk.Label(text="Hello, Tkinter!", fg="white",
bg="black")
```

```
lab1.pack()
lab2 = tk.Label(text="Hello, Tkinter!", background="#34A2FE")
lab2.pack()
root.mainloop()
```

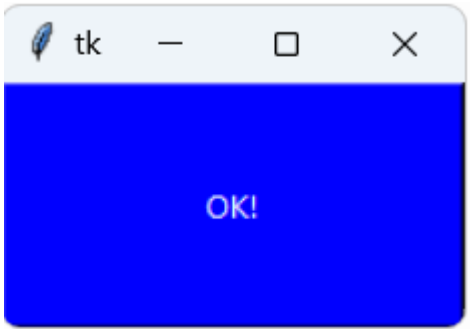


Most szép kék színű a háttér. A hexadecimális RGB értékek a színnevekkel ellentétben kódolva vannak, és így jobban kezelhetők. Rendelkezésre állnak eszközök a hexadecimális színkódok egyszerű és gyors beszerzéséhez.

```
import tkinter as tk
window = tk.Tk()
label = tk.Label(text="Hello, Tkinter!", fg="white",
bg="black", width=20, height=10 ); label.pack()
window.mainloop()
```


```
import tkinter as tk
window = tk.Tk()
```

```
button = tk.Button( text="OK!", width=25, height=5,
    bg="blue", fg="white"); button.pack()
window.mainloop()
```



A következő két widget, az *Entry* és a *Text* a felhasználó bemeneti adatainak bevitelére szolgál.

VISSZA

3. Entry (Egysoros szövegmező)

Azokban az esetekben, amikor szöveges információkat kell kérni a felhasználótól, például egy e-mail címet, az **Entry** widgetet használják. Egy szövegmezőt jelenít meg, ahová a felhasználó beírhat szöveget.

Az Entry widget létrehozása gyakorlatilag nem különbözik a parancsikon és nyomógomb létrehozásának folyamatától. Például a következő kód létrehoz egy widgetet kék háttérrel és sárga szöveggel, amely 50 szövegegység hosszúságú:

```
entry = tk.Entry(fg="yellow", bg="blue", width=50)
```

Egy ilyen mező a tkinter modul Entry osztályának meghívásával jön létre. A felhasználó csak egy sort írhat be.

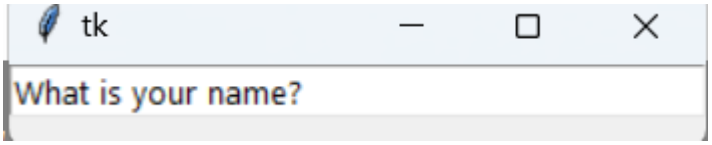
```
ent = Entry(root,width=20,bd=3)
```

A bd a borderwidth rövidítése.

Példa.

```
import tkinter as tk
window = tk.Tk()
entry = tk.Entry(width=40, bg="white", fg="black")
entry.pack()
```

```
entry.insert(0, "What is your name?")
window.mainloop()
```



Három fő művelet végezhető el az Entry widgettel:

- **get() –** a szöveg lekérése egy változóba
- **delete() –** szöveg törlése
- **insert() –** új szöveg beszúrása

Az **Entry** widgetek a **pack()** metódus végrehajtása után válnak láthatóvá az ablakon:

```
entry.pack()
```

A szöveg lekéréséhez és értékének pl. a **name** változóhoz való hozzárendeléséhez használjuk a **get()** metódust:

```
name = entry.get()
```

A szöveg is a **delete()** metódussal törölhető. Ez a metódus egy argumentumot vesz fel, amely egy egész szám, és megmondja a Pythonnak, hogy melyik karaktereket kell eltávolítani. Például az alábbi kód eltávolítja az első karaktert a szövegmezőből:

```
entry.delete(0)
```

A Pythonban a karakterláncokhoz hasonlóan az egysoros szövegdoboz widgetek szövege is indexelve van, 0-tól kezdve.

Ha több karaktert kell törölnie egy szövegmezőből, egy második egész argumentumot kell átadnia a delete()-nek, jelezve annak a karakternek az indexét, amelynél a törlési folyamat véget ér. Például a következő kód eltávolítja az első négy betűt egy szövegmezőből:

```
entry.delete(0, 4)
```

A delete() metódus hasonlóan működik, mint a string **slice()** metódus a karakterek egy részének eltávolítására a karakterláncból. Az első argumentum a törlés kezdő indexét adja meg, az utolsó index pedig azt, hogy pontosan hol álljon le a törlési folyamat.

FONTOS! Gyakran alkalmazzák. A szövegmező teljes szövegének eltávolításához a delete() metódus második argumentuma a **tk.END** speciális állandót használunk:

```
entry.delete(0, tk.END)
```

Az **insert()** metódussal szöveget lehet beszúrni az egysoros szövegdoboz widgetbe:

```
entry.insert(0, "János")
```

Az első argumentum megmondja az insert() metódusnak, hogy hova kell beszúrni a szöveget. Ha a szövegmezőben nincs szöveg, akkor az új szöveg a widget elejére kerül, függetlenül attól, hogy milyen értéket adunk át első argumentumként.

Ha a szövegmező már tartalmaz szöveget, akkor az insert() új szöveget szúr be a megadott helyre, és a meglévő szöveget jobbra tolja:

```
entry.insert(0, "Kati")
```

Az egysoros szövegmező widgetek kiválóan alkalmasak arra, hogy rövid szöveget kapjanak a felhasználótól, de csak egy sornyi szöveget jelenítenek meg, így nem alkalmasak nagy mennyiségű információ tárolására. Ilyen esetekben jobb a **Text** widgetek használata.

VISSZA

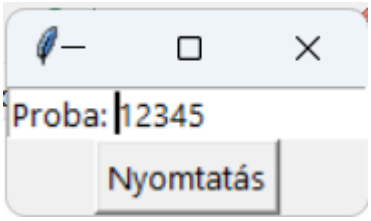
3.1. Textvariable változó alkalmazása

A **textvariable** a Tkinterben egy speciális paraméter, amely Python változót társít egy Tkinter widgethez. Ha ennek a változónak az értéke megváltozik, az automatikusan megjelenik a kapcsolódó widgetben, és fordítva.

Az Entry widgetnél a szövegváltozó általában a felhasználó által beírt szöveg kényelmes szabályozására szolgál. Egy példa. Egy Tkinter ablakot hozunk létre egy Entry widgettel. Létrehozunk egy **var** változót, és a **textvariable** paraméteren keresztül hozzárendeljük az Entry widgethez. Amikor a felhasználó szöveget ír be az Entry mezőbe, ez az érték automatikusan a **var** mezőbe kerül. A "Nyomtatás" gombra kattintva megjelenítjük a konzolon a **var** aktuális értékét, vagyis azt, amit a felhasználó beírt.

```
import tkinter as tk
def print_entry_value():
    print("A  bevitt érték:", var.get())
root = tk.Tk(); var = tk.StringVar()
entry = tk.Entry(root, textvariable=var); entry.pack()
```

```
button = tk.Button(root, text=" Nyomtatás",
command=print_entry_value); button.pack()

root.mainloop()
```

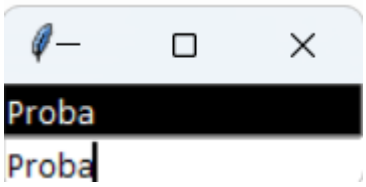


A bevitt érték: *Proba: 12345*

Példa

Itt az egyik szövegmező tartalma azonnal megjelenik a másikban, mert mindkét mező ugyanahhoz a **v** változóhoz van kötve.

```
from tkinter import *
root = Tk()
v = StringVar()
ent1 = Entry (root, textvariable = v,bg="black",fg="white")
ent2 = Entry(root, textvariable = v)
ent1.pack()
ent2.pack()
root.mainloop()
```



VISSZA

4. Text (Többsoros szövegmező)

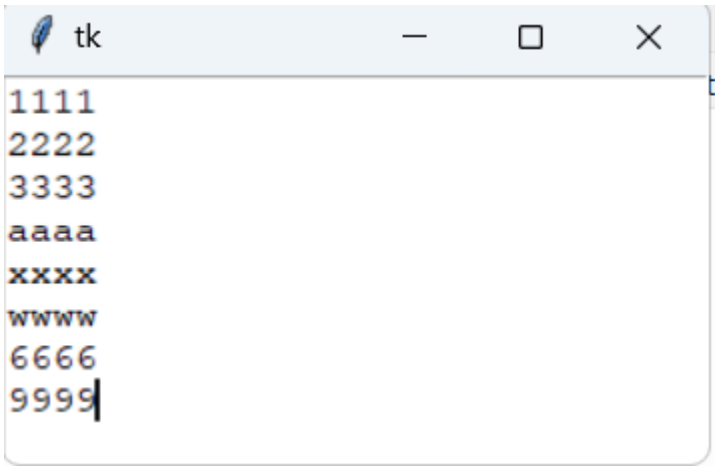
Text widgetek szövegbevitelre szolgálnak, akárcsak a Entry widgetek. A különbség az, hogy a szöveg több sornyi szöveget is tartalmazhat.

```
tex = tk.Text(root,width=40,font="Verdana 12", wrap=WORD)
```

A **wrap** tulajdonság értékétől függően, lehetővé teszi, hogy a felhasználó által beírt szövegetkarakterrel vagy szóval tördeljük, vagy egyáltalán nem tördeljük, amíg a felhasználó meg nem nyomja az *Entert.*

A Text widget segítségével a felhasználó egész bekezdéseket vagy szövegoldalakat írhat be.

```
import tkinter as tk
window = tk.Tk()
text_box = tk.Text();
text_box.pack()
window.mainloop()
```



Az Entry widgetekhez hasonlóan a Text widgetekkel is három alapvető műveletet hajthat végre:

- Szöveg lekérése **get()** metódussal
- Szöveg törlése **delete()** metódussal
- Szöveg beszúrása **insert()** metódussal

Bár a metódusnevek megegyeznek az Entry metódusaival, a megvalósítási folyamat kissé eltér.

Alapértelmezés szerint a szövegdobozok lényegesen nagyobbak, mint az Entry egysoros szövegbeviteli widgetek. A szövegmező aktiválásához kattintson az ablak bármely pontjára. Írja be a „Hello” szót. Nyomja meg az Enter gombot a billentyűzeten, majd írja be a „World” szót a második sorba.

Az Entry widgetekhez hasonlóan a Text widgetek szövegét megkaphatja a **get()** metódussal. A get() argumentumok nélküli meghívása azonban nem adja vissza a szöveget, mint az Entry egysoros szövegbeviteli widgetek esetében. A rendszer TypeError kivételt dob fel:

```
text_box.get()
```

Traceback (most recent call last):

File "<pyshell#4>", line 1, in <module>

text_box.get()

TypeError: get() missing 1 required positional argument: 'index1'

A szöveges widget get() metódusa legalább egy argumentumot igényel. A get() egyetlen indexű meghívása egy karaktert ad vissza. Ha több karaktert szeretne kapni, át kell adnia a kezdő és a záró indexet. A Text widgetekben az indexek másképp működnek, mint az Enter widgetekben. Mivel a Text widgetek több sornyi szöveget is fogadhatnak, az *indexnek a következő két elemet* kell tartalmaznia:

• sorszám, ahol a szimbólum található;

• a karakter pozíciója a sorban.

A sorszámok 1-től kezdődnek, a karakterek pozíciója pedig 0-tól.

Az index megszerzéséhez egy "<sor>.<char>" formátumú karakterlánc jön létre, ahol a <sor> helyére a sorszám, a <char> pedig a sorszámra kerül. a karakter számát. Például az „1.0” az első sor első karakterét jelöli, a „2.3” pedig a második sor negyedik karakterét.

Az "1.0" indexet arra használjuk, hogy megkapjuk a korábban létrehozott szövegmező első betűjét:

text_box.get("1.0")

A „Hello” szónak 5 betűje van, az o betű a 4-es indexnél van, mivel a karakterek száma 0-tól kezdődik, a „Hello” szó pedig a szövegmező első sorából kezdődik. Ahhoz, hogy a teljes „Hello” szót a szövegmezőből lehessen lekérni, a végindexnek eggyel nagyobbnak kell lennie, mint az utolsó beolvasott karakter. Tehát, ha a „Hello” szót egy szövegmezőből szeretné megkapni, használja az „1.0” értéket az első indexhez, és az „1.5”-et a második indexhez:

text_box.get("1.0", "1.5")

'Helló'

Ha a "World" szót a szövegmező második sorából szeretné megkapni, módosítsa a sorszámot 2-re:

text_box.get("2.0", "2.5")

'World'

A szöveg widgetből való teljes szöveg lekéréséhez állítsa az indexet "1.0"-ra, és használja a speciális **tk.END** konsztanst a második indexhez:

```
text_box.get("1.0", tk.END)
```

```
'Hello\nWorld\n'
```

A **get()** metódus által visszaadott szöveg **\n** (új sor) karaktert tartalmaz. Ebben a példában azt is láthatja, hogy a Text widget minden sora újsor karakterrel végződik, beleértve a szövegmező utolsó sorát is.

A **delete()** metódus karakterek eltávolítására szolgál egy szöveges widgetből. Pontosan ugyanúgy működik, mint a delete() metódus az Entry widgetek esetében. A delete() kétféleképpen használható: egy vagy két argumetummal.

Az egyargumentumos opció használatával átadja a törölni kívánt karakter indexét a delete()-nek. Például a következő kód eltávolítja az első karaktert:

```
text_box.delete("1.0")
```

Most a szöveg első sora: "elló". A kétargumentumú változat két indexet ad át a karakterek egy csoportjának eltávolításához, kezdve az első karakterrel és a második karakterrel végződve.

Például az „ello” fennmaradó részének eltávolításához a szövegmező első sorából használja az „1.0” és „1.4” indexet:

```
text_box.delete("1.0", "1.4")
```

Az első sorban egy karakter maradt - az újsor karakter \n. Ezt láthatjuk a get() metódus meghívásával:

```
text_box.get("1.0")
```

```
'\n'
```

Ha eltávolítja ezt a karaktert, a szöveg widget fennmaradó tartalma egy sorral feljebb kerül:

```
text_box.delete("1.0")
```

Most a "World" szó átkerült a szöveg widget első sorába. Próbáljuk törölni a fennmaradó részt a szöveg widgetből. Állítsuk be az "1.0"-t kezdő indexnek, és a tk.END-t második indexnek:

```
text_box.delete("1.0", tk.END)
```

Most a szöveges widgetünk teljesen üres.

Egy Text widgetbe szöveget is beszúrhatunk az insert() metódussal:

```
text_box.insert("1.0", "Hello")
```

Az **insert** metódus beszúrja a "Hello" szót a szöveg widget elejére. A már ismert „<sor>.<oszlop>” formátumot használjuk, akárcsak a get() metódust a szövegbeillesztési pozíció

tisztázására. Ha új sorba szeretnénk helyezni a szöveget, akkor kell beilleszteni az újsor \n karaktert:

```
text_box.insert("2.0", "\nWorld")
```

Most a "World" szó a szöveg widget második sorában található.

Az **insert()** metódus a következő két műveletet hajtja végre:

- **Szöveg beszúrása a megadott helyre, ha már van szöveg ezen a helyen vagy utána;**
- **Szöveget ad hozzá a megadott sorhoz, ha a karakterszám nagyobb, mint a szövegmező utolsó karakterének indexe.**

Általában nem célszerű az utolsó karakter indexét követni. A Text widget végére való szöveg beszúrásának legjobb módja - a tk.END parameter alkalmazása:

```
text_box.insert(tk.END, "Szúrj be a legvégére!")
```

Ne felejtsen el egy újsor karaktert (\n) elhelyezni a szöveg elejében, ha új sorba szeretné helyezni:

```
text_box.insert(tk.END, "\nSzúrj be egy új sort!")
```

VISSZA

5. Button (Nyomógomb)

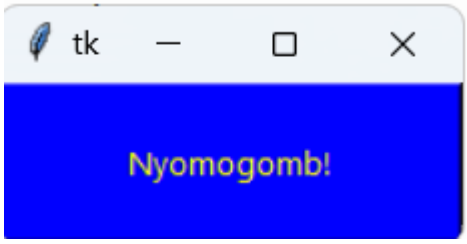
A kattintható nyomógombok létrehozásához **Button** widgetek szükségesek. Beállíthatók úgy, hogy megnyomásakor egy adott függvényt hívjon meg. Egyelőre kezdjük el a gomb widget stílusának kialakítását.

Sok hasonlóság van a Button és Label widgetek között. Lényegében egy gomb csak egy parancsikon, amelyre rákattinthat. Ugyanezeket az argumentumokat használják a címke- és gombmodulok stílusainak létrehozására is.

Egy gombobjektum a tkinter modul Button osztályának meghívásával jön létre. Ebben az esetben csak a szülő widget kötelező argumentum. Más tulajdonságok megadhatók a gomb létrehozásakor vagy későbbi beállításakor (módosításakor).

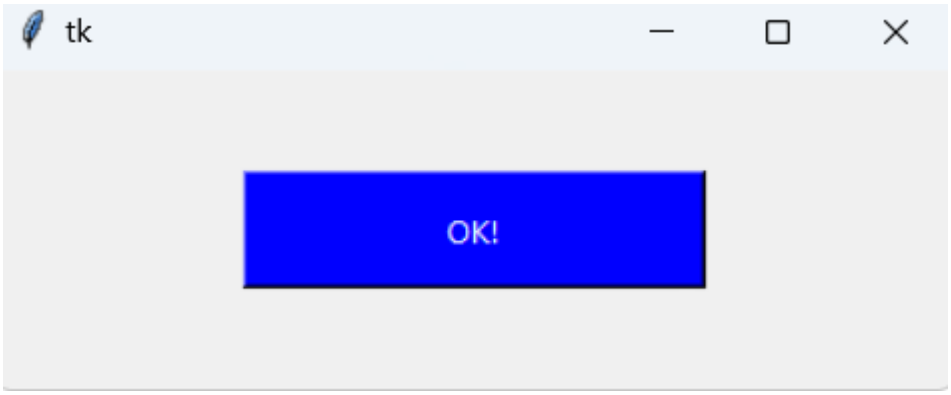
Például a következő kód létrehoz egy gombot kék háttérrel és sárga szöveggel. A szélesség és magasság 25 és 3 szövegegység értékkel van beállítva:

```
import tkinter as tk
window = tk.Tk()
button = tk.Button(text="Nyomogomb!",  width=25, height=3,
    bg="blue", fg="yellow"); button.pack()
window.mainloop()
```



Példa.

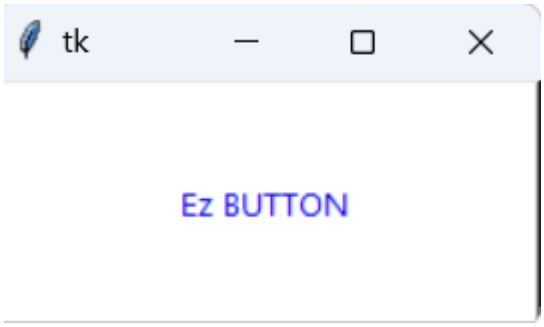
```
import tkinter as tk
window = tk.Tk()
button = tk.Button( text="OK!", width=25, height=2,
    bg="blue", fg="white")
button.pack( padx=100, pady=40)
window.mainloop()
```



Példa.

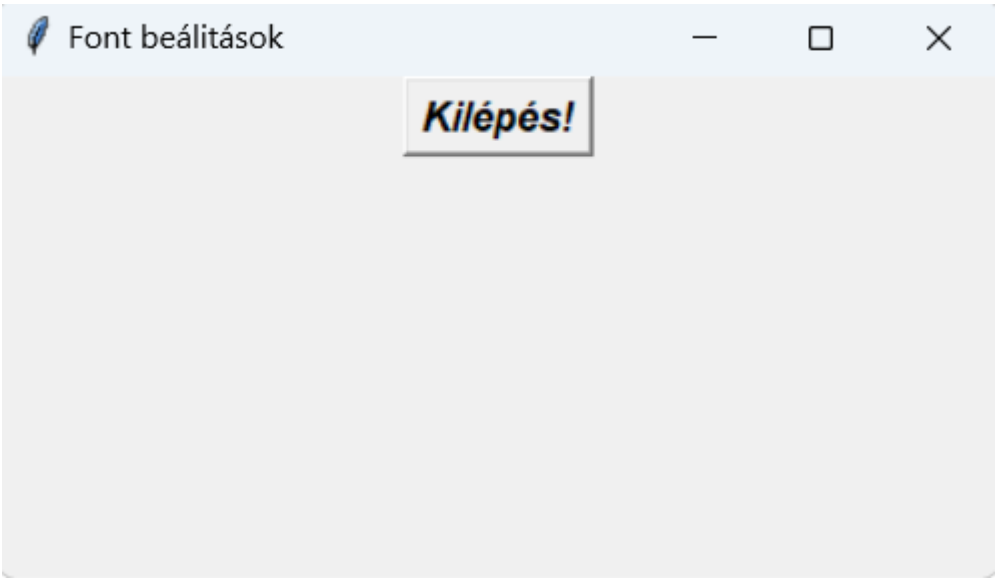
```
from tkinter import *
root = Tk()
but = Button(root,
text="Ez BUTTON", # felirat a gombon
width=30,height=5, # szélesség és magasság
bg="white",fg="blue") #háttér és felirat szine
```

```
but.pack()
root.mainloop()
```



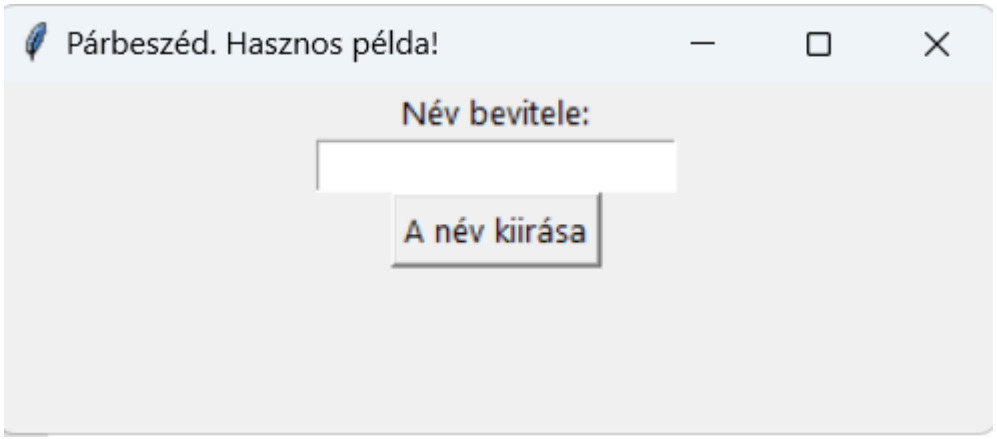
Példa. Fontbeállítások

```
import tkinter as tk
root = tk.Tk() # ablak létrehozása
root.option_add("*font","Arial 12 bold italic")
root.title("Font beálitás") # ablak fejléce
root.geometry("400x200") # ablak mérete
but = tk.Button(root, text="Kilépés!", command=root.quit)
but.pack()
root.mainloop()
```



Példa.

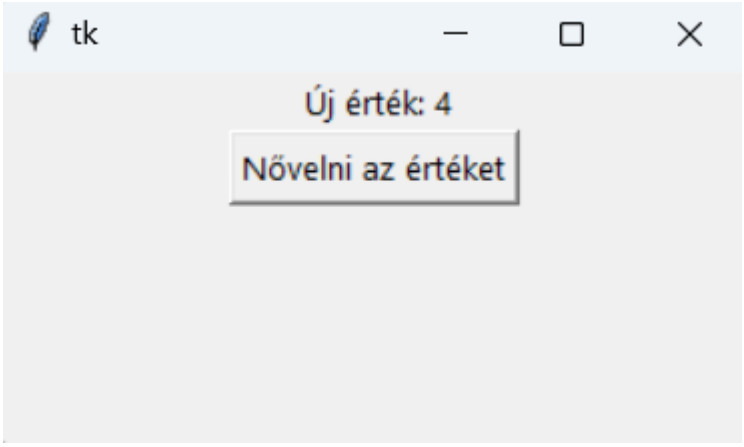
```
def show_name():
    name = entry.get()
    label2.config(text="Üdvözöljük, " + name + "!")

import tkinter as tk
root = tk.Tk()
root.title("Párbeszéd. Hasznos példa!")
root.geometry("400x140") #
label = tk.Label(root, text="Név bevitele:");label.pack()
entry = tk.Entry(root);entry.pack()
button = tk.Button(root, text="A név kiirása",
command=show_name); button.pack()
label2 = tk.Label(root, );label2.pack()
label2 = tk.Label(root, text=""); label2.pack()
root.mainloop()
```



Példa

Globális változó segitségével megváltozható az **Entry** tartalma:

```
def increment_global():
    global glob
    glob += 1
    label.config(text=f" Új érték: {glob}")

import tkinter as tk
glob = 0
root = tk.Tk()
root.geometry("200x100")
label = tk.Label(root, text=f"Globalis  Változó: {glob}")
label.pack()
button = tk.Button(root, text="Növelni az értéket",
command=increment_global); button.pack()
root.mainloop()
```



VISSZA

6. Rádióbutton (választógomb, rádiógomb)

A rádiógomb objektumot soha nem használják egyedül. Csoportosan használják őket, és a csoportban csak egy gombot

lehet „bekapcsolni”. Az **IntVar()** típusú objektumváltozó használata választógomb-csoport létrehozásakor:

```
v=IntVar()
v.set(1)
rad0 = Radiobutton(root,text="első", variable=v, value=0)
rad1 = Radiobutton(root,text="második”, variable=v,value=1)
rad2 = Radiobutton(root,text="harmadik", variable=v,value=2)
```

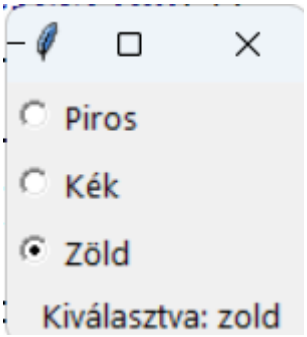
Létrejön az **IntVar** osztály egy objektuma, és hozzárendelődik a **v** változóhoz, és a **set()** metódus a kezdeti értéket 1-re állítja. A csoport egy változó értékét határozza meg, vagyis ha a példában a rad2 rádiógombot választjuk, akkor a **v** változó értéke 2 lesz. Három választógomb ugyanabba a csoportba tartozik: ezt bizonyítja a variable opció (tulajdonság) azonos értéke. A **variable** úgy van kialakítva, hogy egy Tkinter-változót egy rádiógombhoz társítson. A **value** opció megadja azt az értéket, amelyet a rendszer átad a változónak, ha a gomb aktiv állapotban van. Ha a **v** változó megváltozik, ez hatással lesz a csoportra. A **get** metódus lehetővé teszi a változó értékének fogadását.

```
def display(event):
     v = var.get()
     if v == 0:
          print ("Első gomb van bekapcsolva")
     elif v == 1:
          print ("Második gomb van bekapcsolva ")
     elif v == 2:
          print ("Harmadik gomb van bekapcsolva ")
but = Button(root,text="Megadni az értéket")
but.bind('<Button-1>',display)
```

A **display** függvény meghívásakor a **v** változó a **get()** metódus által a **var** változó értékét kapja.

Példa. Radiobutton alkalmazása: szin kiválasztása.

```
def show():
    lab.config(text=" Kiválasztva: " + var.get())

import tkinter as tk
root = tk.Tk(); root.title("Szin választása")
var = tk.StringVar()
tk.Radiobutton(root, text="Piros", variable=var,
value="piros", command=show).pack(anchor=tk.W)
tk.Radiobutton(root, text="Kék", variable=var, value="kék",
command=show).pack(anchor=tk.W)
tk.Radiobutton(root, text="Zöld", variable=var, value="zold",
command=show).pack(anchor=tk.W)
lab = tk.Label(root, text=""); lab.pack()
root.mainloop()
```



anchor=tk.W opció magyarázata. A Tkinter pack() metódusában található *anchor=tk.W* opció a widget szülőtárolóban lévő pozíciójának horgonyának beállítására szolgál.

A tk.W az ablak nyugati (bal) szélére mutat. Ez azt jelenti, hogy a widget úgy lesz igazítva, hogy a bal széle a szülőtároló bal oldalán legyen.

VISSZA

7. Checkbutton (Jelölőnégyzet, Jelölőgomb)

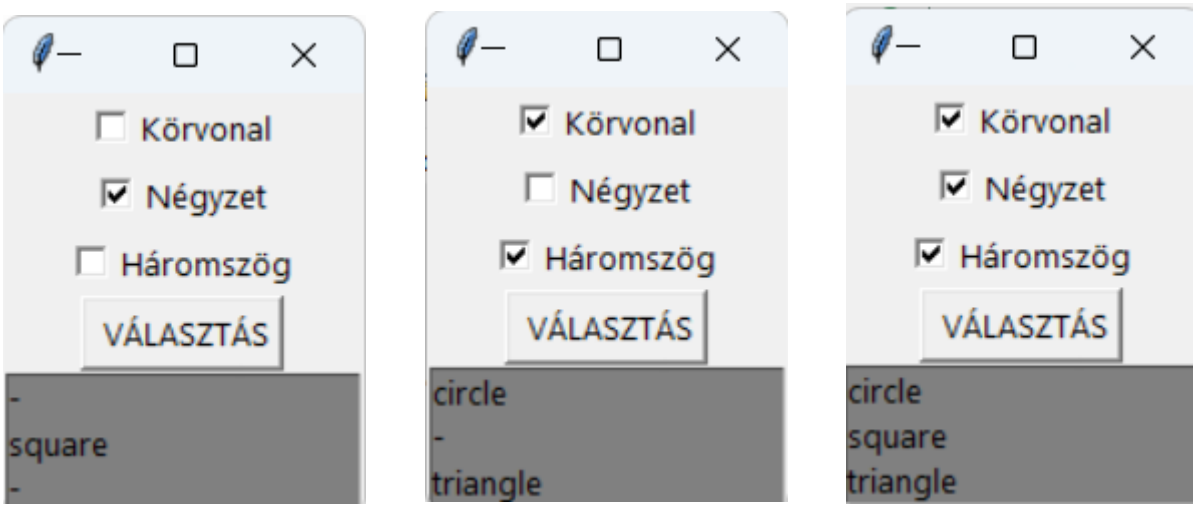
A checkbutton objektum nem kölcsönösen kizáró elemek kijelölésére szolgál egy ablakban (egy csoportban aktiválhat egynél több elemet). A rádiógombokkal ellentétben minden jelölőnégyzet értéke saját változóhoz van kötve, amelynek értéke a jelölőnégyzet leírásában szereplő **onvalue** (enabled, bekapcsova) és **offvalue** (disabled, kikapcsolva) opciók határozzák meg.

```
c1 = IntVar()
c2 = IntVar()
che1 = Checkbutton(root,text="első",
variable=c1,onvalue=1,offvalue=0)
che2 = Checkbutton(root,text="második",
variable=c2,onvalue=2,offvalue=0)
```

A Checkbutton-nal valamivel bonyolultabb a helyzet, mint a Rádióbuttom-nál, mivel Checkbutton állapotai függetlenek egymástól, mindegyikhez saját társított objektumváltozóval kell rendelkeznie.

```
def result(event):
    v0 = var0.get()
    v1 = var1.get()
    v2= var2.get()
```

```
        l = [v0,v1,v2]
        lis.delete(0,2)
        for v in l:
            lis.insert(END,v)

from tkinter import *
root = Tk()
var0=StringVar(); var1=StringVar(); var2=StringVar()
ch0 = Checkbutton(root,text="Körvonal",variable=var0,
onvalue="circle",offvalue="-")
ch1 = Checkbutton(root,text="Négyzet",variable=var1,
onvalue="square",offvalue="-")
ch2 = Checkbutton(root,text="Háromszög",variable=var2,
onvalue="triangle",offvalue="-")
ch0.deselect(); ch0.pack()
ch1.deselect(); ch1.pack()
ch2.deselect(); ch2.pack()
but = Button(root,text=" VÁLASZTÁS")
but.bind('<Button-1>',result); but.pack()
lis = Listbox(root,height=3,bg='grey'); lis.pack()
root.mainloop()
```



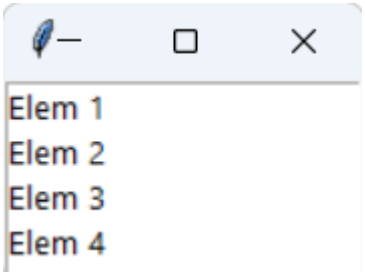
VISSZA

8. Listbox (Lista, Lista mező)

A **Listbox** osztály meghívása létrehoz egy objektumot, amelyben a felhasználó kiválaszthat egy vagy több elemet az opció értékétől függően.

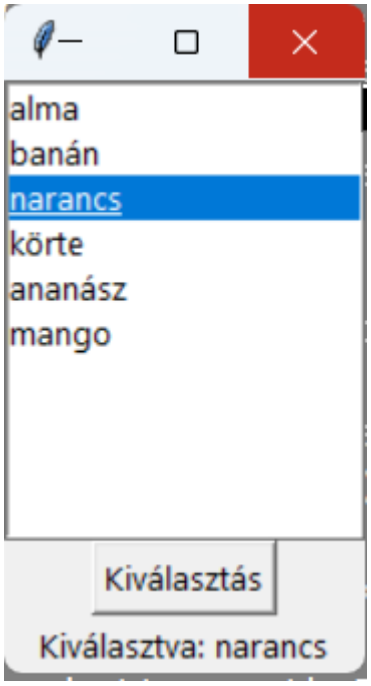
```
import tkinter as tk
root = tk.Tk()
listbox = tk.Listbox(root)
listbox.pack()
```

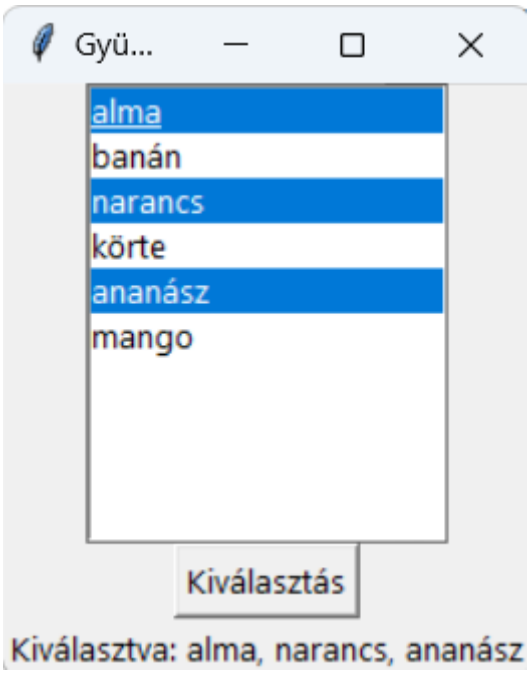
```
for i in range(4):
    listbox.insert(tk.END, f"Elem {i+1}")
root.mainloop()
```



Példa. Több elem kiválastása a listából.

```
def selection():
    selected = [listbox.get(idx) for idx in
    listbox.curselection()]
    if selected:
        lab.config(text="Kiválasztva: " + ", ".join(selected))
    else:
        lab.config(text="Min egy elemet választjon!")

import tkinter as tk
root = tk.Tk()
root.title("  Gyümölcs választása")

fruits = ["alma", "banán", "narancs", "körte", "ananász",
"mango"]
listbox = tk.Listbox(root, selectmode=tk.MULTIPLE)
for fruit in fruits:
    listbox.insert(tk.END, fruit)
listbox.pack()
confirm_button = tk.Button(root, text="Kiválasztás",
command=selection); confirm_button.pack()
lab = tk.Label(root, text=""); lab.pack()
root.mainloop()
```



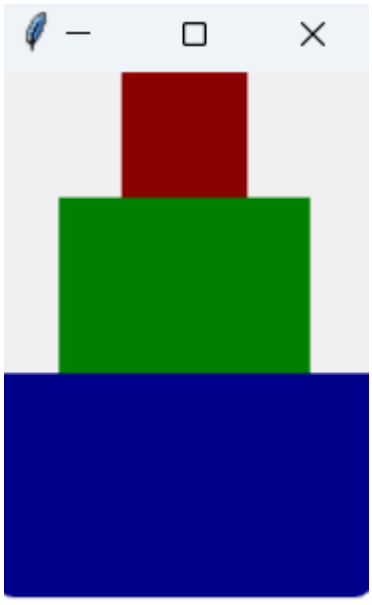


VISSZA

9. Frame (Keret)

A keretek jó eszközt jelentenek más widgetek az ablakon belüli csoportokba rendezésére, valamint a tervezésre.

```
from tkinter import *
root = Tk()
fra1 = Frame(root,width=50,height=50,bg="darkred")
fra2 = Frame(root,width=100,height=60,bg="green",bd=20)
fra3 = Frame(root,width=150,height=70,bg="darkblue")
fra1.pack(); fra2.pack(); fra3.pack()
root.mainloop()
```



Ez a szkript három különböző méretű keretet hoz létre. A bd tulajdonság (a boderwidth rövidítése) határozza meg a keret széle és a benne lévő widgetek távolságát (ha vannak).
A widgeteket a keretekre is lehet elhelyezheti, mint a fő ablakban:

```
ent1 = Entry(fra2,width=20)
ent1.pack()
```

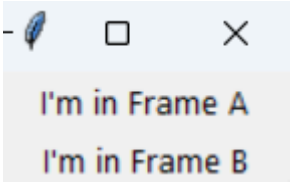
A következő szkript létrehoz egy üres keret widgetet, és hozzáadja az alkalmazás főablakához:

```
import tkinter as tk
window = tk.Tk()
frame = tk.Frame()
frame.pack()
window.mainloop()
```

A frame.pack() metódus keretet helyez az ablakra, és az ablakot olyan kicsire szabja, amennyire csak lehet, és hogy beleférjen a keretbe. A fenti szkript futtatásakor az üres frame widget gyakorlatilag láthatatlan. A Frame-t úgy lehet tekinteni, mint widgetek konténereit.

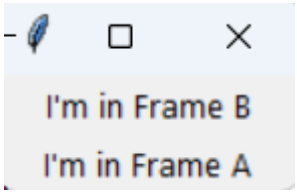
Most írjunk egy szkriptet, amely két Frame widgetet hoz létre frame_a és frame_b néven. Ebben a szkriptben a frame_a egy **"I'm in Frame A"** szövegű címkét, a frame_b pedig egy **"I'm in Frame B"** szövegű címkét tartalmaz. Íme, hogyan kell csinálni:

```
import tkinter as tk
window = tk.Tk()
frame_a = tk.Frame(); frame_a.pack()
frame_b = tk.Frame(); frame_b.pack()
label_a = tk.Label(master=frame_a, text="I'm in Frame A")
label_b = tk.Label(master=frame_b, text="I'm in Frame B")
label_a.pack(); label_b.pack()
window.mainloop()
```



A **frame_a** az ablakban a **frame_b** elé kerül. A megnyíló ablak a **frame_a** címkét mutatja a **frame_b** címke felett. Most pedig nézzük meg, mi történik, ha megváltoztatjuk a **frame_a.pack()** és **frame_b.pack()** beillesztési sorrendjét:

```
import tkinter as tk
window = tk.Tk()
frame_b = tk.Frame(); frame_b.pack()
frame_a = tk.Frame(); frame_a.pack()
label_a = tk.Label(master=frame_a, text="I'm in Frame A")
label_b = tk.Label(master=frame_b, text="I'm in Frame B")
label_b.pack(); label_a.pack()
window.mainloop()
```



Master attribútum

Mind a négy eddig tárgyalt widgettípus – **Label**, **Button**, **Entry** **és** **Text**

rendelkezik egy *master* attribútummal, amely a létrehozáskor kerül beállításra. Így szabályozhatja, hogy a widget melyik kerethez legyen hozzárendelve. A keret widgetek kiválóan alkalmasak más widgetek logikai rendszerezésére az alkalmazás ablakában. A kapcsolódó widgetek ugyanahhoz a kerethez rendelhetők, így ha a keret valaha elmozdul az ablakban, a kapcsolódó widgetek is vele együtt mozognak.

relief attribútum

A keretek megváltoztathatják stílusukat a **relief** attribútum használatával, amely szegélyt hoz létre a keret körül:

- tk.FLAT: Nincs kereteffektus (alapértelmezett);
- tk.SUNKEN: Az elem elmélyítésének hatását hozza létre;
- tk.RAISED: Kidudorodó hatást hoz létre az elemen;

• tk.GROOVE: A textúrába ágyazott keret hatását hozza létre, egyfajta bemélyedést;

• tk.RIDGE: Emelt bevágás hatást hoz létre.

borderwidth attribútum

A szegélyeffektus alkalmazásához a **borderwidth** attribútum értékét 1-nél nagyobbra kell beállítani. Ez az attribútum beállítja a szegély szélességét képpontokban.

Íme egy szkript, amely öt keretet helyez el egy ablakon, mindegyik más-más relief argumentumértékkel:

```
import tkinter as tk
border_effects = {flat": tk.FLAT, "sunken": tk.SUNKEN,
    "raised": tk.RAISED,"groove": tk.GROOVE, "ridge": tk.RIDGE,}
window = tk.Tk()
for relief_name, relief in border_effects.items():
    frame = tk.Frame(master=window, relief=relief, borderwidth=5)
    frame.pack(side=tk.LEFT)
    label = tk.Label(master=frame, text=relief_name)
    label.pack()
window.mainloop()
```

A szkript elemzése soronként:

• 3-9. sorok egy szótárt hoznak létre, amelynek kulcsai a Tkinterben elérhető különféle dombormű-effektusok nevei. Ez a szótár a **border_effects** változóhoz van hozzárendelve

• A 13. sor egy ciklust futtat a **border_effects** szótár elemeihez

• A 14. sor létrehoz egy új Frame widgetet, és hozzárendeli az ablak objektumhoz. A relief attribútum a megfelelő relief elemre van beállítva a border_effects szótárban, a border attribútum pedig 5 képpontra van állítva, hogy látható legyen a hatás

• 15. sor a .pack() metódussal helyezi el a keretet az ablakban. Az oldalsó kulcsszó megmondja, hogy hol helyezze el a szegélyt.

• A 16. és 17. sor létrehoz egy szöveges címke widgetet, amely megjeleníti a terep nevét, és elhelyezi a létrehozott keretben.

Példa.

```
    def submit():
        name = name_entry.get()
        age = age_entry.get()
        res.config(text=f"Név: {name}, Életkor: {age}")
    import tkinter as tk
    root = tk.Tk()
    input_frame = tk.Frame(root); input_frame.pack(padx=20,
    pady=20)
    name = tk.Label(input_frame, text="Név:")
    name.grid(row=0, column=0, padx=5, pady=5)
    name_entry = tk.Entry(input_frame)
    name_entry.grid(row=0, column=1, padx=5, pady=5)
    age = tk.Label(input_frame, text="Életkor:")
    age.grid(row=1, column=0, padx=5, pady=5)
    age_entry = tk.Entry(input_frame)
    age_entry.grid(row=1, column=1, padx=5, pady=5)
    but = tk.Button(input_frame, text="Küldés", command=submit)
    but.grid(row=2, columnspan=2, padx=5, pady=5)
    fra = tk.Frame(root); fra.pack(padx=20, pady=20)
    res = tk.Label(fra, text=""); res.pack()
```

```
root.mainloop()
```

VISSZA

10. Scale (Skála)

A skála célja, hogy a felhasználó kiválaszthasson egy értéket egy bizonyos tartományból. Külsőleg a skála egy vízszintes vagy függőleges csík jelölésekkel, amelyek mentén a felhasználó mozgathatja a csúszkát, és ezáltal kiválaszthat egy értéket:

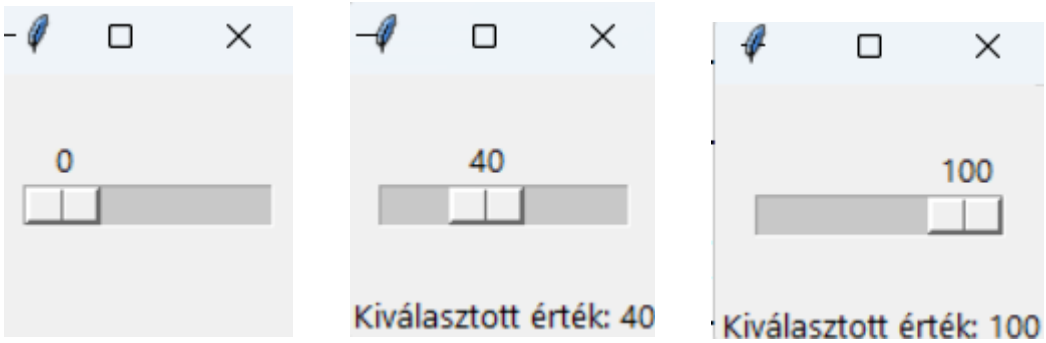
```
sca1 = Scale(fra3,orient=HORIZONTAL,length=300,
from_=0,to=100,tickinterval=10,resolution=5)
sca2 = Scale(root,orient=VERTICAL,length=400,
from_=1,to=2,tickinterval=0.1,resolution=0.1)
```

Tulajdonságok:
orient határozza meg a skála irányát;
length – skála hossza pixelben;
from and to – milyen értékkel kezdődik és mire végződik a skála (azaz az értéktartomány);
tickinterval – az az intervallum, amelyen belül a skála értékei megjelennek;
resolution – a szegmens minimális hossza, amellyel a felhasználó mozgathatja a csúszkát.

Példa.orient=tk.HORIZONTAL eset

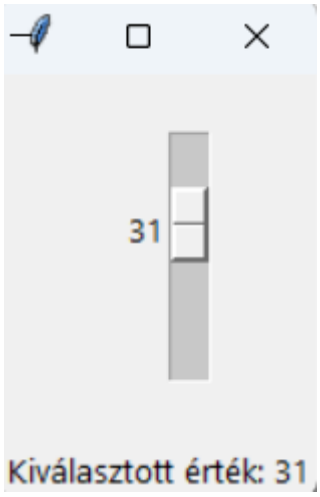
```
def show_value(value):
    value_label.config(text="Kiválasztott érték: " +
str(value))

import tkinter as tk
```

```
root = tk.Tk()
scale = tk.Scale(root, from_=0, to=100, orient=tk.HORIZONTAL,
command=show_value)
scale.pack(pady=20)
value_label = tk.Label(root, text="")
value_label.pack()
root.mainloop()
```



Példa. orient=tk.VERTICAL eset

```
def show_value(value):
    value_label.config(text="Kiválasztott érték:"+str(value))
import tkinter as tk
root = tk.Tk()
scale = tk.Scale(root, from_=0, to=100, orient=tk.VERTICAL,
command=show_value); scale.pack(pady=20)
value_label = tk.Label(root, text=""); value_label.pack()
root.mainloop()
```

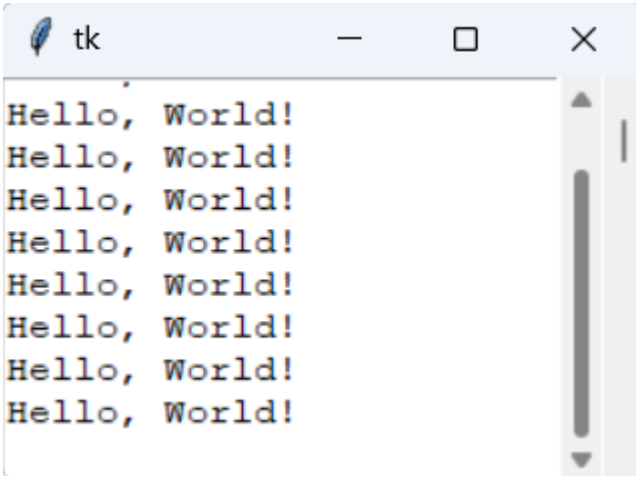


VISSZA

11. Scrollbar (görgetősáv)

Ez a widget lehetővé teszi egy másik widget (például szövegmező vagy lista) tartalmának görgetését. A görgetés lehet vízszintes vagy függőleges.

```
def insert_text():
    text_area.insert(tk.END, "Hello, World!\n")
import tkinter as tk
from tkinter import scrolledtext
root = tk.Tk()
scrollbar = tk.Scrollbar(root)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
text_area = scrolledtext.ScrolledText(root,
yscrollcommand=scrollbar.set)
text_area.pack(expand=True, fill='both')
scrollbar.config(command=text_area.yview)
insert_button = tk.Button(root, text="Szöveg hozzáadása",
command=insert_text); insert_button.pack()
root.mainloop()
```



VISSZA

12. Toplevel (legfelső szintű ablak)

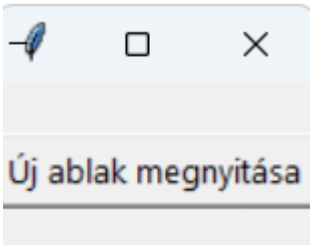
A Toplevel osztály segítségével gyermekablakok jönnek létre, amelyeken widgetek is elhelyezhetők. Megjegyzendő, hogy a főablak (vagy a szülő) bezárásakor a felső szintű ablak **Toplevel** is bezárul. Másrészt a gyermekablak bezárása nem zárja be a főablakot.

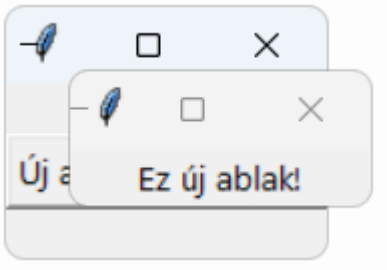
A **title** metódus az ablak címét adja meg. A **minsize** metódus konfigurálja a minimális ablakméretet (van egy **maxsize** metódus, amely meghatározza a maximális ablakméretet). Ha a **minsize** argumentumok megegyeznek a **maxsize** argumentumokkal, akkor a felhasználó nem tudja átméretezni az ablakot.

Példa.

```
def open_new_window():
    new_window = tk.Toplevel(root)
```

```
    new_window.title("Új ablak")
    label=tk.Label(new_window,text="Ez új ablak!")
    label.pack()
import tkinter as tk
root = tk.Tk(); root.title("FŐ ablak")
button = tk.Button(root, text="Új ablak megnyitása",
command=open_new_window); button.pack(pady=20)
root.mainloop()
```





VISSZA

13. Események

A GUI-alkalmazások jellemzően megvárnak valamilyen külső bevitelt (egérkattintások, billentyűleütések), majd végrehajtják a programozó által tervezett műveletet. A grafikus komponensre gyakorolt külső hatást **eseménynek** nevezzük. Több esemény létezik. Egyelőre csak kétféle eseményt fogunk használni:

- **bal egérgomb**
- **Enter billentyű**

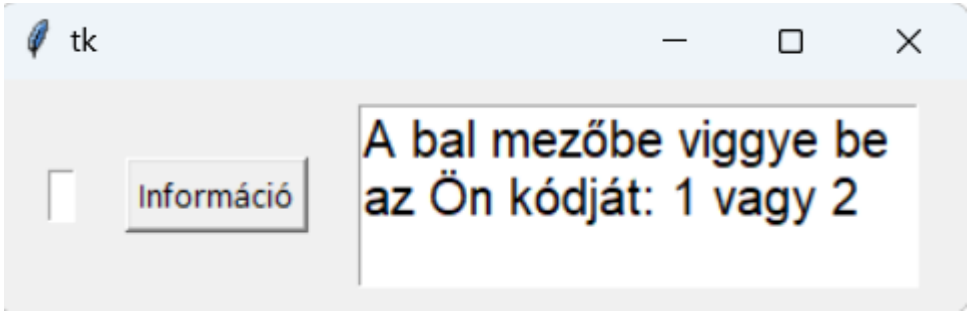
lenyomásával.

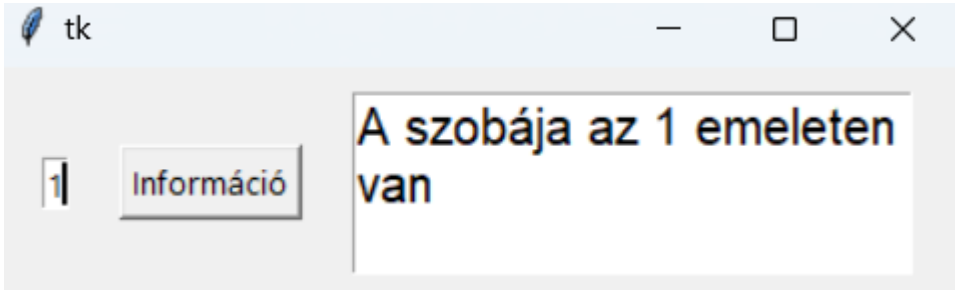
Egy widget eseményének és függvényének összerendelésének egyik módja a **bind** metódus használata.

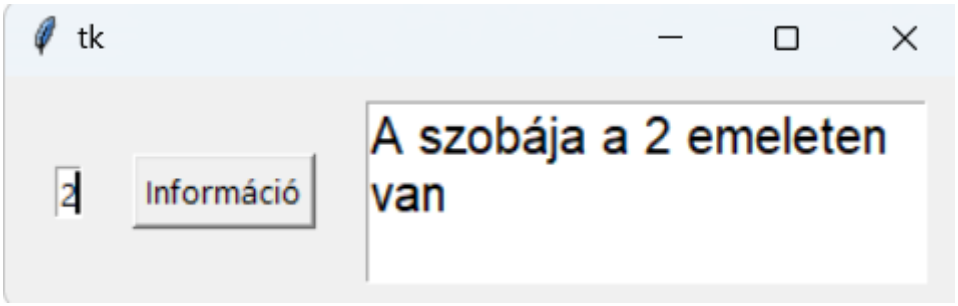
Példa.

```
def output(event):
    s = ent.get()
    if s == "1":
        tex.delete(1.0,END)
        tex.insert(END,"A szobája az 1 emeleten van")
    elif s == "2":
```

```
            tex.delete(1.0,END)
            tex.insert(END,"A szobája a 2 emeleten van")
        else:
            tex.delete(1.0,END)
            tex.insert(END,"A bal mezőbe viggye be az Ön kódját:
1 vagy 2 ")
from tkinter import *
root = Tk()
ent = Entry(root,width=1)
but = Button(root,text="Információ")
tex = Text(root,width=20,height=3,font="12",wrap=WORD)
ent.grid(row=0,column=0,padx=20)
but.grid(row=0,column=1)
tex.grid(row=0,column=2,padx=20,pady=10)
but.bind("<Button-1>",output)
root.mainloop()
```







Nézzük a kódot. Három widgetet hozunk létre: egy egysoros szövegmezőt, egy gombot és egy többsoros szövegmezőt. A felhasználónak be kell írnia valamit az első mezőbe, majd kattintson a gombra, és a második mezőben választ kap.

A **grid** metódusok helyezik el a widgeteket. A **padx** és a **pady** tulajdonságok határozzák meg a widgettől a keret (vagy cella) széléig terjedő pixelek számát az x és y tengely mentén.

A Button bal egérgomb megnyomása az output függvényhez kapcsolódik. Ez a három összetevő (widget, esemény és függvény) a bind metódussal van összekötve. Ebben az esetben, ha bal egérgombbal kattint a **but** gombra, az **output** függvény meghívódik.

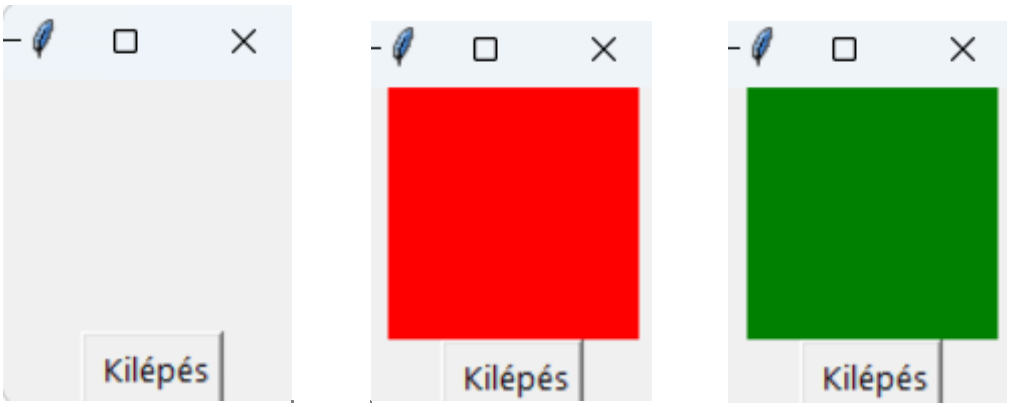
Ha a felhasználó a bal egérgombbal rákattint a gombra, akkor az output függvény végrehajtásra kerül (más esetben nem). Ez a függvény egy második szövegmezőben jeleníti meg az információkat. Hogy pontosan milyen információkat adott meg, az attól függ, hogy a felhasználó mit írt be az első szövegmezőbe. Egy esemény argumentumként kerül átadásra a függvénynek.

Az if-elif-else ágakon belül a delete és insert metódusokat használunk. Az első eltávolítja a karaktereket a szövegmezőből, a második beszúrja azokat. 1.0 - az első sort, az első karaktert jelöli (a karakterszámozás nullától kezdődik).

Példa.

```
li = ["red","green"]
def color(event):
    fra.configure(bg=li[0])
    li[0],li[1] = li[1],li[0]
def outgo(event):
    root.destroy()
from tkinter import *
root = Tk()
```

```
fra = Frame(root,width=100,height=100);
but = Button(root,text="Kilépés")
fra.pack()
but.pack()
root.bind("<Return>",color)  # <Return> = Enter
but.bind("<Button-1>",outgo)
root.mainloop()
```



Itt két widget jön létre: egy keret és egy gomb.

Az alkalmazás két eseményre reagál: az **Enter** billentyűre a főablakban és a bal egérgombbal kattintson a but Button-ra. Az első esetben a színváltoztatást hívja, a másodikban – „kilépés”.

A **color** függvény megváltoztatja a keret (fra) háttérszínét (bg) a **configure** metódussal, amely a widget tulajdonságainak megváltoztatására szolgál a szkript végrehajtása során. A bg opció értéke a lista első elemére van állítva. Ezután a lista két eleme felcserélődik, így az Enter legközelebbi megnyomásakor a keret színe ismét megváltozik.

A **outgo** függvény meghívja a **destroy** metódust a főablakban. Ennek a metódusnak a célja az ablak bezárása.

VISSZA

13.1. Esemény tipusok

A **bind** metódus meghívásakor az esemény első argumentumként kerül átadásra.

Az esemény neve idézőjelbe, valamint < és > közé kerül. Egy esemény leírása lefoglalt kulcsszósorozatok használatával történik.

Egér események

- <Button-1> - bal egérkattintás
- <Button-2> - kattintás a középső egérgombbal
- <Button-3> - jobb egérkattintás
- <Double-Button-1> - dupla kattintás a bal egérgombbal
- <Motion> - az egér mozgása
- <Return> - Enter billentyű, stb.

Példa.

```
from tkinter import *
def b1(event):
    root.title("bal egérkattintás ")
def b3(event):
    root.title("jobb egérkattintáss")
def move(event):
    root.title("Egér mozgatása")

root = Tk()
root.minsize(width = 400, height=150)
root.bind('<Button-1>',b1)
root.bind('<Button-3>',b3)
root.bind('<Motion>',move)
root.mainloop()
```

Ebben a programban a főablak címe attól függően változik, hogy mozog-e az egér, vagy a bal vagy jobb egérgombot kattintottuk.

Billentyűzet-események

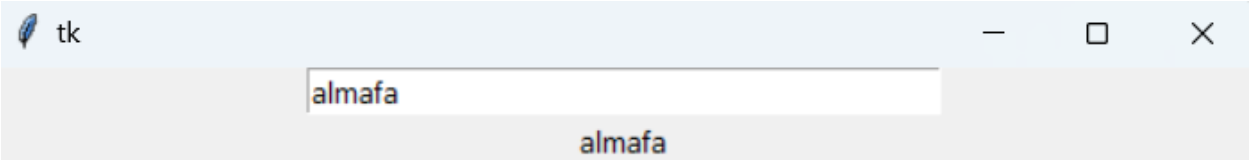
A betűbillentyűk szögletes zárójelek nélkül írhatók (pl. „L”).
Vannak speciális fenntartott szavak a nem alfabetikus billentyűkhöz

- <Return> - az Enter billentyű lenyomása;
- <space> - szóköz;

stb.
A billentyűpárokat kötőjellel kell írni.

```
from tkinter import *

def exit_(event):

    root.destroy()

def caption(event):

    t = ent.get()
```

```
        lbl.configure(text = t)
    root = Tk()
    ent = Entry(root, width = 40); ent.pack()
    lbl = Label(root, width = 80); lbl.pack()
    ent.bind('<Return>',caption)
    root.bind('<Control-z>',exit_)
    root.mainloop()
```



Ha megnyomja az **Enter** billentyűt egy szöveges sztringen belül (ent), meghívódik a **caption** függvény, amely karaktereket helyez el a szövegből (ent) a lbl címkébe. A **Ctrl + z** billentyűkombináció megnyomásával bezárja a főablakot.

VISSZA

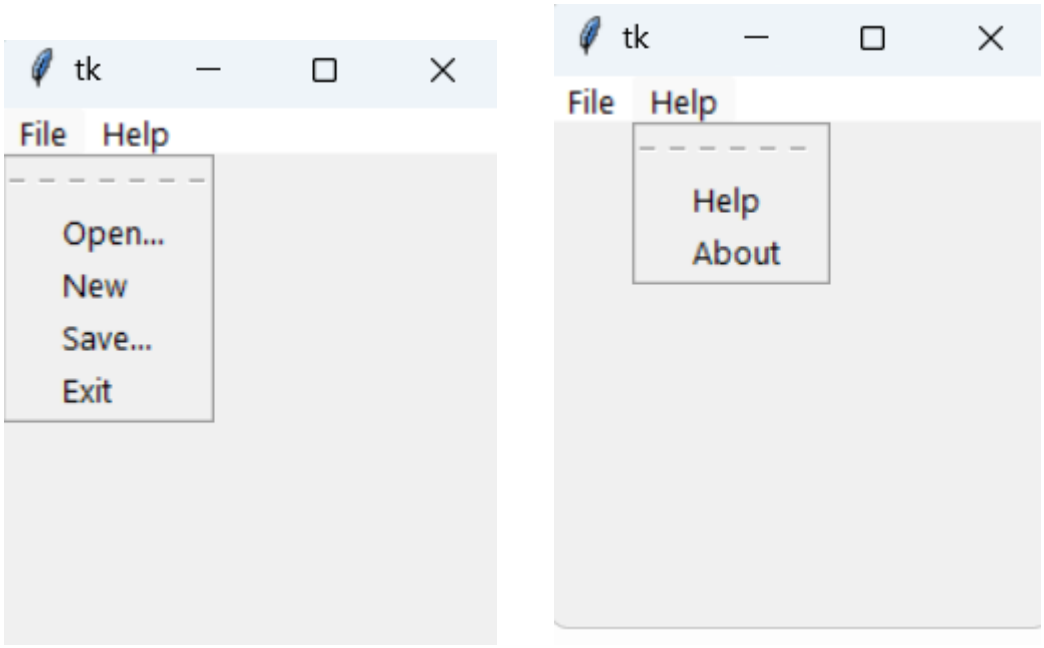
14. Menu (menü)

A menü egy olyan objektum, amely számos felhasználói alkalmazásban megtalálható. A címsor alatt található, és a szavak alatti legördülő listákból áll; minden ilyen lista tartalmazhat egy másik beágyazott listát. Minden listaelem egy parancsot jelent, amely műveletet indít el vagy párbeszédpanelt nyit meg.

Példa. Menü létrehozása a Tkinterben.

```
from tkinter import *
root = Tk()
m = Menu(root)
root.config(menu=m)
fm = Menu(m)
m.add_cascade(label="File",menu=fm)
fm.add_command(label="Open...")
fm.add_command(label="New")
fm.add_command(label="Save...")
fm.add_command(label="Exit")
```

```
hm = Menu(m)
m.add_cascade(label="Help",menu=hm)
hm.add_command(label="Help")
hm.add_command(label="About")
root.mainloop()
```



Az **add_cascade** metódus egy új elemet ad a menühöz, amely a menüopció értékeként van megadva. Az **add_command** metódus egy új parancsot ad egy menüelemhez. Ennek a metódusnak az egyik opciója- command - összekapcsolja ezt a parancsot egy kezelő funkcióval. Létrehozhat egy almenüt. Ehhez hozzon létre egy másik menüt, és az add_cascade segítségével kösse a szülőelemhez.

```
nfm = Menu(fm)
fm.add_cascade(label="Import",menu=nfm)
nfm.add_command(label="Image")
nfm.add_command(label="Text")
```

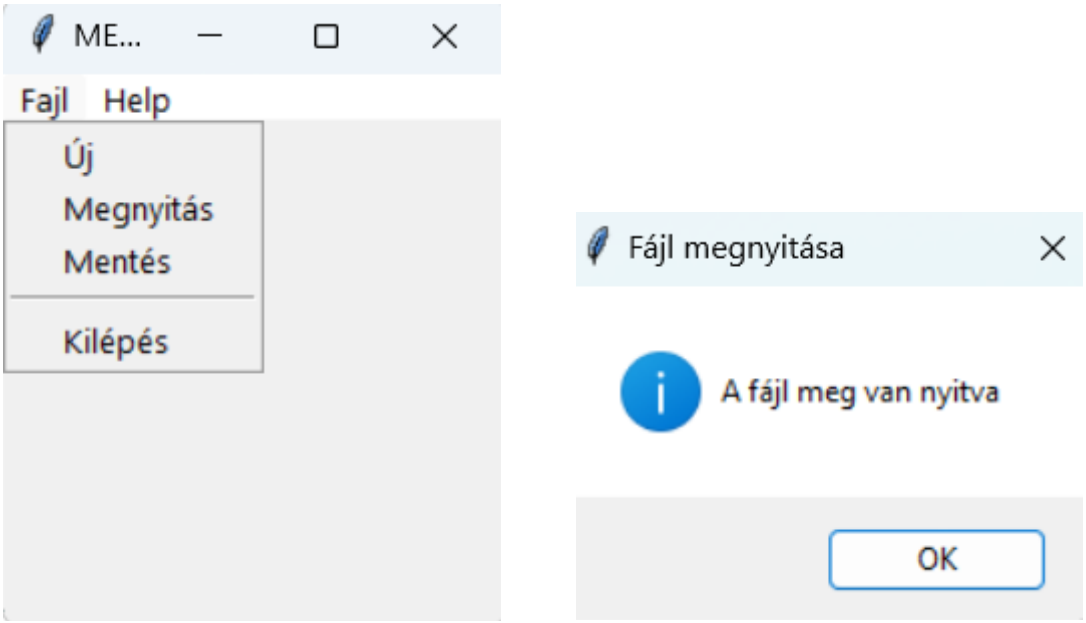
VISSZA

Függvények összekapcsolása a menükkel

Minden menüparancshoz általában hozzá kell rendelni a saját függvényét, amely bizonyos műveleteket (kifejezéseket) hajt végre. A kommunikáció az **add_command** metódus command opciójával történik. Először a kezelő függvényt kell definiálni.
A függvények akkor hívódnak meg, amikor a felhasználó bal egérgombbal kattint a megfelelő almenüelemekre.

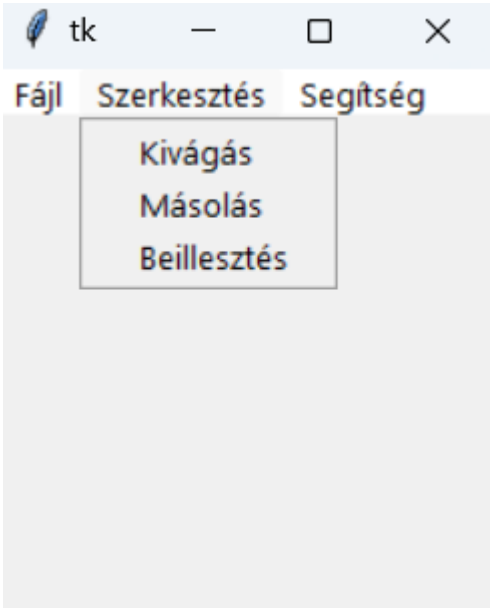
Példa. Menü messagebox alkalmazásával.

```
def new_file():
    messagebox.showinfo("Fájl létrehozása", "A fájl
létrejött")
def open_file():
    messagebox.showinfo("Fájl megnyitása", "A fájl meg van
nyitva")
def save_file():
    messagebox.showinfo("Fájl mentése", "A fájl mentve van")
def exit_app():
    root.destroy()
import tkinter as tk
from tkinter import messagebox
root = tk.Tk()
root.title("MENÜ")
menu_bar = tk.Menu(root)
file_menu = tk.Menu(menu_bar, tearoff=0)
file_menu.add_command(label="Új", command=new_file)
file_menu.add_command(label="Megnyitás", command=open_file)
file_menu.add_command(label="Mentés", command=save_file)
file_menu.add_separator()
file_menu.add_command(label="Kilépés", command=exit_app)
menu_bar.add_cascade(label="Fajl", menu=file_menu)
help_menu = tk.Menu(menu_bar, tearoff=0)
help_menu.add_command(label="A programról", command=lambda:
messagebox.showinfo("A programról","Menü példája a Tkinter-
ben"))
menu_bar.add_cascade(label="Help", menu=help_menu)
root.config(menu=menu_bar)
root.mainloop()
```



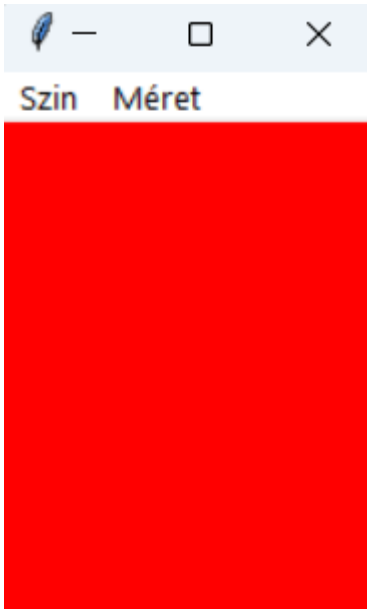
Példa. Még egy tipikus menü szerkesztése.

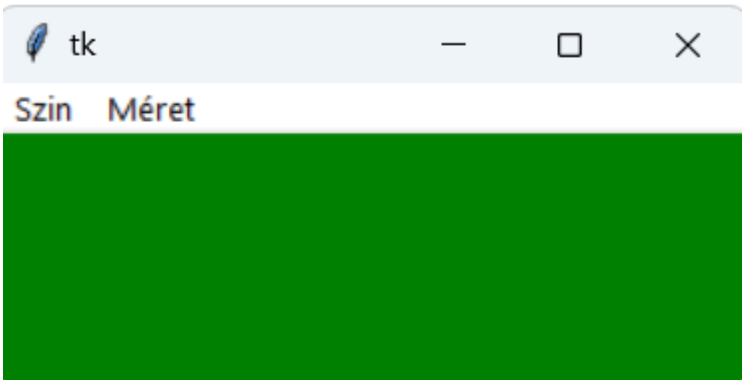
```
from tkinter import Tk
from tkinter import Menu

def f():
    print('---------')

root = Tk()

def hello():
    print("Helló")

menusav = Menu(root)

#Legördülő menü
fajlmenu = Menu(menusav, tearoff=0)
fajlmenu.add_command(label="Megnyitás", command=f())
fajlmenu.add_command(label="Mentés", command=f())
fajlmenu.add_separator()
fajlmenu.add_command(label="Kilépés", command=root.quit)
menusav.add_cascade(label="Fájl", menu=fajlmenu)

editmenu = Menu(menusav, tearoff=0)
editmenu.add_command(label="Kivágás", command=hello)
editmenu.add_command(label="Másolás", command=hello)
editmenu.add_command(label="Beillesztés", command=hello)
menusav.add_cascade(label="Szerkesztés", menu=editmenu)

helpmenu = Menu(menusav, tearoff=0)
helpmenu.add_command(label="Névjegy", command=hello)
menusav.add_cascade(label="Segítség", menu=helpmenu)

root.config(menu=menusav)
root.mainloop()
```

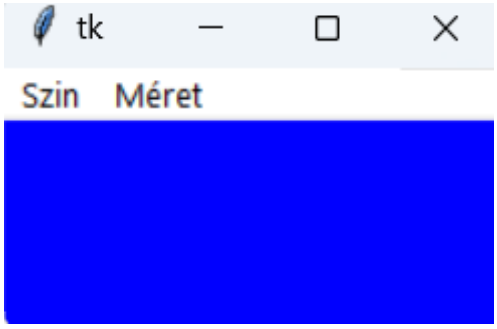


Példa.
Egy alkalmazás, amelyben a menü két elemet tartalmazz: Szín és Méret.
A **Szín** elemnek három parancsot kell tartalmaznia (piros, zöld és kék), színváltó keretek a főablakon.
A **Méret** elemnek tartalmaznia kell két parancsot (200x80 és 150x200), amelyek megváltoztatják a keret méretét.

```
from tkinter import *
root = Tk()

def colorR():
    fra.config(bg="Red")
def colorG():
    fra.config(bg="Green")
def colorB():
    fra.config(bg="Blue")
def square():
    fra.config(width=200)
    fra.config(height=80)
def rectangle():
    fra.config(width=150)
    fra.config(height=200)

fra = Frame(root,width=300,height=100,bg="grey")
fra.pack()
m = Menu(root)
root.config(menu=m)

cm = Menu(m)
m.add_cascade(label="Color",menu=cm)
cm.add_command(label="Red",command=colorR)
cm.add_command(label="Green",command=colorG)
cm.add_command(label="Blue",command=colorB)

sm = Menu(m)
m.add_cascade(label="Size",menu=sm)
sm.add_command(label="200x80",command=square)
sm.add_command(label="150x200",command=rectangle)
root.mainloop()
```







VISSZA

15. Messagebox (üzenetdobozok)

Az üzenetdobozok létrehozására alkalamzzák a **tkMessageBox** modult. A tkMessageBox modul Pythonban a Tkinter csomag része, és üzenetablakokat biztosít különböző üzenetek megjelenítésére a felhasználó számára, például információt, figyelmeztetések vagy hibaüzenetek formájában. A modulban számos függvény található, amelyek segítségével különböző típusú üzenetablakokat jeleníthetünk meg. Ezek a függvények nagyon hasznosak, ha interaktív üzeneteket kell megjeleníteni a felhasználónak Python Tkinter alkalmazásokban.

Néhány gyakran használt függvény a tkMessageBox modulban:

1. **showinfo(title, message)**: Ezzel a függvénnyel információs ablak jeleníthető meg a felhasználónak.

2. **showwarning(title, message)**: Ezzel a függvénnyel figyelmeztető ablak jeleníthető meg a felhasználónak.

3. **showerror(title, message):** Ezzel a függvénnyel hibaablak jeleníthető meg a felhasználónak.

4. **askquestion(title, message):** Ezzel a függvénnyel kérdező ablak jeleníthető meg a felhasználónak, ahol a felhasználó igen vagy nem választhat.

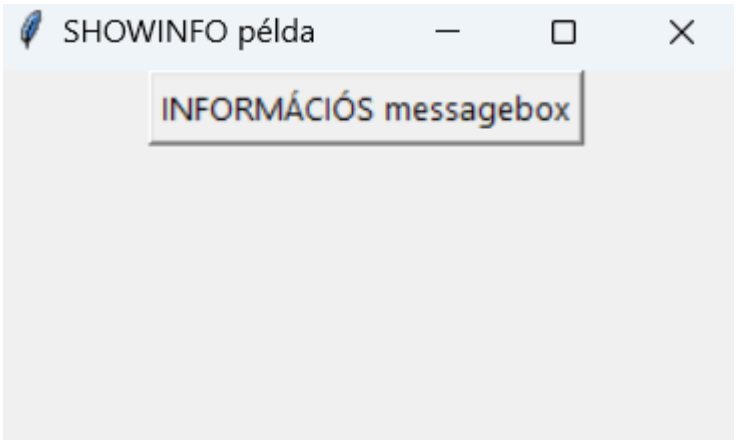
5. **askyesno(title, message):** Ezzel a függvénnyel igen-nem ablak jeleníthető meg a felhasználónak.

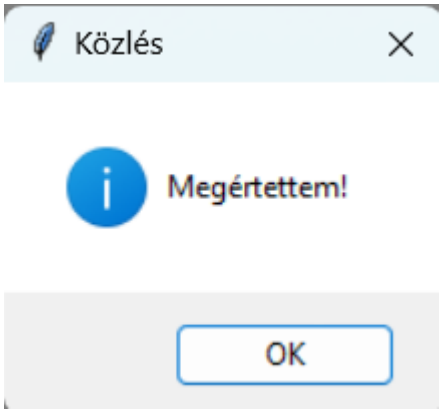
6. **askokcancel(title, message):** Ezzel a függvénnyel OK-Mégse ablak jeleníthető meg a felhasználónak.

A függvények paraméterei: **title** a cím, a **message** az üzenet szövege. title és message kívül alkalmazhatók opcionális paraméterek is.

Példa.

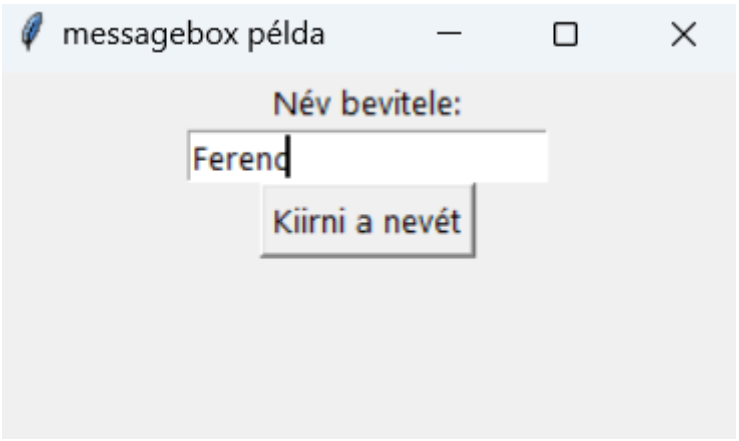
```
def box():
   messagebox.showinfo("Közlés", "Megértettem!")

import tkinter
from tkinter import messagebox
root = tkinter.Tk()
root.title("SHOWINFO példa")
root.geometry("300x150") #

but = tkinter.Button(root, text = "INFORMÁCIÓS messagebox",
command = box)
but.pack()
root.mainloop()
```



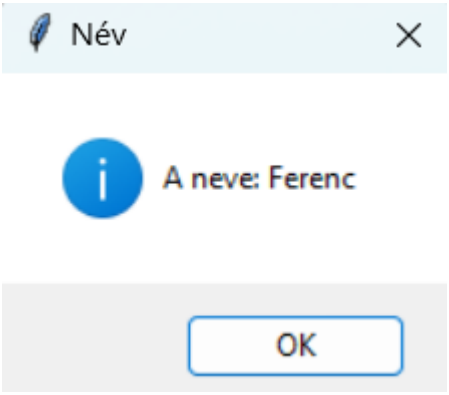


Példa.

```
import tkinter as tk
from tkinter import messagebox
def show_name():
    name = entry.get()
    messagebox.showinfo("Név", "A neve: " + name)

root = tk.Tk()
root.title("messagebox példa")
root.geometry("300x150") #
label = tk.Label( text="Neve bevitele:"); label.pack()
entry = tk.Entry(); entry.pack()
button = tk.Button( text="Kiirni a nevét", command=show_name)
button.pack()
abel = tk.Label( text="")
abel.pack()
root.mainloop()
```

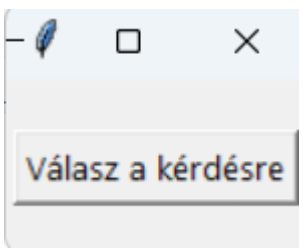


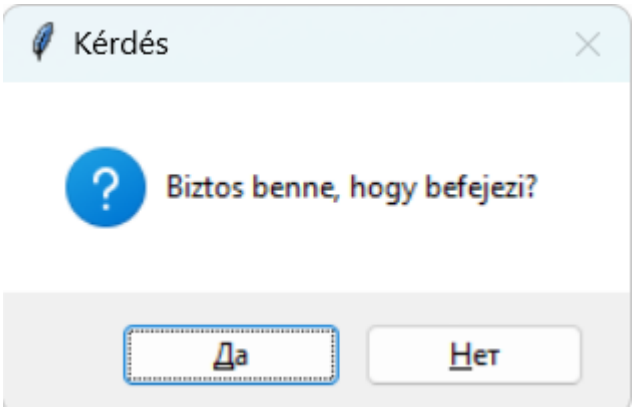
Eredmény:



Példa.

```
def ask():
    result = messagebox.askyesno("Kérdés", "Biztos benne, hogy befejezi?")
    if result:
        print("A válasz: igen.")
    else:
        print("A válasz: nem.")
import tkinter as tk
from tkinter import messagebox
root = tk.Tk()
root.title("askyesno Példa")
button = tk.Button(root, text="Válasz a kérdésre", command=ask)
button.pack(pady=20)
root.mainloop()
```





VISSZA

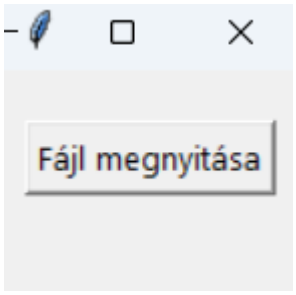
16. filedialog (fájlválasztó dialógus)

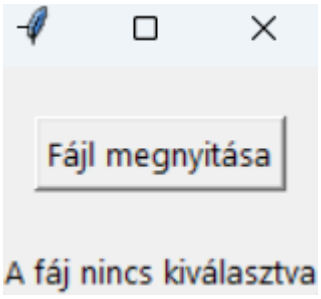
A tkinter.filedialog modul a Tkinter csomag része, és lehetővé teszi az ablakos fájlválasztó dialógusok létrehozását. Ezek a dialógusok által a felhasználó böngészhet a fájlok között, választhat ki egyet vagy többet, vagy adja meg fájlokat és mappákat az alkalmazás számára. Fontosabb műveletek:

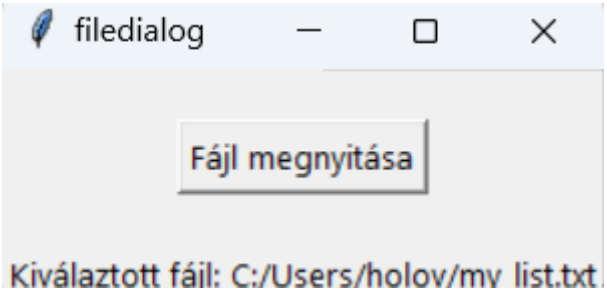
- **Fájl megnyitása.** Ha az alkalmazásnak fájlt kell megnyitnia, a *filedialog.askopenfilename()* függvény segítségével a felhasználó kiválaszthat egyet a rendszerből.
- **Fájl mentése.** Ha a felhasználónak egy fájlt kell mentenie az alkalmazásból, a *filedialog.asksaveasfilename()* függvény segítségével megadhatja a menteni kívánt fájl nevét és helyét.
- **Mappa kiválasztása.** Ha az alkalmazásnak egy mappát kell kiválasztania, a *filedialog.askdirectory()* függvény segítségével kiválaszthat egy mappát a rendszerből.
- K**önyvtár kiválasztása.** *askdirectory()*függvényt alkalmazzuk.

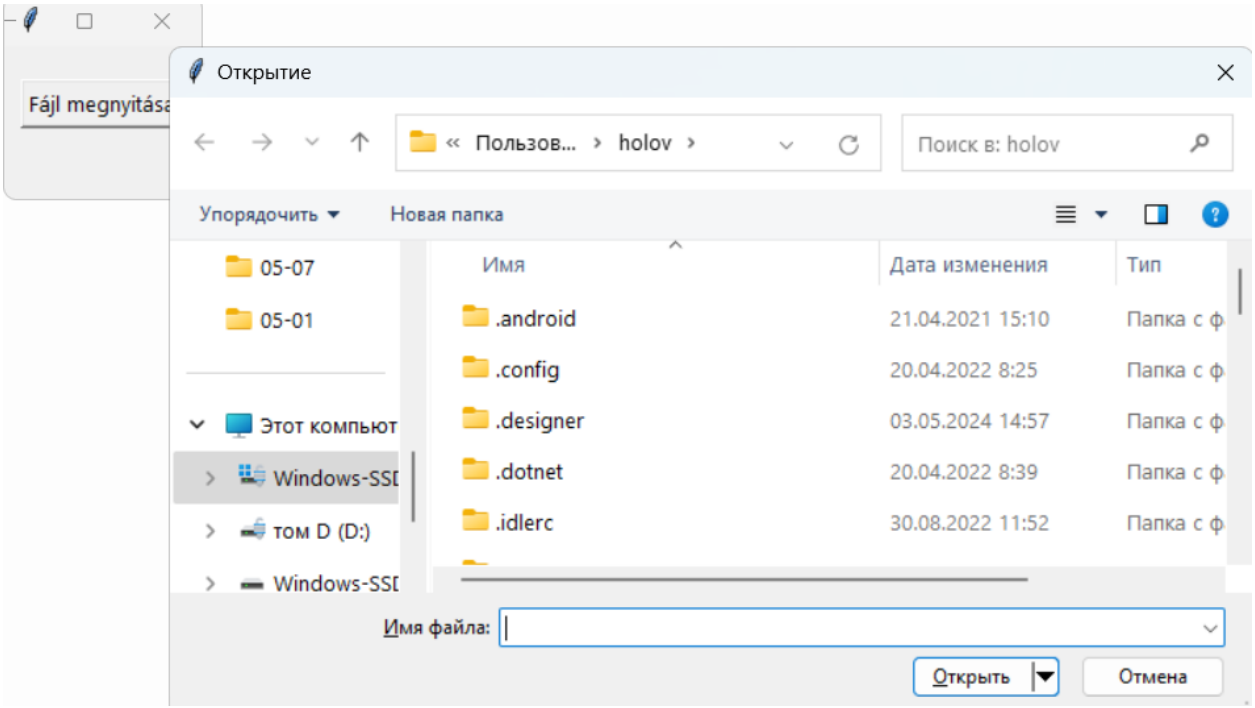
Példa.

```
import tkinter as tk
from tkinter import filedialog

def open_file_dialog():
    file_path = filedialog.askopenfilename()
    if file_path:
        info_label.config(text="Kiválaztott fájl: " +
file_path)
    else:
        info_label.config(text="A fáj nincs kiválasztva")

root = tk.Tk()
root.title("filedialog")

open_button = tk.Button(root, text="Fájl megnyitása",
command=open_file_dialog)
open_button.pack(pady=20)
info_label = tk.Label(root, text="")
info_label.pack()

root.mainloop()
```









VISSZA

17. Geometry Manager. Az alkalmazás elrendezésének testreszabása

A "Geometry Manager" a Tkinterben az a mechanizmus, amely felelős a widgetek elhelyezéséért az alkalmazás ablakában. A Tkinternek három fő geometriakezelője, metódusa van:

- **pack()**, a widgeteket blokkokba rendezi,

- **grid()**, a widgeteket táblázatba helyezi el,
- **place()**, megadott pozícióban helyezzük el a widgeteket.

pack(): Ez a geometriakezelő lehetővé teszi, hogy widgeteket csomagoljon egy tárolóba (általában egy frame-be, keretbe) a négy irány egyikében: fent, lent, balra vagy jobbra. Automatikusan kezeli a widgetek méretét és pozícióját.

grid(): Ez a geometriakezelő lehetővé teszi a widgetek elrendezését egy **rácsba (sorok és oszlopok)** egy szülőtárolón. A widgetek pozícióját és méretét sor- és oszlopkoordináták segítségével szabályozhatja.

place(): egy olyan metódus a Tkinterben, amely lehetővé teszi widgetek manuális elhelyezését a programablakban úgy, hogy minden widgethez abszolút koordinátákat (x, y) és méretet (szélesség és magasság) ad meg. Más geometriakezelőkkel ellentétben, mint például a pack() és grid(), amelyek automatikusan elosztják a widgeteket egy ablakban, a place() pontosabban szabályozza a widgetek elhelyezését.

Egy alkalmazásban minden ablak és keret (frame) csak egy geometriakezelőt használhat. A különböző keretek pedig különböző geometriakezelőket használhatnak, még akkor is, ha egy másik geometriakezelő által használt ablakhoz vagy kerethez vannak hozzárendelve.

17.1. Pack() geometriakezelő.

Automatikus elhelyezés egymás alatt

A pack() menedzser arra szolgál, hogy egy csomagoló algoritmus segítségével meghatározott sorrendben widgeteket helyezzen el egy keretben vagy alkalmazás ablakában. Egy adott widget esetében a csomagolási algoritmusnak két fő szakasza van:

• Egy parcellának nevezett téglalap alakú terület kiszámítása. A terület mérete a widgethez illeszkedik, és az ablak fennmaradó szélessége (vagy magassága) üres hellyel van kitöltve;

• A widget középre helyezése a területen, hacsak nincs megadva más hely az ablakban.

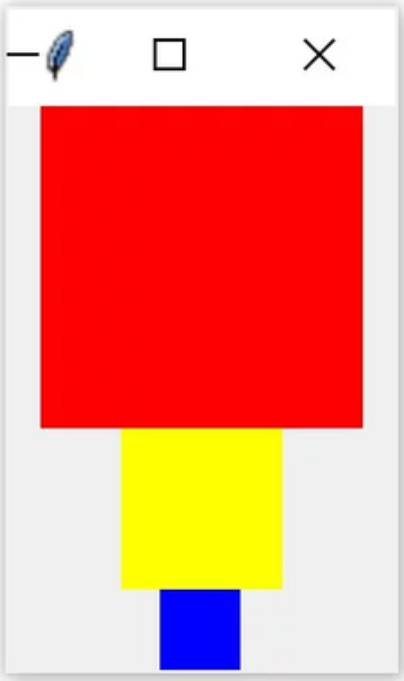
Példa.

Három szöveges címkemodult kell elhelyezni egy keretben a pack() kezelővel.

```
import tkinter as tk
window = tk.Tk()
```

```
frame1 = tk.Frame(master=window, width=100, height=100,
bg="red"); frame1.pack()

frame2 = tk.Frame(master=window, width=50, height=50,
bg="yellow"); frame2.pack()

frame3 = tk.Frame(master=window, width=25, height=25,
bg="blue"); frame3.pack()

window.mainloop()
```

Alapértelmezés szerint a **pack()** minden egyes keretet egymás alá helyez abban a sorrendben, ahogyan az alkalmazás ablakába kerülnek.



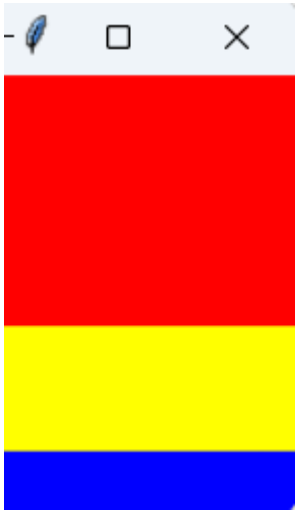
Minden keret a legmagasabb elérhető pozícióban van. A piros az ablak tetején, a sárga közvetlenül a piros alatt, a kék pedig a sárga alatt van. Minden keretben három láthatatlan terület található. Az egyes szakaszok szélessége egybeesik az ablak szélességével, a magassága pedig a belső keret magasságával. Minden keret az ablak közepén van.

A pack() **fill** argumentuma azt határozza meg, hogy a keretek milyen irányban legyenek kitöltve. A választható opciók a következők: **tk.X** vízszintes irányhoz, **tk.Y** függőleges irányhoz és **tk.BOTH** mindkettőhöz. A következő kód az ablak három kerettel történő vízszintes van kitöltve:

```
import tkinter as tk
window = tk.Tk()
frame1 = tk.Frame(master=window, height=100, bg="red")
frame1.pack(fill=tk.X)
frame2 = tk.Frame(master=window, height=50, bg="yellow")
frame2.pack(fill=tk.X)
frame3 = tk.Frame(master=window, height=25, bg="blue")
frame3.pack(fill=tk.X)
window.mainloop()
```

A keret **width** szélessége nincs beállítva. A szélesség argumentumra már nincs szükség, mivel minden keret a pack() függvényt használja a vízszintes kitöltéshez, felülírva minden egyedi szélességbeállítást.

A fenti szkript kimenete így fog kinézni:



A pack() metódus **side** argumentuma határozza meg, hogy az ablak melyik oldalára kerüljön a widget. Elérhető opciók:

- tk.TOP felül;
- tk.BOTTOM alsó;
- tk.LEFT balra;
- tk.RIGHT a jobb oldalon.

Abban az esetben, ha az oldal argumentum nincs megadva, akkor a pack() metódus automatikusan a tk.TOP-ot használja, és új widgeteket helyez el az ablak tetején, vagy az ablak legtetején, amelyet még nem foglalt el widget. Például a

következő szkript három keretet helyez el egymás mellé balról jobbra, és mindegyik keretet kibontja, hogy függőlegesen kitöltse az ablakot:

```
import tkinter as tk
window = tk.Tk()
frame1 = tk.Frame(master=window, width=200, height=100,
bg="red")
frame1.pack(fill=tk.Y, side=tk.LEFT)
frame2 = tk.Frame(master=window, width=100, bg="yellow")
frame2.pack(fill=tk.Y, side=tk.LEFT)
frame3 = tk.Frame(master=window, width=50, bg="blue")
frame3.pack(fill=tk.Y, side=tk.LEFT)
window.mainloop()
```

Most meg kell adnia a height (magasság) argumentumot legalább egy keretben, hogy az ablaknak egy adott magasságot adjon.

A létrehozott ablak így néz ki:

Ha beállítja a fill=tk.X beállítást, a keretek alkalmazkodnak az ablak vízszintes átméretezésekor, és ha megadja a fill=tk.Y értéket, akkor alkalmazkodnak az ablak függőleges átméretezésekor.

Ahhoz, hogy az elrendezés valóban érzékeny legyen, beállíthatja a keretek kezdeti méretét a width és height attribútumok használatával. Ezután a pack() metódusból származó fill argumentum értékét tk.BOTH-ra kell állítani, és az expand argumentum értékét is igazra kell állítani:

```
import tkinter as tk
window = tk.Tk()
frame1 = tk.Frame(master=window, width=200, height=100,
bg="red")
frame1.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
frame2 = tk.Frame(master=window, width=100, bg="yellow")
frame2.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
frame3 = tk.Frame(master=window, width=50, bg="blue")
frame3.pack(fill=tk.BOTH, side=tk.LEFT, expand=True)
window.mainloop()
```

A fenti kód futtatásakor az eredmény egy ablak lesz, amely kezdetben az előző példához hasonlóan néz ki. A különbség az, hogy mostantól bármilyen irányba átméretezheti az ablakot, és a keretek ennek megfelelően alkalmazkodnak az ablak átméretezésekor:

VISSZA

17.2. Place() geometriakezelő.

Elhelyezés adott pozícióban

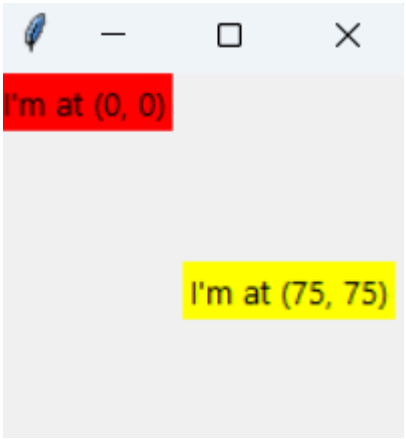
A widget ablakban vagy keretben való pontos helyének szabályozásához a place() kezelőt használjuk. Meg kell adnia két kulcsargumentumot x és y, amelyek meghatározzák az x és y koordinátákat a widget bal felső sarkában. Az x és y argumentumokat képpontokban mérjük, nem szövegegységekben.

Ne feledje, hogy az origó (ahol x és y 0) a keret vagy az ablak bal felső sarka. Tehát a place() metódusban az y argumentumot az ablak tetején lévő képpontok számának tekinthetjük, az x argumentumot pedig az ablak bal oldalán lévő pixelek számának.

Példa a place() geometriakezelő működésére:

```
import tkinter as tk

window = tk.Tk()

frame = tk.Frame(master=window, width=165, height=150)

frame.pack()

label1 = tk.Label(master=frame, text="I'm at (0, 0)",
bg="red")

label1.place(x=0, y=0)

label2 = tk.Label(master=frame, text="I'm at (75, 75)",
bg="yellow")

label2.place(x=75, y=75)

window.mainloop()
```

Eredmény:



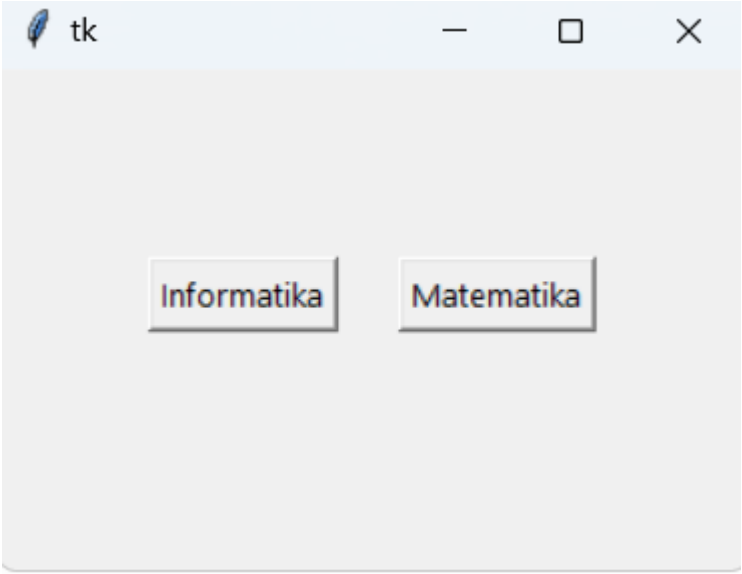
A geometriakezelő place() nem túl gyakran használatos. Két jelentős hátránya van:

• A place() metódus használatával az elrendezés nehezen testreszabható, különösen olyan esetekben, amikor az alkalmazások sok wimetódusdgettel rendelkeznek;

• A place() kezelővel létrehozott elrendezések nem változnak a főablak átméretezésekor.

Példa.

```
from tkinter import Tk
from tkinter import Frame
from tkinter import Button
root = Tk()
keret = Frame(root, width=300,height=200); keret.pack()
but1 = Button(root, text="Informatika"); but1.place(x=60,
y=75)
but2 = Button(root, text="Matematika");  but2.place(x=160,
y=75)
root.mainloop()
```



VISSZA

17.3.grid() geometriakezelő

Táblázatos elhelyezés

A Tkinter legnépszerűbb geometriakezelője a grid(). Sokkal könnyebben használható, mint a pack().

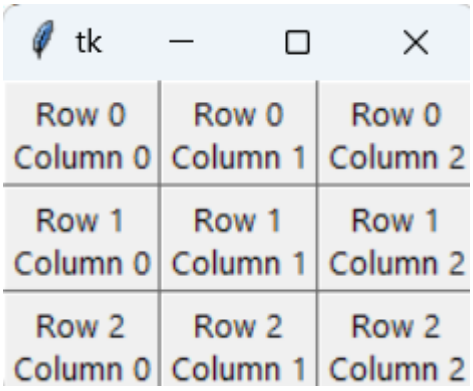
A grid() kezelő úgy működik, hogy egy ablakot vagy keretet sorokra és oszlopokra oszt (rácsokra). A widget helyét a .grid() metódus meghívásával és a sor- és oszlopindexek sor- és oszlopkulcs-argumentumainak átadásával adhatja meg.

A sor- és oszlopindexek 0-val kezdődnek, tehát az 1. sorindex és a 2. oszlopindex azt mondja a .grid() metódusnak, hogy a widgetet a második sor harmadik oszlopába helyezze.

A következő szkript 3 × 3-as keretekből álló rácsot hoz létre, amely szöveges címkéket tartalmaz.

```
import tkinter as tk
window = tk.Tk()
for i in range(:
    for j in range(3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1 )
        frame.grid(row=i, column=j)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn
{j}")
        label.pack()
window.mainloop()
```

Eredmény:



A grid() metódus minden keretobjektumban meghívásra kerül, de a geometriakezelő alkalmazásra kerül az ablakra. Hasonlóképpen, az egyes keretek elhelyezését a pack() geometriakezelő vezérli.

Az előző példában a keretek egymás mellett helyezkednek el. Ha helyet szeretne adni az egyes keretek körül, beállíthat egy kitöltést a rács minden egyes cellájához.

A padding vagy a padding egyszerűen egy üres tér, amely körülveszi a widgetet, és vizuálisan elválasztja a tartalmától.

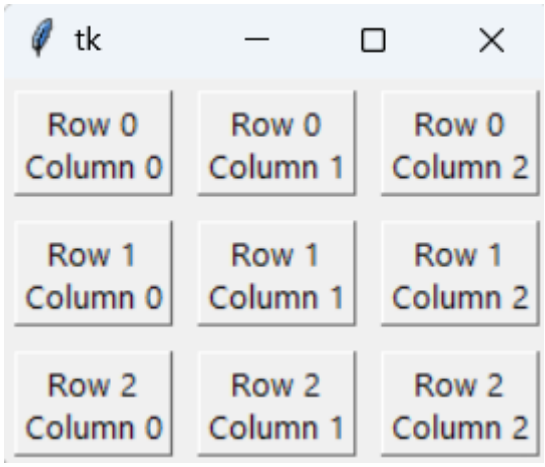
Kétféle behúzás létezik - külső és belső. A külső párnázás helyet ad a rácscella körül. A vezérlés a grid() metódus két kulcsfontosságú argumentumán keresztül történik:

- a **padx** behúzást ad vízszintes irányban;
- a **pady** függőleges irányban párnázást ad.

A padx és a pady argumentumokat képpontokban mérik, nem szövegegységekben, így mindkettőjük azonos értékre állításával mindkét irányban azonos mennyiségű kitöltés jön létre.

```
import tkinter as tk
window = tk.Tk()
for i in range(3):
    for j in range(3):
        frame = tk.Frame(
            master=window,
            relief=tk.RAISED,
            borderwidth=1 )
        frame.grid(row=i, column=j, padx=5, pady=5)
        label = tk.Label(master=frame, text=f"Row {i}\nColumn
{j}")
        label.pack()
window.mainloop()
```

Eredmény:



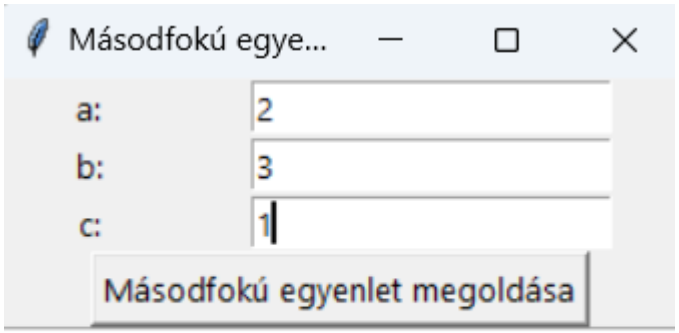
VISSZA

Melléklet

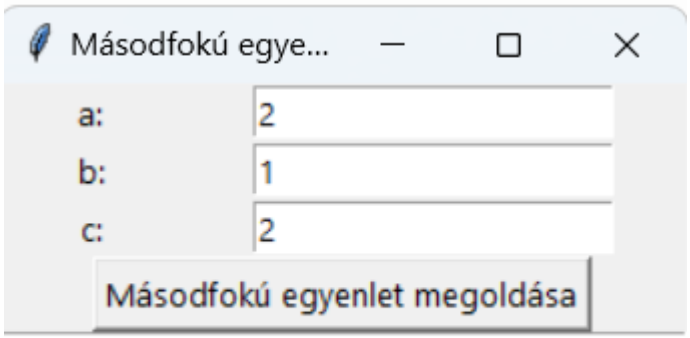
A) Másodfokú egyenlet megoldása

```
import tkinter as tk
def gyok2():
        a = float(entry_a.get())
        b = float(entry_b.get())
        c = float(entry_c.get())
        d = b**2 - 4*a*c
        if d > 0:
            r1 = (-b + d**0.5) / (2*a)
            r2 = (-b - d**0.5) / (2*a)
            res = f"Két gyök van: {r1}, {r2}"
        elif d == 0:
            r = -b / (2*a)
```

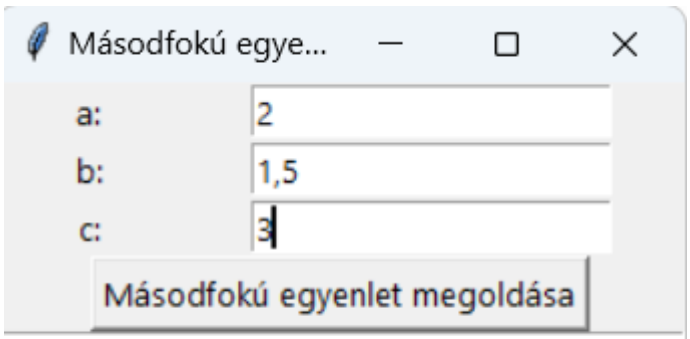
```
            res = f"Egy gyök: {r}"
        else:
            res = "Nincs valós gyök"

        res_widget.config(state=tk.NORMAL)
        res_widget.delete("1.0", tk.END)
        res_widget.insert(tk.END, res)
        res_widget.config(state=tk.DISABLED)

root = tk.Tk()
root.title("Másodfokú egyenlet")
label_a = tk.Label( text="a:"); label_a.grid(row=0, column=0)
entry_a = tk.Entry(); entry_a.grid(row=0, column=1)
label_b = tk.Label( text="b:"); label_b.grid(row=1, column=0)
entry_b = tk.Entry(); entry_b.grid(row=1, column=1)
label_c = tk.Label( text="c:");label_c.grid(row=2, column=0)
entry_c = tk.Entry(); entry_c.grid(row=2, column=1)

calculate_button = tk.Button( text="Másodfokú egyenlet
megoldása",  command=gyok2)
calculate_button.grid(row=3, columnspan=2)
res_widget = tk.Text( height=3, width=30)
res_widget.grid(row=6, columnspan=2)
root.mainloop()
```



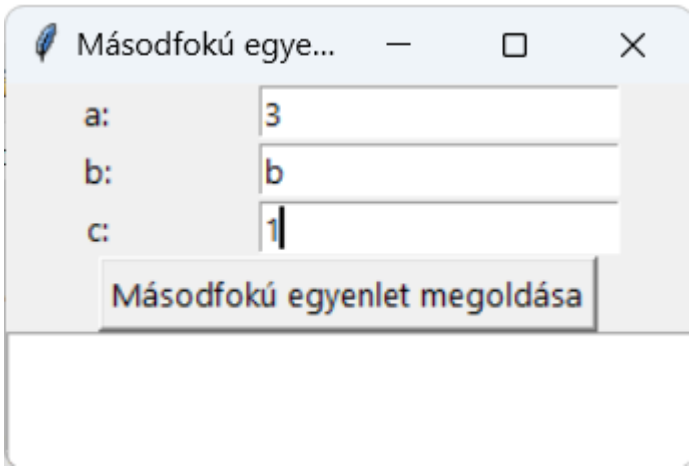

```
Két gyök van: -0.5, -1.0
```




```
Nincs valós gyök
```




```
Exception in Tkinter callback
Traceback (most recent call last):
  File "D:\Thonny\lib\tkinter\__init__.py", line 1921, in
__call__
    return self.func(*args)
  File "D:\python_thonny\kvadrat_a_b.py", line 22, in gyok2
    b = float(entry_b.get())
ValueError: could not convert string to float: '1,5'
```



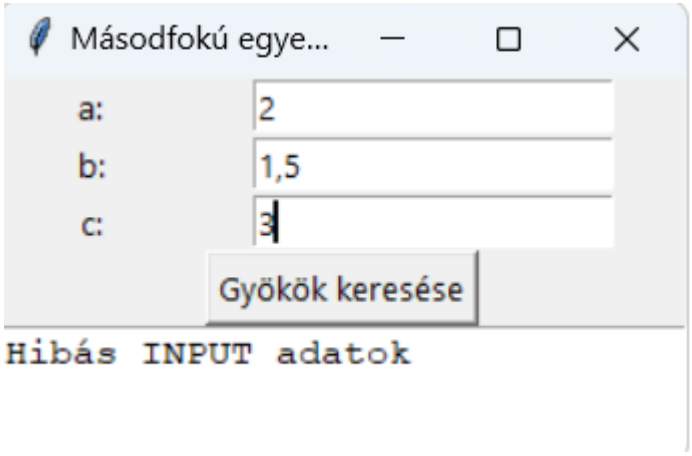

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "D:\Thonny\lib\tkinter\__init__.py", line 1921, in
__call__
    return self.func(*args)
  File "D:\python_thonny\kvadrat_a_b.py", line 22, in gyok2
    b = float(entry_b.get())
ValueError: could not convert string to float: 'b'
```

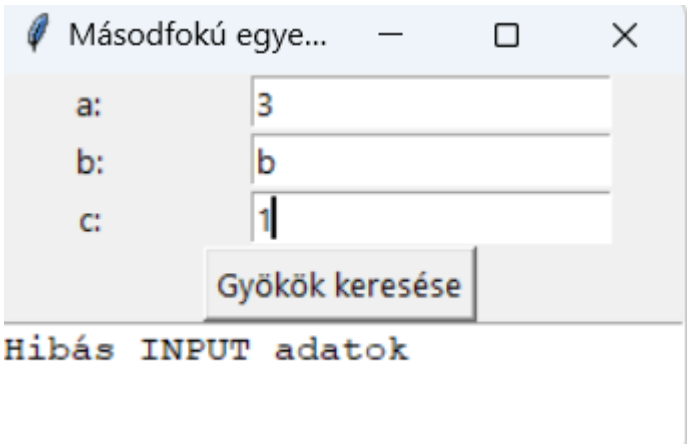
B) Másodfokú egyenlet megoldása (try, except alkalamzása)

```
import tkinter as tk
def gyok2():
    try:
        a = float(entry_a.get())
        b = float(entry_b.get())
        c = float(entry_c.get())
        d = b**2 - 4*a*c

        if d > 0:
            x1 = (-b + d**0.5) / (2*a)
            x2 = (-b - d**0.5) / (2*a)
```

```
            res = f"Két gyök: {x1:.3f}, {x2:.3f}"
        elif d == 0:
            x = -b / (2*a)
            res = f"Egy gyök: {x:.3f}"
        else:
            res = "Nincs valós gyök"

        res_widget.config(state=tk.NORMAL)
        res_widget.delete("1.0", tk.END)
        res_widget.insert(tk.END, res)
        res_widget.config(state=tk.DISABLED)
    except ValueError:
        res_widget.config(state=tk.NORMAL)
        res_widget.delete("1.0", tk.END)
        res_widget.insert(tk.END, "Hibás INPUT adatok")
        res_widget.config(state=tk.DISABLED)
root = tk.Tk()
root.title("Másodfokú egyenlet")
#  a, b, c bevitele:
label_a = tk.Label( text="a:")
label_a.grid(row=0, column=0)
entry_a = tk.Entry()
entry_a.grid(row=0, column=1)

label_b = tk.Label( text="b:")
label_b.grid(row=1, column=0)
entry_b = tk.Entry()
entry_b.grid(row=1, column=1)
label_c = tk.Label( text="c:")
label_c.grid(row=2, column=0)
entry_c = tk.Entry()
entry_c.grid(row=2, column=1)
```

```
# Gomb: Gyökök keresése
calculate_button = tk.Button( text="Gyökök keresése",
command=gyok2)
calculate_button.grid(row=3, columnspan=2)
# Eredmény kivitele
res_widget = tk.Text(height=3, width=30)
res_widget.grid(row=4, columnspan=2)
root.mainloop()
```





VISSZA

IRODALOMJEGYZÉK

1. Васильєв О. М. Програмування мовою Python. Тернопіль: Навчальна книга Богдан, 2019. 504с
2. Peter Wentworth, Jeffrey Elkner, Allen B. Downey and Chris Meyers, Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével, 2019 https://mtmi.unideb.hu/pluginfile.php/554/mod_resource/content/1/thinkcspy3.pdf
3. *Костюченко А.О.* Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М. 2020. 180 с.
4. https://www.w3schools.com/python
5. https://www.malnasuli.hu/python/keszits-hazilag-grafikus-vezerlofeluletet-tkinter-1-resz/
6. http://nyelvek.inf.elte.hu/leirasok/Python/index.php?chapter=16
7. https://szit.hu/doku.php?id=oktatas:programozas:python:tkinter:hagyomanyosan
8. https://mek.oszk.hu/08400/08435/08435.pdf

Модуль Tkinter бібліотеки Python/Python könyvtári Tkinter modul методичні вказівки до практичних занять з дисципліни «Сучасні мови програмування» для здобувачів першого (бакалаврського) рівня вищої освіти денної та заочної форм навчання, освітня програма: «Середня освіта (Інформатика)», галузь знань: «01 Освіта/Педагогіка», спеціальність (спеціалізація): «014 Середня освіта (014.04 Інформатика)» / Розробник: Йожеф Головач. – Берегове: ЗУІ ім. Ф.Ракоці II, 2024. 72 с. (угорською мовою).

Метою методичного посібника є поглиблення знань студентів з дисципліни « Сучасні мови програмування». У роботі описано модуль Tkinter мови програмування Python, наводяться приклади програм з використанням модуля Tkinter. Методичні вказівки можуть бути використані на практичних заняттях та при самостійній роботі по даному курсу. Наведені програми можуть бути використані, як взірці, при розв'язуванні студентами типових завдань по курсу «Сучасні мови програмування».

Виробничо-практичне видання

Модуль Tkinter бібліотеки Python/Python könyvtári Tkinter modul

Методичні вказівки до практичних занять з дісципліни «Сучасні мови програмування»

2024 р.

Затверджено до використання у навчальному процесі
на засіданні кафедри математики та інформатики ЗУІ ім. Ф.Ракоці ІІ
(протокол № 1 від «13» серпня 2024 року)

Розглянуто та рекомендовано Навчально-методичною радою
Закарпатського угорського інституту імені Ференца Ракоці ІІ
(протокол №22 від «26» серпня 2024 року)

Рекомендовано до видання в електронній формі (PDF)
рішенням Вченої ради Закарпатського угорського інституту імені Ференца Ракоці ІІ (протокол №7 від « 27» серпня 2024 року)

Розробник методичних вказівок:

Йожеф ГОЛОВАЧ – професор кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доктор технічних наук

Рецензенти:

Олександр МІЦА – завідувач кафедри інформаційних управляючих систем та технологій УжНУ, доктор техніічих наук, професор (м. Ужгород)

Мирослав СТОЙКА – доцент кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, кандидат фізико-математичних наук, доцент

Відповідальні за випуск:

Каталін КУЧІНКА – завідувач кафедри математики та інформатики Закарпатського угорського інституту імені Ференца Ракоці ІІ, доцент, кандидат фізико-математичних наук

Олександр ДОБОШ – начальник Видавничого відділу ЗУІ ім. Ф.Ракоці ІІ

Видавництво: Закарпатський угорський інститут імені Ференца Ракоці ІІ (адреса: пл. Кошута 6, м. Берегове, 90202. Електронна пошта: foiskola@kmf.uz.ua) *Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції Серія ДК 7637 від 19 липня 2024 року*

Шрифт «Courier New». Розмір сторінок методичних вказівок: А4 (210х297мм).