Programozás Python nyelven 3.

VIII. Grafikus felület, Tkinter modul
IX. Idő kezelése, kép kezelése, ...(Time, Pillow, ...)
X. Adatok tárolása, adatbázis kezelés
XI. Hálózat

8.1. Grafika (GUI)

GUI

Graphical User Interface (grafikus felhasználói felület)

Grafikus könyvtárak

- a grafikus felület felépítéséhez, működtetéséhez szükséges osztályokat tartalmazzák
- Pythonhoz több használható grafikus könyvtár van →
  pl. Tkinter, WxPython, pyQt, pyGTK, Kivy …
- Tkinter használata → Tkinter csomag importálása →

```
import tkinter                          # (2-es Pythonban  Tkinter  !!)
```

vagy

```
from tkinter import *                   # ha nagyon sokféle widgetet használunk
```

vagy

```
from tkinter import Button, Label, …    # konkrét widgetek megadása
```

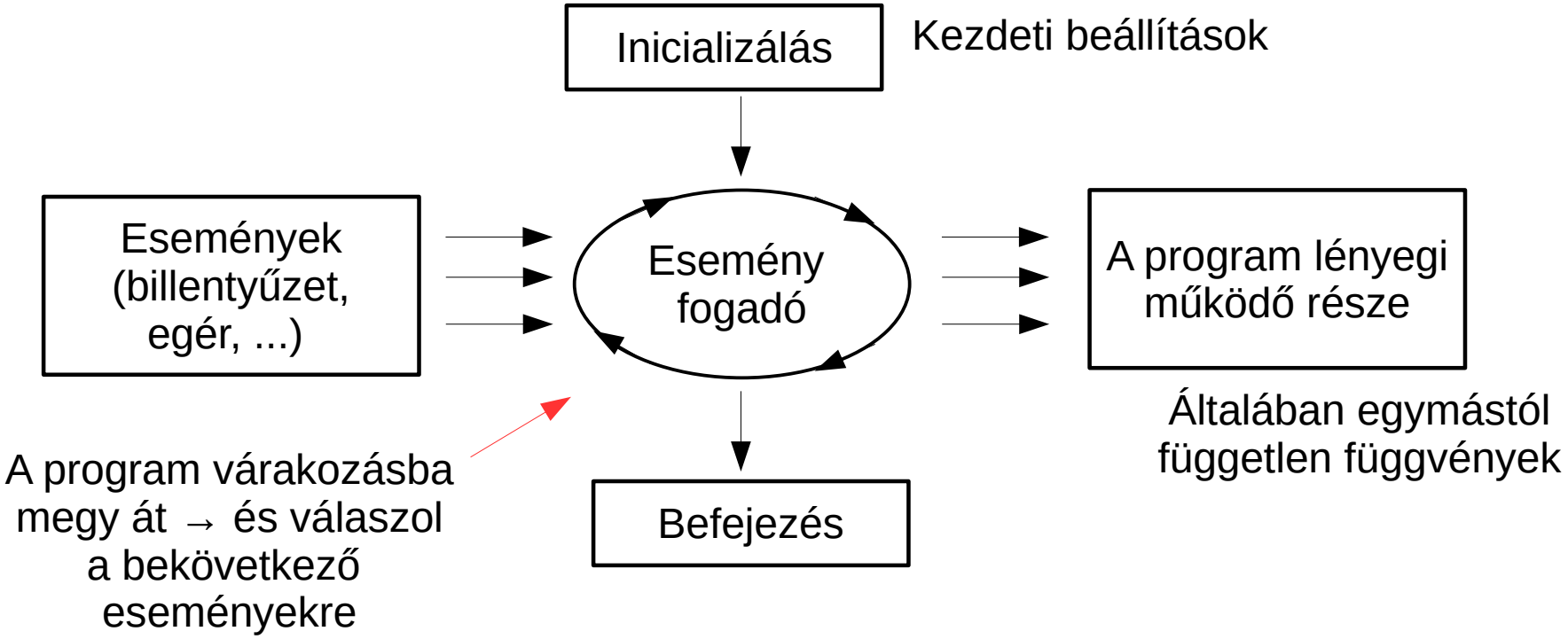
Widget (windows gadget)

- grafikus komponens (objektum) →
  pl. nyomógomb (button), ablak (window), címke (label), …
- ezekből épül fel egy program grafikai felülete

8.2. Grafika (GUI)

<u>Eseményvezérelt programozás</u>

- a program folyamatosan figyeli a külvilág eseményeit, és azoknak megfelelően csinál valamit (vagy nem csinál) →
- a kezdeti beállítások után a háttérben folyamatosan fut egy fő ciklus, amelyben le tudjuk folyamatosan kérdezni a perifériákat (vagy megszakítással jeleznek) és a megtörtént eseménynek (pl. egérkattintás) megfelelő programrészt (függvényt, metódust) megtudjuk hívni, illetve a szükséges attribútum módosításokat el tudjuk végezni
- sok esemény egyszerre ! → ~ párhuzamos működés



8.3. Tkinter widget osztályai

Tkinter alaposztályai

- Button (nyomógomb) → utasítás végrehajtása, alprogram meghívása
- Label (címke) → információ kiírása (esetleg kép)
- Entry (adat beviteli mező) → szöveg bekérése felhasználótól
- Text (szöveg) → formázott szöveg kiírása, szerkesztése (kép is lehet)
- Canvas (vászon) → rajzolásra, grafikus elemek elhelyezésére
- Checkbutton (jelölő négyzet) → be/ki kapcsolása dolgoknak
- Radiobutton (kiválasztó mező) → választás több elem közül
- Listbox (lista) → választás listából
- Frame (keret) → téglalap alakú felület, grafikus elemek elkülönítésére
- Menu, Menubotton (menü, menüelem)→ Menük létrehozására
- Scale ( ) → érték választás grafikusan
- Scrollbar (görgetősor) → más widgetekhez kapcsolható
- Message (üzenet) → szöveges üzenet kiírása

8.4. Tkinter widget osztályai

Tkinter alaposztályai

- Toplevel ( ) → külön, felülre kiírt ablak
- Tk (grafikus ablak)→ Tkinter alap osztálya, alkalmazás-ablak létrehozására
  → minden egyéb widgetet ezen helyezünk el !

pl. ablak egy címkével

```
import tkinter as tk      # tkinter grafikus könyvtár importálása

ablak1 = tk.Tk( )         #  Tk  osztály  egy példányának létrehozása  →fő ablak
udv = tk.Label(ablak1, text = "Helló !", fg='red')          #  egy címke létrehozása
udv.pack( )                                                 # címke elhelyezése
ablak1.mainloop( )                                  #  fő ablak esemény ciklusának indítása
```

- egy widget létrehozásakor meg kell adni 1. paraméterként, hogy melyik másik widgetben hozzuk létre pl. udv = tk.Label(ablak1,….)

Az eredeti Tkinter mellett egy ideje van egy új modul is ! → ttk
Ebben sok widget hasonló mint az eredeti Tkinterben, de van amelyik picit változott, és vannak teljesen új widgetek is.

8.5. Tkinter néhány metódusa

1. Mainloop metódus

- egy fő metódus, lényegében egy létrehozott ablak viselkedését, működését szabályozza
- egy programhurkot hoz létre, amely a háttérben folyamatosan fut ! → és várja az eseményeket, üzeneteket (az operációs rendszertől)
- másképpen értelmezve → folyamatosan lekérdezi környezetét, és reagál a bekövetkezett eseményekre → meghívja a megfelelő eseményt lekezelő metódust
- az eseményeket lekezelő metódusokat nekünk kell megírnunk !! → így tudjuk létrehozni a kívánt módon működő programot

2. Listbox widget metódusai

Index → 0 - END

curselection(), melyik a kiválasztott index
getactive(), melyik a kiválasztott elem
get(), az adott pozícióban levő elem
insert(index,újelem), új elem beszúrása
delete(kezdet,[vég]), elem(ek) törlése

8.6. Widget-ek pozícionálása

1. pack metódus

- többféle widgetre is alkalmazható (pl. Label, Button, ...)
- a widgetek geometriai elhelyezkedését állítja be
- pl. b1 = Button(..)
  b1.pack(side=LEFT,padx=3,pady=3)
- oldal (side) lehet: LEFT, RIGHT, TOP, BOTTOM
- widgetek közötti üres terület → padx, pady

2. grid metódus

- mátrix szerűen lehet elrendezni az elemeket
- sor, oszlop megadása → grid(row=x,column=y)
- ha valamelyiket nem adjuk meg → nulla
- igazítás → sticky → N,S,W,E (égtájak)
- pl. text1.grid(row=0,sticky=E)
  text2.grid(row=1,column=1)

-kiterjesztés több sorra → rowspan argumentum
pl. rowspan = 2

-kiterjesztés több oszlopra → columnspan argumentum

3. place metódus

8.7. Események

1. egér események

- bal gomb lenyomása → <Button-1>
- bal gomb lenyomva, közben egérmozgás → <Button1-Motion>
- bal gomb felengedése → <Button1-ButtonRelease>

2. billentyűzet események

- billentyű lenyomva, <KeyPress> → ’event’ esemény keysym attributuma tárolja melyik billentyű

vagy pl. enter → <Return>

pl. bal-nyíl lekérdezése → if event.keysym == 'Left' ….

3. bind metódus

- esemény összekapcsolása objektummal és az azt lekezelő függvénnyel
- pl.

```
self.c=Canvas( … )
self.c.bind(„<Button-1>”,self.gomb1le)
…
def gomb1le(self,event):
    ….
```

8.8. Események

4. event objektum

Az esemény jellemzőit tárolja
- automatikusan létrejövő objektum
- ezzel lehet átadni az eseménykezelőnek az esemény tulajdonságait →
- az esemény típusa → egérgomb lenyomása, felengedése, egér elmozdulása,
  egy billentyű lenyomása, kurzor egy adott zónába kerülése,
  ablak megnyitása, bezárása
  pl. egér bal gomb lenyomása → <Button-1>
- az esemény egyéb tulajdonságai → keletkezésének pillanata, koordinátái,
  az érintett widgetek jellemzői

5. egérpozíció lekérdezése

- event.x event.y

8.9. Button, Label

1. Minta program

pl. ablak egy címkével és egy nyomógombbal

```
import tkinter as tk
ablak1 = tk.Tk( )        #  Tk  osztály  egy példányának létrehozása  →fő ablak
szoveg1= tk.Label(ablak1, text='Hello!', fg='red')   #  egy címke létrehozása
szoveg1.pack( )
gomb1= tk.Button(ablak1, text='Kilépés', command=ablak1.destroy)
gomb1.pack( )                    #  egy nyomógomb létrehozása
ablak1.mainloop( )          #  fő ablak esemény ciklusának indítása

#  command= … a nyomógombhoz rendelt metódus/függvény megadása
#  a nyomógombra kattintva → a metódus meghívódik
# Tk  ablak  destroy()  metódusa → bezárja az ablakot → a program vége
# text paraméter → Label, Button,… szövegének megadása
# fg paraméter → szöveg szinének megadása
```

8.10. Button, Label

1. Minta program másképp

Általában inkább így szokás ! Saját osztályt hozunk létre, egy Tk osztályból (vagy Frame osztályból)

```
import tkinter as tk

class Ablak1(tk.Tk):
    """saját ablak osztály"""
    def __init__(self):
        tk.Tk.__init__(self)
        self.szoveg1= tk.Label(self, text='Hello!', fg='red')   #  egy címke létrehozása
        self.szoveg1.pack( )
        self.gomb1= tk.Button(self, text='Kilépés', command=self.destroy)
        self.gomb1.pack( )                          #  egy nyomógomb létrehozása

if __name__ == '__main__':
    Ablak1().mainloop()          # itt indul a program !!
```

- először létrejön egy Ablak1 objektum → meghívódik az __init__ () metódusa, és abban létrehozzuk az ablakra szánt widgeteket
- majd meghívódik az így létrehozott Ablak1 objektum mainloop() metódusa

8.11. Canvas, PhotoImage

2. Minta program

Kép beillesztése Canvasra, és cimkék, szövegbeviteli mezők

```
import tkinter as tk

abl1 = tk.Tk()
# a 'Label' és 'Entry' widgetek létrehozása:
txt1 = tk.Label(abl1, text ='Első mező :')
txt2 = tk.Label(abl1, text ='Második :')
mezo1 = tk.Entry(abl1)
mezo2 = tk.Entry(abl1)

# egy bitmap képet tartalmazó 'Canvas' widget létrehozása
can1 = tk.Canvas(abl1, width =160, height =160, bg ='white')
photo = tk.PhotoImage(file ='virag11.gif')
item = can1.create_image(80, 80, image =photo)
# elhelyezés a 'grid' metódus segítségével
txt1.grid(row =1, sticky = "E")
txt2.grid(row =2, sticky = "E")
mezo1.grid(row =1, column =2)
mezo2.grid(row =2, column =2)
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)

abl1.mainloop()          # indítás
```

8.12. Entry

3. Minta program

Egyszerű számológép

```
import tkinter as tk
from math import *

def kiertekel(event):
    kijelzo.configure(text = "Eredmény" +  str(eval(bead.get() )))
        #  eval → beépített függvény

ablak1 = tk.Tk( )        #   Tk  osztály  egy példányának létrehozása  →fő ablak
bead = tk.Entry(ablak1)          #  egy beviteli mező létrehozása
bead.bind("<Return>",kiertekel)
bead.pack( )
kijelzo = tk.Label(ablak1)         #  egy címke létrehozása
kijelzo.pack( )
ablak1.mainloop( )                 #  fő ablak esemény ciklusának indítása
```

8.13. Metódusok

3. Minta program, metódusok, függvények

- Entry osztály get() metódusa
  a beírt érték kiolvasása

- eval() beépített függvény
  az interpreterrel kiértékelteti az átadott kifejezést

- configure() metódus
  létező widgetek tulajdonságainak módosítása

- str() beépített függvény
  számot sztringgé alakít

- Entry osztály insert() metódusa
  a beviteli mező értékének módosítása

- Tk osztály after(idő,függvény) metódusa
  „időnként” (millisecundum) automatikusan meghívja a megadott függvényt
  → animáció

8.14. Frame

4. Minta program

Egérkattintás kezelése

```
import tkinter as tk

def mutato(event):              #  egér kattintás lekezelése
      kijelzo.configure(text = "kattintás, x = " + str(event.x) + ", y =" + str(event.y) )

ablak1 = tk.Tk( )          #   fő ablak
keret = tk.Frame(ablak1,width = 200,height = 150,bg = "light yellow")
keret.bind("<Button-1>",mutato)    # bal egérgomb lenyomása
keret.pack( )
kijelzo = tk.Label(ablak1)   #  egy címke létrehozása
kijelzo.pack( )
ablak1.mainloop( )             #  fő ablak esemény ciklusának indítása
```

8.15. Rajzolás

5. Minta program

Rajzolás Canvas widgetre

```
import tkinter as tk
from random import randrange

def vonalrajz():            #  egyenes rajzolás vélelenszerű helyre
    x1 = randrange(szeles)      # koordináták véletlenszerű választása
    x2 = randrange(szeles)
    y1 = randrange(magas)
    y2 = randrange(magas)
    rajz.create_line(x1,y1,x2,y2,width=2,fill=szin)

szeles = 200        #  globális változók, szélesség, magasság, rajzolás színe
magas = 200
szin= "red"
abl1 = tk.Tk( )             #   fő ablak
rajz = tk.Canvas(abl1,width = 200,height = 200,bg = "dark grey")
rajz.pack(side = 'left' )       #  canvas létrehozása
gomb1= tk.Button(abl1, text='Vonalat rajzol', command=vonalrajz)
gomb1.pack()                #  egy nyomógomb létrehozása
gomb2= tk.Button(abl1, text='Kilépés', command=abl1.quit)
gomb2.pack(side = 'bottom' )          #  még egy nyomógomb létrehozása
abl1.mainloop( )        #  fő ablak esemény ciklusának indítása
```

8.16. Rajzolás

6. Minta program

Rajzolás Canvas widgetre, mozgatás billentyűzettel

```
import tkinter as tk

def key_pressed(event):
    circle_coord = rajz.coords(circle)
    if event.keysym == 'Left' and circle_coord[0] > 10:
        rajz.move(circle, -10, 0)          # már megrajzolt objektum mozgatása
    if event.keysym == 'Right' and circle_coord[0] <= szeles-30:
        rajz.move(circle, 10, 0)
    if event.keysym == 'Up' and circle_coord[1] > 10:
        rajz.move(circle, 0, -10)
    if event.keysym == 'Down' and circle_coord[1] <= magas-30:
        rajz.move(circle, 0, 10)

szeles = 500
magas = 400
abl = tk.Tk()
abl.title('Mozgatás')
rajz = tk.Canvas(master=abl, width=szeles, height=magas, bg='white')
rajz.pack()
circle = rajz.create_oval(szeles/2, magas/2, szeles/2 +20, magas/2 +20,
                                                    width=3, outline='black', fill='yellow')
abl.bind('<KeyPress>', key_pressed)
abl.mainloop()
```

8.17. Canvas osztály

Canvas osztály metódusai

create_line(x1,y1,x2,y2,width,fill)
egyenes rajzolása, (x1,y1) ponttól (x2,y2) pontig, width → vonalvastagság, fill → szín

create_oval(x1,y1,x2,y2,width,fill)
ellipszis,kör rajzolása, (x1,y1) - (x2,y2) téglalapba, width → vonalvastagság, fill → szín

create_rectangle(x1,y1,x2,y2,width,fill)
téglalap rajzolása, (x1,y1)→ bal felső sarok, (x2,y2) → jobb alsó sarok,
width → vonalvastagság, fill → kitöltés színe

create_text(x,y,text,anchor)
szöveg kiírása, (x,y) ponttól, text →szöveg, anchor→ igazítás

move(obj,dx,dy)
canvason lévő objektum mozgatása

itemconfigure(obj,fill=…, ...)
canvason lévő objektum módosítása

delete(ob)
canvason lévő objektum törlése

delete(ALL)
canvason lévő minden objektum törlése !!

8.18. Checkbutton

7. Minta program
kiválasztás

```
import tkinter as tk

def kiertekel():
        if valt1.get()==1:              #  checkbutton értékének lekérdezése (be-1 vagy ki-0)
                str1 = "t1 BE"
        else:
                str1 = "t1 KI"
        if valt2.get()==1:
                str2 = "t2 BE"
        else:
                str2 = "t2 KI"
        kijelzo.configure(text = str1 + "  "+ str2)

abl = tk.Tk( )          #   Tk  osztály  egy példányának létrehozása  →fő ablak
valt1 = tk.IntVar()
valt2 = tk.IntVar()
cb1 = tk.Checkbutton(abl,text = "t1 opció",variable=valt1)
cb1.pack( )             #  egy checkbutton létrehozása, és elhelyezése
cb2 = tk.Checkbutton(abl,text = "t2 opció",variable=valt2)
cb2.pack( )             #  egy checkbutton létrehozása, és elhelyezése
gomb= tk.Button(abl, text='Kiértékel', command=kiertekel)
gomb.pack(side = 'bottom' )          #   egy nyomógomb létrehozása
kijelzo = tk.Label(abl,text = "?")      #  egy címke létrehozása
kijelzo.pack( )
abl.mainloop( )             #  fő ablak esemény ciklusának indítása
```

8.19. Radiobutton

8. Minta program
választás

```
import tkinter as tk

def kiertekel():
        str1 = rbvalt.get()                  #  radiobutton  lekérdezése
        kijelzo.configure(text = str1)

abl = tk.Tk( )          #   Tk  osztály  egy példányának létrehozása  →fő ablak
rbvalt = tk.StringVar()
rb1 = tk.Radiobutton(abl,text = "választás 1",variable=rbvalt,value="t1",command=kiertekel)
rb1.pack( )             #  egy radiobutton létrehozása, és elhelyezése
rb2 = tk.Radiobutton(abl,text = "választás 2",variable=rbvalt,value="t2",command=kiertekel)
rb2.pack( )             #  egy másik radiobutton létrehozása, és elhelyezése
kijelzo = tk.Label(abl,text = "?")      #  egy címke létrehozása
kijelzo.pack( )
rbvalt.set("t1")        #  melyik radiobutton legyen alapértelmezetten kijelölve
kiertekel( )
abl.mainloop( )             #  fő ablak esemény ciklusának indítása
```

9.1. Time modul

Time modul függvényei

time()

Az 1970.01.01. 0 óra, 0 perc, 0 másodparc óta eltelt időt adja meg másodpercben (float típus)

```
# 9.1. mintaprogram
import time

t1= time.time()
print(t1)              # 1658496156.6345294
```

gmtime()

Az UTC szerinti időt adja meg objektumként

localtime()

A helyi időt adja meg objektumként →

attributumai → tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec, tm_wday, …

tm_wday → a hét napjai, 0,1,2,...6 (H,K,...)

```
# 9.2. mintaprogram
import time

t1loc = time.localtime()
print(t1loc)
    # vagy
print("év: ", t1loc.tm_year, " hó: ", t1loc.tm_mon, " nap: ", t1loc.tm_mday)
```

9.2. Time modul

Time modul függvényei

monotonic()

A számítógép bekapcsolása óta eltelt időt adja meg másodpercben (float típus), jól használható időtartamok számításához

sleep(idő_másodpercben)

A programot a megadott időre alvó állapotba teszi

```
# 9.3. mintaprogram
import time

t1= time.monotonic()        # 1. időpont
print(t1)
for x in range(10000):
    negyzet=x*x
time.sleep(1)
t2= time.monotonic()        # 2. időpont
print(t2)
print("eltelt idő: ", t2-t1)    # →  pl. 1.0024719229995753
```

9.3. Datetime modul

Datetime modul datetime osztálya

utcnow() metódus

Az UTC szerinti időt adja meg datetime objektumként

now() metódus

A helyi időt adja meg datetime objektumként →

attributumai → year, month, day, hour, minute, second, microsecond

```
# 9.4. mintaprogram
import datetime

t1loc = datetime.datetime.now()
print(t1loc)                # 2022-07-22 16:20:29.333439
    # vagy
print("év: ", t1loc.year, " hó: ", t1loc.month, " nap: ", t1loc.day)
                            # év: 2022 hó: 7 nap: 22
```

date() metódus (datetime objektum metódusa)

A dátumot adja meg (év-hó-nap)

time() metódus (datetime objektum metódusa)

Az időt adja meg (h:m:s.mikrosec)

```
# 9.5. mintaprogram
datum = t1loc.date()
print(datum)                # 2022-07-22
ido = t1loc.time()
print(ido)                  # 16:20:29.333439
```

9.4. Képek kezelése, PIL, Pillow

Pillow telepítése

Igazából a Pillow csomag a PIL csomag tovább fejlesztése. A PIL csak a Python 2 változatnál volt, már nem fejlesztik (lehet hogy újra van PIL, a Pillow forkja ?).
Nem része az alap Python telepítésnek, külön fel kell telepíteni !
Telepítése a Python pip csomagkezelőjével: pip install Pillow vagy
pip install –upgrade Pillow (ha a legfrissebb csomagot akarjuk, akkor így célszerű)

Telepítése Linux saját csomagjaként: általában python-imaging vagy python-pillow
vagy python-pil és python-pil.imagetk néven (mindkét csomag kell !)

Importálása PIL néven továbbra is:
import PIL
vagy
from PIL import Image, ImageTk, ...

Pillow moduljai

- Image
- ImageDraw
- ImageTk
-

9.5. Pillow, Image modul

Image modul függvényei, Image osztály metódusai

Az Image modul tartalmaz egy Image osztályt. Ennek példányaihoz tudunk képet rendelni.

open("kép neve")

Kép betöltése fájlból. Sok formátumot ismer. (jpg, png, gif, …)
Image objektumot ad vissza.

new("formátum",méret,háttérszín)

új kép (Image objektum) létrehozása

save("fájl neve")

Kép (Image objektum) mentése fájlba. Más kép típus is lehet, mint amilyen eredetileg !
→ konvertál, ha tud !

show()

Kép megjelenítése, meghív egy képmegjelenítő alkalmazást !

```
# 9.6. mintaprogram
from PIL import Image

kep = Image.open("virag.png")  # betöltés fáljból, egy Image objektumba (kep)
kep.save("virag.jpg")          # mentés fáljba, Image osztály save metódusával
kep.show()
kep2 = Image.new("RGB",(400,400),(255,20,20))       #új Image objektum (kep2)
kep2.save("kep2.jpg")
kep2.show()
```

9.6. Pillow, Image modul

Image objektum attributumai

size Kép mérete (tuple → szélesség, magasság pixelben)

width Kép szélessége (pixelben)

height Kép magassága (pixelben)

format Kép forrásának típusa, pl. gif, jpg ... (None, ha nem fájlból lett betöltve)

mode pixel formátuma (RGB, RGBA, CMYK, YCbCr, L, P, ...)

RGB → 3x8 bit true color, RGBA 4x8 bit true color (RGB + átlátszóság),
L, P → 8 bit, fekete-fehér ….

info Kép adatai összesítve szótárban

```
# 9.7. mintaprogram
from PIL import Image
kep = Image.open("virag.png")              # betöltés fáljból, egy Image objektumba
print(kep.format, kep.size, kep.mode)  # →  PNG  (560,613)  RGB
kep.save("virag.jpg")
kep2 = Image.new("RGBA",(300,300),(20,20,255,50))      #új Image objektum (kep2)
print(kep2.format, kep2.size, kep2.mode)                  # →  None  (300,300)  RGBA
```

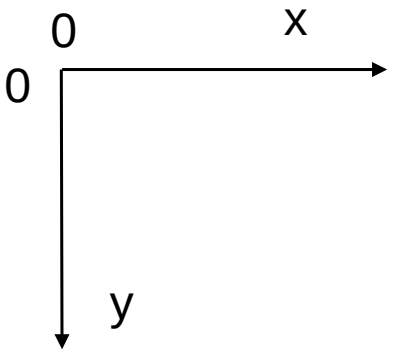
9.7. Pillow, Image modul

Image osztály metódusai

crop(box) Kép részlet kivágása\kimásolása, 'box' egy 4-es tuple → x1,y1,x2,y2 →
→ bal-fent,jobb-lent (egy új Image objektumot ad vissza)

copy() Kép másolása (egy új Image objektumot ad vissza)

paste(képrészlet,box)
Kép részlet bemásolása képbe



rotate(szög)
Kép elforgatása (jobbra)
egy új Image objektumban adja vissza elforgatva !!

```
# 9.8. mintaprogram
from PIL import Image

kep = Image.open("virag.png")                  # betoltés fáljból
kep2 = kep.crop((0,0,100,100))                 # képrészlet kimásolás
kep.paste(kep2,(200,200,300,300))              # képrészlet vissza másolása
kep.show()
kep3 = kep.rotate(90)                          # kép elforgatása, új Image objektum !!
kep3.save("furcsavirag.png")
```

9.8. Pillow, Image modul

Image osztály metódusai

```
# 9.9. mintaprogram
from PIL import Image

kep1 = Image.open("beach1.jpg")          # betoltés fáljból  (kép 400x300)
kep2 = Image.open("palmafa.jpg")         # betoltés fáljból  (kép 400x300)
kep3 = kep1.crop((100,120,300,220))            # képrészlet (200x100) kimásolás
kep2.paste(kep3,(0,0,200,100))           # képrészlet bemásolása (200x100)
kep2.show()
kep2.save("furcsa.jpg")
```

beach1.jpg

palmafa.jpg

furcsa.jpg

9.9. Pillow, ImageDraw modul

ImageDraw modul, Draw osztály

Az Image modul Image objektumához rendel egy rajzolási felületet (Draw osztály példánya), amelyre ezután a metódusaival tudunk rajzolni.

Image objektumhoz rendelés:

```
rajz = ImageDraw.Draw(Image_obj)
```

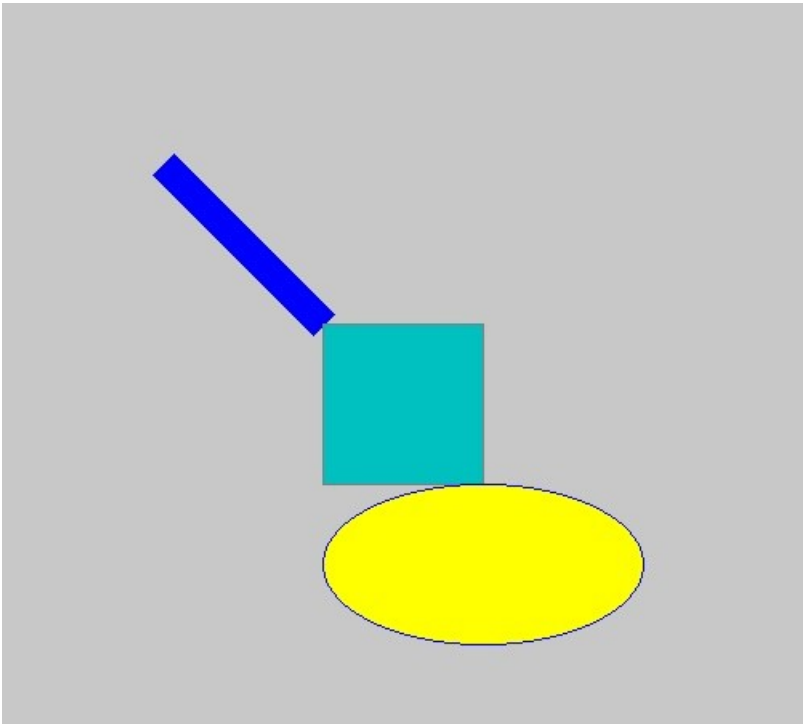
Metódusai:

```
line((x1, y1, x2, y2), fill=szin, width=x)    # egyenes rajzolása
      (x1, y1, x2, y2) → két végpont koordinátája
      fill → szín, 3-as tuple (R,G,B)
      width → vonalvastagság

rectangle((x1, y1, x2, y2), fill=szin, outline=szin2)      # téglalap rajzolása
      (x1, y1, x2, y2) → bal felső és jobb alsó sarok koordinátája
      fill → kitöltés színe, 3-as tuple (R,G,B), outline → körvonal színe (R,G,B)

ellipse((x1, y1, x2, y2), fill=szin, outline=szin2)    # ellipszis rajzolása
      a köréírt téglalap bal felső és jobb alsó csúcsának koordinátáit kell megadni
      fill → kitöltés színe (R,G,B),     outline → körvonal színe (R,G,B)
```

9.10. Pillow, ImageDraw modul

ImageDraw modul, Draw osztály

```
# 9.10. mintaprogram
from PIL import Image as im
from PIL import ImageDraw as imd

kep = im.new("RGB",(500,450),(200,200,200))    # új Image objektum (kep)
rajz = imd.Draw(kep)        # az Image objektumhoz hozzárendelünk egy Draw objektumot
rajz.line((100, 100, 200, 200), fill=(0, 0, 255), width=20)
rajz.rectangle((200, 200, 300, 300), fill=(0, 192, 192), outline=(128, 128, 128))
rajz.ellipse((200, 300, 400, 400), fill=(255, 255, 0), outline=(0, 0, 255))
kep.show()
kep.save('kep.jpg', quality=95)
```



9.11. Pillow, ImageDraw modul

ImageDraw modul, Draw osztály

További metódusok:

polygon(((x1, y1),(x2, y2),(x3,y3),...), fill=szin, outline=szin2) # sokszög rajzolása
csúcsok → pont sorozatot (x,y koordináták) kell megadni → tuple-ben 2-es tuple-k
fill → kitöltés színe (R,G,B), outline → körvonal színe (R,G,B)

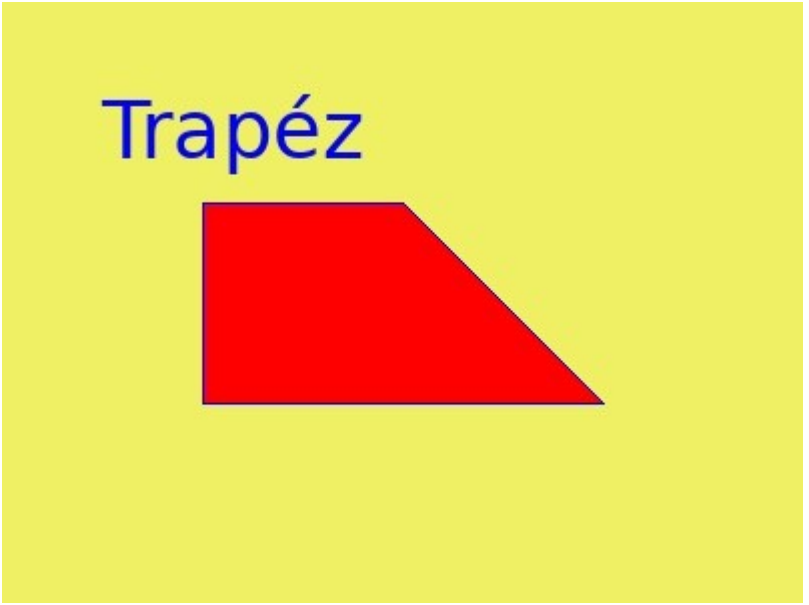
text((x1, y1), text, font=imfont_obj, fill=szin) # szöveg (text) elhelyezése
(x1, y1) pozíciótól
font → ImageFont modul kell hozzá !
pl. ImageFont.truetype('arial.ttf', 30)

textsize(text, font) # szöveg méretét adja meg → tuple (width,height)

9.12. Pillow, ImageDraw modul

ImageDraw modul, Draw osztály

```
# 9.11. mintaprogram
from PIL import Image as im
from PIL import ImageDraw as imd
from PIL import ImageFont as imf

kep = im.new("RGB",(400,300),(240,240,100))    # új Image objektum (kep)
rajz = imd.Draw(kep)        # az Image objektumhoz hozzárendelünk egy Draw objektumot
rajz.polygon(((100,100),(200,100),( 300, 200),(100,200)), fill=(255,0,0), outline=(0,0,255))
font1 = imf.truetype("DejaVuSans.ttf", 30)                    # font megadása
rajz.text((50, 40), "Trapéz", font=font1, fill=(0,0,255) )    # szöveg kirajzolása
kep.show()
kep.save('kep2.jpg', quality=95)
```

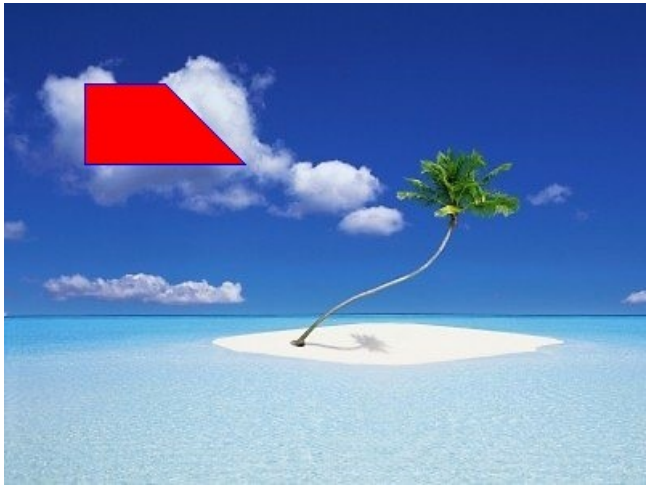


9.13. Pillow, ImageDraw modul

ImageDraw modul, Draw osztály

Rajzolhatunk meglévő képre is.

```
# 9.12. mintaprogram
from PIL import Image as im
from PIL import ImageDraw as imd

kep = im.open("palmafa.jpg")                              # betoltés fáljból  (kép 400x300)
rajz = imd.Draw(kep)        # az Image objektumhoz hozzárendelünk egy Draw objektumot
rajz.polygon(((50,50),(100,50),( 150, 100),(50,100)), fill=(255,0,0), outline=(0,0,255))
kep.show()
kep.save('palmafa2.jpg', quality=95)
```



9.14. Pillow, Image modul

Image osztály további metódusai

thumbnail((new_width, new_height))	Kép kicsinyítése
resize((new_width, new_height))	Kép átméretezése → új objektumba !
merge(mode, bands)	Képek összevonása
filter(ImageFilter.xfilter)	Kép szűrése, ImageFilter osztály kell hozzá ! ImageFilter osztály tartalmaz különféle szűrőket (xfilter), pl. BLUR → elmosódottság
transpose(mode)	Kép tükrözése, elforgatása → új objektumba ! mode → Image modul konstansai

FLIP_LEFT_RIGHT → tükrözés vízszintesen
FLIP_TOP_BOTTOM → tükrözés függőlegesen
ROTATE_90 →elforgatás 90 fokkal

9.15. Pillow, Image modul

Image osztály további metódusai

```
# 9.13. mintaprogram
from PIL import Image

kep = Image.open("palmafa.jpg")                          # betoltés fáljból
kep2 = kep.transpose(Image.FLIP_LEFT_RIGHT)              # tükrözés vízszintesen
kep2.show()
kep2.save("palmafa4.jpg")
```

palmafa.jpg

palmafa4.jpg

9.16. Pillow, ImageTk modul

ImageTk modul

Tkinter BitmapImage, PhotoImage objektumait tudjuk manipulálni PIL-ből.

osztályai:

PIL.ImageTk.BitmapImage
Tkinter kompatibilis BitmapImage

PIL.ImageTk.PhotoImage
Tkinter kompatibilis PhotoImage

```
# 9.14. mintaprogram
import tkinter as tk
from PIL import Image as im
from PIL import ImageDraw as imd
from PIL import ImageTk as imtk

kep = im.new("RGB",(600,500),(200,200,200))    # új Image objektum (kep)
rajz = imd.Draw(kep)        # az Image objektumhoz hozzárendelünk egy Draw objektumot
rajz.ellipse((200, 300, 400, 400), fill=(255, 255, 0), outline=(0, 0, 255))

ablak1 = tk.Tk( )        #   Tk  osztály  egy példányának létrehozása  →fő ablak
foto = imtk.PhotoImage(kep)          # PhotoImage objektum létrehozása
szoveg1= tk.Label(ablak1, image=foto)    #  egy címke létrehozása  képpel
szoveg1.pack( )

ablak1.mainloop( )          #  fő ablak esemény ciklusának indítása
```

9.17. Pillow, ImageTk modul

ImageTk modul

```
# 9.15. mintaprogram
from tkinter import *
from PIL import Image, ImageTk

class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.pack(fill=BOTH, expand=1)

        im = Image.open("viragszn2.jpg")
        kep = ImageTk.PhotoImage(im)
        img = Label(self,image=kep)
        img.image = kep
        img.place(x=0, y=0)


root = Tk()
app = Window(root)
root.wm_title("Tkinter window")
root.geometry("600x600")
root.mainloop()
```

10.1. Adatok tárolása

DBM modul

- DBM fájlok → karakterláncokat tudnak tárolni
- hasonlóan manipulálhatóak mint a szótárak
- megnyitás → open(fájlnév,'c') # 'c' létrehozza a fájlt, ha még nem létezik
- lezárás → close()
- kulcsok kinyerése listába → keys()
- elem törlése → del

```
# 10.1. mintaprogram

import dbm

file1 = dbm.open('fajl1','c')        # létrehozás
file1['egy'] = "elso elem"           # írás bele
file1['ketto'] = "második elem"
file1['harom'] = 'harmadik elem'
del file1['ketto']                   # elem törlése
file1.close()                        # lezárás

file1 = dbm.open('fajl1')            # megnyitás olvasásra
for x in file1.keys():                       #olvasás
    print(x,':  ',file1[x])
file1.close()
```

10.2. Adatok tárolása

Pickle modul

- memória objektumokat tárol binárisan fájlban! → serializálás → dump() metódus
  → adatok tárolhatók, hálózaton továbbíthatók
- adat visszanyerése → deserializálás → load() metódus
- nem struktúrált → nincs gyors keresés

- serializálás

```
# 10.2. mintaprogram

import pickle

szotar = {'egy':'1', 'ketto':'2','harom':'3','negy':'4'}
file1 = open('ki.txt','bw')          # bináris fájl létrehozása
pickle.dump(szotar,file1)            # írás bele
file1.close()                        # fájl lezárás
```

- deserializálás

```
file2 = open('ki.txt','br')          # bináris fájl megnyitás olvasásra
szotar2 = pickle.load(file2)         #olvasás
file2.close()
print(szotar2)
```

10.3. MySQL, MariaDB

Telepítés

MySQL vagy MariaDB adatbázis rendszerek használatához Python-ból a mysql-connector-python csomag szükséges.
Ez nem része az alap Python telepítésnek, külön fel kell telepíteni !
Telepítése a Python pip csomagkezelőjével: pip install mysql-connector-python

Importálása:
import mysql.connector

MariaDB telepítéséhez érdemes a XAMPP csomagot használni (ingyenesen letölthető, használható), az tartalmazza az Apache + MariaDB + PHP + Perl összetevőket.
Egy mappába települ az egész.

A szerverek indítása linux alatt:
sudo */opt/lampp/lampp start*
vagy grafikus felület indítása
sudo */opt/lampp/manager-linux-x64.run*

10.4. MySQL, MariaDB

Csatlakozás az adatbáziskezelőhöz

A mysql.connector modul connect osztályával:
connect(host="gépnév", user="username", password="passw")
→ kapunk egy adatbázis objektumot, rajta keresztül érjük el az adatbázisokat
Vagy, egy létező konkrét adatbázishoz csatlakozás:
connect(host="gépnév", user="username", password="passw", database= "db_name")

```
# 10.3. mintaprogram
import mysql.connector as sqlc
mydb=sqlc.connect(host="localhost", user="proba", password="proba123")
```

Cursor objektum létrehozása

cursor()
Rajta keresztül végezhető lekérdezés, és az eredményeket is rajta keresztül érjük el.
SQL lekérdezés → a cursor objektum execute("SQL parancs") metódusával

```
# 10.4. mintaprogram
import mysql.connector as sqlc
mydb = sqlc.connect(host="localhost", user="proba", password="proba123")
mycur= mydb.cursor()
```

10.5. SQL parancsok

Adatbázisok létrehozása, listázása

CREATE DATABASE database_name → adatbázis létrehozása

SHOW DATABASES → meglévő adatbázisok listázása

```
# 10.5. mintaprogram
import mysql.connector as sqlc
mydb = sqlc.connect(host="localhost", user="proba", password="proba123")
mycur= mydb.cursor()
mycur.execute("CREATE DATABASE proba")
mycur.execute("SHOW DATABASES")
for x in mycur:
  print(x)
```

10.6. SQL parancsok

Adattáblák létrehozása, listázása, módosítása

```
CREATE TABLE table_name  (mező1_név típus1, mező2_név típus2, ... )
SHOW TABLES
ALTER TABLE table_name  ADD COLUMN mező_név típus  egyéb_jellemzők
```

```
# 10.6. mintaprogram

import mysql.connector as sqlc

mydb = sqlc.connect(host="localhost", user="proba", password="proba123", database= "proba")
mycur= mydb.cursor()
mycur.execute("CREATE TABLE  tabl1 (nev VARCHAR(60), tel VARCHAR(30))")
mycur.execute("SHOW TABLES")
for x in mycur:
  print(x)
mycur.execute("ALTER TABLE  tabl1 ADD COLUMN id INT  AUTO_INCREMENT PRIMARY KEY")
```

10.7. SQL parancsok

Adattáblák feltöltése, rekordok törlése

INSERT INTO table_name (mező1, mező2, ...) VALUES ("érték1","érték2", ...)

DELETE FROM `table_name` WHERE ….

!! kell utána az adatbázis objektum commit() metódusát meghívni !! hogy valóban végrehajtódjon

```
# 10.7. mintaprogram

import mysql.connector as sqlc

mydb = sqlc.connect(host="localhost", user="proba", password="proba123", database= "proba")
mycur = mydb.cursor()
sql_str = "INSERT INTO  tabl1 (nev, tel) VALUES (%s,%s)"
rec1 = ("KIS_A","334466")
rec2 = ("NAGY_B","884411")
rec3 = ("VÍG_C","345678")
rec4 = ("SZÉP_D","123678")
mycur.execute(sql_str,rec1)
mycur.execute(sql_str,rec2)
mycur.execute(sql_str,rec3)
mycur.execute(sql_str,rec4)
mydb.commit()
```

10.8. SQL parancsok

Adattábla lekérdezése

SELECT * FROM table_name

utána a cursor objektum fetchall() metódusa adja vissza az eredményeket

```
# 10.8. mintaprogram

import mysql.connector as sqlc

mydb = sqlc.connect(host="localhost", user="proba", password="proba123", database= "proba")
mycur = mydb.cursor()

mycur.execute("SELECT * FROM tabl1")
eredmeny = mycur.fetchall()
for x in eredmeny:
  print(x)
```

11.1. Hálózat

Socket (csatorna)

- szoftver objektum (adat szerkezet)
- kommunikációs pontot valósít meg
- programok összekapcsolódását teszi lehetővé
  - Ugyanazon számítógépen futó két program között (IPC)
  - Két számítógép hálózaton keresztüli kommunikációjához

Socket - Port

A csatornák portokhoz vannak kötve → azonosításuk számmal (16 bit)

Kiszolgáló elérése: szerver IP címe + port szám

A kiszolgáló program portja

Kiszolgáló kommunikációs portja

- jól ismert
- mindig nyitva →
- mindig kész az ügyfelek kéréseinek kiszolgálására

Kliens kommunikációs portja

- nem biztos hogy jól ismert és mindig nyitva van
- kapcsolódni akar szerverhez → kommunikációs pont létrehozása → csatlakozás a szerver ismert kommunikációs pontjához → adatcsere → kapcsolat végén a kliens kommunikációs pontja bezáródik !

11.2. Hálózat

Protokollok - Portok

- Protokoll: kommunikációs szabályok
- Sokféle protokoll létezik, különböző hálózati szerver alkalmazások más-más protokollokat használnak
- két alap típusa van:
    1. kapcsolat orientált (pl. TCP)
    2. kapcsolat nélküli (pl. UDP)

A fontos, gyakran használt protokollokhoz → alapértelmezett portok vannak kiosztva

pl. FTP – 21 HTTP – 80 POP – 110 SMTP – 25 Telnet – 23

Socket (csatorna) használata

Socket modul → csatorna alapú programozás

11.3. Socket modul

Socket modul

Csatorna létrehozása:

socket() metódus → socket objektumot ad vissza (socketType osztály)

```
socket("family", "type")
```

vagy

```
socket("family", "type"  , "protocol")
```

AF → address family

family lehet: AF_UNIX → azonos számítógépen futó programok között
vagy AF_INET → különböző gépeken futó programok között

type lehet: SOCK_STREAM → kapcsolat orientált (pl. TCP)
vagy SOCK_DGRAM → datagram, kapcsolat nélküli (pl. UDP)
vagy egyéb, (SOCK_RAW, SOCK_RDM, SOCK_SEQPACKET, …)

protocol → nem kötelező megadni, csak a nyers típusú csatornáknál kell

```
# 1. mintaprogram

import socket
csat = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

11.4. Socket modul

Socket objektumok metódusai

bind(address) → csatorna kötése egy címhez (hálózati cím, port cím)

close() → csatorna lezárása

listen(max_kapcsolat) → figyeli a kapcsolat iránti kéréseket (szerver)
max_kapcsolat >=1

connect(address) → kapcsolódik egy címhez (kliens)
address → (IP cím, port cím)

accept() → fogadja a kapcsolatot (szerver)
visszaad → egy csatorna objektumot (kliens),
amelyen keresztül az átvitel elvégezhető,
→ és a kliens csatorna címét

recv(bufsize) → adatok fogadása a csatornáról → karakterláncot ad vissza ?
bufsize → puffer méret (max.)

send(bytes) → adatok küldése a csatornára
→ visszaadja az elküldött adatok méretét !

recvfrom(bufsize) → adatok fogadása a csatornáról, de az adatok mellett a küldő címét is visszaadja → (bytes, address)

sendto(bytes,flags,address) → adatok küldése a csatornára szintén, de a címet is most adjuk meg ! → nem kell hogy előtte kiépüljön a kapcsolat ! (UDP)

11.5. Minta programok

TCP kiszolgáló

```
# 2. mintaprogram   TCP kiszolgáló

import socket as soc

port = 12345
szerver_cim = ('',port)
szerver_csat = soc.socket(soc.AF_INET, soc.SOCK_STREAM)
puff = 500
szerver_csat.bind(szerver_cim)
szerver_csat.listen(3)

while 1:
        print('Kiszolgáló kész kapcsolatra')
        kliens_csat,kliens_cim = szerver_csat.accept()      # szerver vár csatlakozásra
        print(' Kapcsolódott a',kliens_cim)
        print('adatfogadásra kész')
        while 1:
                adatok_be = kliens_csat.recv(puff)          # beérkező adatok fogadása
                if not adatok_be:
                        print('ügyfél zárta a kapcsolatot')
                        break
                print("az ügyfél ezt küldte: \n",adatok_be)
                kistr = 'Halló Ügyfél!'
                adatok_ki = bytes(kistr,'utf-8')            # string átalakítása byte formába
                kliens_csat.send(adatok_ki)
                print(' A kiszolgáló  várja a további adatokat')
        kliens_csat.close()
szerver_csat.close()
```

11.6. Minta programok

TCP ügyfél

```
# 3. mintaprogram   TCP kliens
import socket as soc
port = 12345
szerver_cim = ('localhost',port)
kliens_csat = soc.socket(soc.AF_INET, soc.SOCK_STREAM)
puff = 500
kliens_csat.connect(szerver_cim)              #  csatlakozás a szerverhez
while 1:
        print('Kapcsolódás ok')
        kistr = input('mit küldjünk ?')
        if not kistr:
                print('nincs adat, kapcsolat vége !')
                break
        adatok_ki = bytes(kistr,'utf-8')      # string átalakítása byte formába
        kliens_csat.send(adatok_ki)           # adatok küldése szervernek
        adatok_be = kliens_csat.recv(puff)    # adatok fogadása szervertől
        if not adatok_be:
                print('nincs bejövő adat !')
                break
        print(' A kiszolgáló  ezt küldte:')
        print(adatok_be)
kliens_csat.close()
```

11.7. Minta programok

UDP kiszolgáló

```
# 4. mintaprogram  UDP kiszolgáló

import socket as soc

port = 12345
szerver_cim = ('',port)
szerver_csat = soc.socket(soc.AF_INET, soc.SOCK_DGRAM)
puff = 500
szerver_csat.bind(szerver_cim)

while 1:
        print('Kiszolgáló kész kapcsolatra')
        adatok_be,kliens_cim = szerver_csat.recvfrom(puff)
        print(' Kapcsolódott a',kliens_cim)
        bestr= str(adatok_be,'utf-8')                  # beérkező byte formátum átalakítása stringbe
        print("az ügyfél ezt küldte: \n",bestr)
        kistr = 'Adatok megérkeztek!'
        adatok_ki = bytes(kistr,'utf-8')                       # string átalakítása byte formába
        szerver_csat.sendto(adatok_ki,kliens_cim)
szerver_csat.close()
```

11.8. Minta programok

UDP ügyfél

```
# 5. mintaprogram   UDP kliens
import socket as soc
port = 12345
szerver_cim = ('localhost',port)
kliens_csat = soc.socket(soc.AF_INET, soc.SOCK_DGRAM)
puff = 500
while 1:
        kistr = input('mit küldjünk ?')
        if not kistr:
                print('nincs adat, kapcsolat vége !')
                break
        adatok_ki = bytes(kistr,'utf-8')                    # string átalakítása byte formába
        kliens_csat.sendto(adatok_ki, szerver_cim)
        adatok_be,szerver2_cim = kliens_csat.recvfrom(puff)
        if not adatok_be:
                print('nincs bejövő adat !')
                break
        print(' A kiszolgáló  ezt küldte:')
        bestr= str(adatok_be,'utf-8')               # beérkező byte formátum átalakítása stringbe
        print(bestr)
kliens_csat.close()
```