Programozás Python nyelven 4.

XII. Grafikus felület, Kivy
XIII. Tömbök, NumPy
XIV. Matplotlib
XV. Pygame

12.1. Kivy

Kivy telepítése

A Kivy library segítségével grafikus felületet tudunk létrehozni (mint a Tkinter), de modernebb felület, érintőképernyős eszközöknél is működik → Windows, Linux, macOS mellett Android vagy iOS operációs rendszereken is futtatható programokat lehet készíteni.
Nem része az alap Python telepítésnek, külön fel kell telepíteni !

Telepítése a Python pip csomagkezelőjével: pip install kivy
vagy pip install "kivy [base]" kivy_examples
vagy pip install "kivy [full]"

Telepítése Linux saját csomagjaként: általában python-kivy vagy python3-kivy és python-kivy-examples néven

Telepítése Android rendszeren: python-for-android csomag
Telepítése IOS rendszeren: kivy-ios csomag

Kivy moduljai

- kivy.app → alap alkalmazás osztály (App)
- kivy.uix → a grafikus felület elemeit (widgets) és elrendezés kezelőket (layouts) tartalmazza
- kivy.graphics → rajzolás
- kivy.lang → kv leíró nyelvvel kapcsolatos dolgok
- kivy.clock → ütemezők

...

12.2. Kivy

Fontosabb widgetek (grafikus komponensek)

- Button (nyomógomb) → utasítás végrehajtása, alprogram meghívása
- Label (címke) → információ kiírása
- TextInput (adat beviteli mező) → szöveg bekérése felhasználótól, több soros is lehet, kiíratásra is jó
- Image (kép) → kép megjelenítése
- CheckBox (jelölő négyzet) → vagy lehet választómező (RadioButton) is
- ToggleButton (kapcsológomb) → két állapot kapcsolására mint a Button + CheckBox
- Switch (kapcsoló) → két állapot kapcsolására hasonló mint a CheckBox
- Spinner (lista) → választás legördülő listából
- Slider (csúszka) → érték választás csúszkával
- ProgresBar → folyamat haladásának kijelzése
- Video → videó fájl megjelenítése

12.3. Kivy program

Kivy program felépítése

```
# 1. mintaprogram, ablak egy címkével
from kivy.app import App                    # App → alkalmazás (ablak)
from kivy.uix.label import Label            # Label importálása

class MainApp(App):                         #  egy saját App osztály létrehozása
    def build(self):                        # 'build' metódus felel meg a konstruktornak !
        label = Label(text='Hello !')           #  egy címke létrehozása
        return label

app = MainApp()                             #  fő ablak/alkalmazás példány létrehozása
app.run()                                   #  fő ablak esemény ciklusának indítása
```

App osztály

- A kivy programunk alaposztályát az 'App' osztályból kell származtatnunk.
- ezt kell példányosítani és futtatni. (utolsó két sor)

12.4. Kivy program

build() metódus
- a saját App osztályunkban ez felel meg lényegében a konstruktornak → ez fut először amikor az App osztályunk példányosítódik.
- ebben kell megtennünk a kezdeti beállításokat
- és itt kell elindítani a widgetek struktúrájának létrehozását → persze bonyolultabb esetben ezt célszerű külön osztályban megtenni
- a metódusnak a létrehozott alap (root) widgetet kell visszaadnia ! (return után)

run() metódus
- egy fő metódus, lényegében egy létrehozott grafikus alkalmazás/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

12.5. Kivy program

```
# 2. mintaprogram

from kivy.app import App                    # App → alkalmazás (ablak)
from kivy.uix.button import Button          # Button importálása

class MainApp(App):                         #  egy saját App osztály létrehozása
    def build(self):                        # ’build’ metódus felel meg a konstruktornak !
        butt = Button(text='Hello !',color='red',font_size=40)   #  egy nyomógomb létrehozása
        return butt

if __name__ == '__main__':                  #  csak akkor indítjuk, ha ez a fájl lett indítva
    app = MainApp()                         #  fő ablak/alkalmazás példány létrehozása
    app.run()                               #  fő ablak esemény ciklusának indítása
```

12.6. Widget

Widget létrehozása

A megfelelő modul megfelelő osztályát kell importálni, majd példányosítani. Ekkor lehet bizonyos tulajdonságait megadni.
pl.

```
from kivy.uix.label import Label                    # Label importálása
from kivy.uix.button import Button                  # Button importálása
label1 = Label(text='Hello !')                      #  Label példányosítása
butt = Button(text='Hello !',color='red',font_size=40)          #  Button példányosítása
```

Widgetek struktúrája

A widgeteket fa-szerkezetben lehet létrehozni, van egy első, alap (root) widget, ezen tudunk elhelyezni további widgeteket (children), majd azokon szintén, …

add_widget() → widget hozzáadása másikhoz (mint child)
szülőwidget.add_widget(gyermekwidget)

remove_widget() → widget eltávolítása
szülőwidget.remove_widget(gyermekwidget)

clear_widgets() → összes gyermek (children) widget törlése
szülőwidget.clear_widgets()

12.7. Widget

Relatív pozíció és méret

A pos_hint és size_hint jellemzőkkel relatívan adhatjuk meg egy widget pozícióját és méretét, de elrendezéskezelőtől is függ hogy melyik használható.

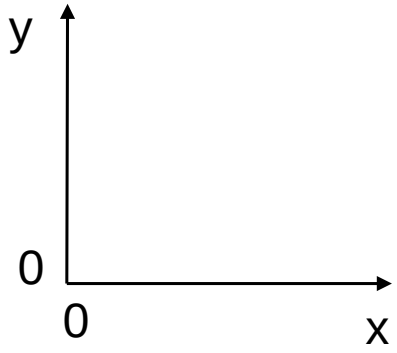
size_hint = (x, y) → relatív méret megadása, (vízsz. , függ.) x, y 0 – 1 közötti szám
ha nem adjuk meg → (1,1)

pl. size_hint = (.5, .5) → a szülő-widget méretének fele lesz mindkét irányban a méret !

!! de pontos értelmezése függ a többi gyermek widgettől is,
és az alkalmazott layout-tól is (később) !!

pos_hint = {'x':x, 'y':y} → relatív pozíció megadása, x, y → 0 – 1 közötti szám

pl. pos_hint = {'x':0.4, 'y':0.2} → a widget kezdő pozíciója (bal-lent) a szülő-widget vízsz. méretének 40 %-ánál (0.4) és a függ. méretének 20 %-ánál (0.2)



pos_hint = {'center_x':x, 'center_y':y} → relatív pozíció megadása ez is, de most a középpontoké

pos_hint = {'right':x} → relatív pozíció, a widget jobboldalának relatív pozíciója

pos_hint = {'top':y} → relatív pozíció, a widget tetejének relatív pozíciója

12.8. Widget-ek elrendezése

Elrendezéskezelők (Layouts)

Speciális widgetek, ezek nem jelennek meg, hanem a hozzájuk adott gyermek widgeteket (children) lehet segítségükkel elrendezni, pozíciókat, méreteket megadni …
A kivy.uix modulban találhatóak.Többféle van →

- BoxLayout → sorban, egymás után helyezi el a gyermek widgeteket (vízszintesen vagy függőlegesen)
- GridLayout → mátrixszerűen helyezi el a gyermek widgeteket sorok, oszlopok számát meg lehet adni (az egyiket muszáj!)
- FloatLayout → a gyermek widgetek tetszőleges pozícióban és méretben tudja elhelyezni
- AnchorLayout → a gyermek widgetek pozícióit a kerethez képest tudjuk megadni elhelyezni (top, bottom, left, right, center)
  pl. anchor_x='right', anchor_y='bottom'
- RelativeLayout → hasonló mint a FloatLayout
- StackLayout

12.9. Widget-ek elrendezése

BoxLayout

```
# 3. mintaprogram

from kivy.app import App                          # App → alkalmazás (ablak)
from kivy.uix.label import Label                  # Label importálása
from kivy.uix.boxlayout import BoxLayout          # BoxLayout importálása

class MainApp(App):                               #  egy saját App osztály létrehozása
    def build(self):                              # 'build' metódus felel meg a konstruktornak !
        box1 = BoxLayout ()                       # BoxLayout létrehozása, ő lesz a root widget
        lab1 = Label(text='Hello !',color='red',font_size=60)          #  egy cimke
        lab2 = Label(text='Mi a helyzet ?',color='blue',font_size=40)  #  másik címke
        box1.add_widget(lab1)                     #  lab1 hozzáadása mint gyermek widget box1-hez
        box1.add_widget(lab2)                     #  lab2 hozzáadása mint gyermek widget box1-hez
        return box1                               # a root widget visszaadása

if __name__ == '__main__':                        #  csak akkor indítjuk, ha ez a fájl lett indítva
    app = MainApp()                               #  fő ablak/alkalmazás példány létrehozása
    app.run()                                     #  fő ablak esemény ciklusának indítása

# BoxLayout() így példányosítva → vízszintesen rendez el
# de megadható az irány →  BoxLayout(orientation='vertical') → így függőlegesen
```

12.10. Widget-ek elrendezése

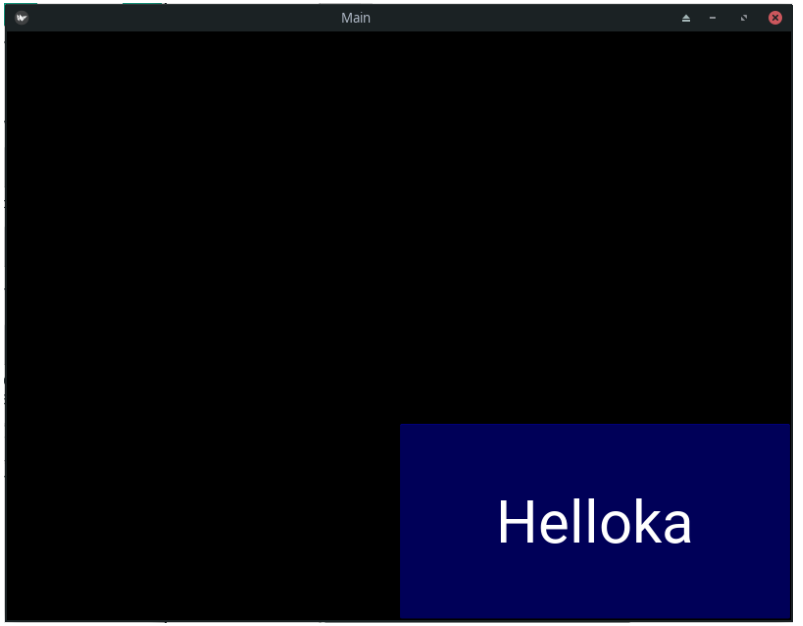
BoxLayout

```
# 4. mintaprogram
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
class MainApp(App):
    def build(self):
        box1 = BoxLayout (orientation='vertical')
        butt1 = Button(text='Hello !',color='red',font_size=40,
                size_hint=(.5, 1/3),background_color='blue')
        butt2 = Button(text='Mi a ',color='blue',font_size=70)
        butt3 = Button(text=' helyzet ?',color='green',font_size=70)
        box1.add_widget(butt1)
        box1.add_widget(butt2)
        box1.add_widget(butt3)
        return box1

if __name__ == '__main__':
    app = MainApp()
    app.run()
```

12.11. Widget-ek elrendezése

AnchorLayout

```
# 5. mintaprogram
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.anchorlayout import AnchorLayout

class MainApp(App):
        def build(self):
                lay1 = AnchorLayout(anchor_x='right', anchor_y='bottom')
                butt1 = Button(text='Helloka', size_hint=(.5, 1/3),font_size=60, background_color='blue')
                lay1.add_widget(butt1)
                return lay1

if __name__ == '__main__':
        app = MainApp()
        app.run()
```



12.12. Widget-ek elrendezése

FloatLayout

```
# 6. mintaprogram
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.floatlayout import FloatLayout

class MainApp(App):
        def build(self):
                lay1 = FloatLayout ()
                butt1 = Button(text='Hello !',color='red',font_size=50,
                                size_hint=(1/3, 1/4),pos_hint={'x': 0,'y':0})
                butt2 = Button(text='Mi a ',color='blue',font_size=50,
                                size_hint=(1/3, 1/4),pos_hint={'x': 1/3,'y':1/3})
                butt3 = Button(text=' helyzet ?',color='green',font_size=50,
                                size_hint=(1/3, 1/4),pos_hint={'x': 2/3,'y':2/3})
                lay1.add_widget(butt1)
                lay1.add_widget(butt2)
                lay1.add_widget(butt3)
                return lay1

if __name__ == '__main__':
        app = MainApp()
        app.run()
```

12.13. Widget properties

Properties

Tulajdonságok

- Az objektum ill. osztály attributumok megadása Kivy widgetek esetén kicsit másképpen történik (nem kell konstruktoron belül) →

pl.

```
class MyWidget(Widget):
    text = StringProperty('valami')        # property, objektum attributum
    max = NumericProperty(10)              # property szintén
```

- Többféle property típus van (különböző adattípusokra):

NumericProperty, StringProperty, ListProperty, ObjectProperty, BooleanProperty, BoundedNumericProperty, OptionProperty, ReferenceListProperty, AliasProperty, DictProperty, VariableListProperty, ConfigParserProperty, ColorProperty

A kivy.properties modulban vannak !
A widgetek meglévő tulajdonságai is jellemzően ilyenek, de mi is hozhatunk létre újat.

- Alapból osztály attributum, és példányosítás után objektum attributum.

- ha megváltoznak → eseményt generálnak !
  → kiváltott esemény: on_property-név
  pl. on_text

12.14. Események

Events

Általuk kapunk a bemenetektől információt (egér valamelyik gombjának lenyomása, érintőképernyő megérintése valamelyik ponton, ….), illetve a programunk jelezhet (valamilyen widget esemény, vagy egy property megváltozott), és időzített, ütemezett esemény is lehet (pl. bizonyos időnként egy függvény meghívása).
Tehát többféle esemény van:

- input events (motion events) → Widget eseményeket generálnak
  - touch event
  - no touch event
- clock events (időzítés, ütemezés)
- property events
- custom events

Alaposztály: EventDispatcher

Widget események:

on_property-név → widget property megváltozásakor
pl. on_size on_text ...

on_press → button lenyomásakor

…

Clock események:

Clock.schedule_interval(fv,idő) → meghatározott időnként meghív egy függvényt
Clock.schedule_once(fv,idő) → meghatározott idő múlva meghív egy függvényt

12.15. Események

Touch events

Ezen eseményeket lekezelő metódusoknak átadódik az objektum és az érintés koordinátái →

on_touch_down(self,touch) → érintéskor
on_touch_move(self,touch) → érintés után elmozdulás
on_touch_up(self,touch) → érintés megszünésekor

Clock events

Clock objektum - a clock modulból - kell hozzá.
- Ütemezett, időzített függvény hívás.

Clock.schedule_interval(fv,idő) → meghatározott időnként meghív egy függvényt

```
pl.  def my_callback(dt):
         ....
     event = Clock.schedule_interval(my_callback,0.1)    # 0,1 másodpercenként
```

clock event törlése →

Clock.unschedule(event) vagy event.cancel()
vagy callback függvény 'False' értéket ad vissza !

- Egyszer ütemezett esemény

Clock.schedule_once(fv,idő) → meghatározott idő múlva meghív egy függvényt

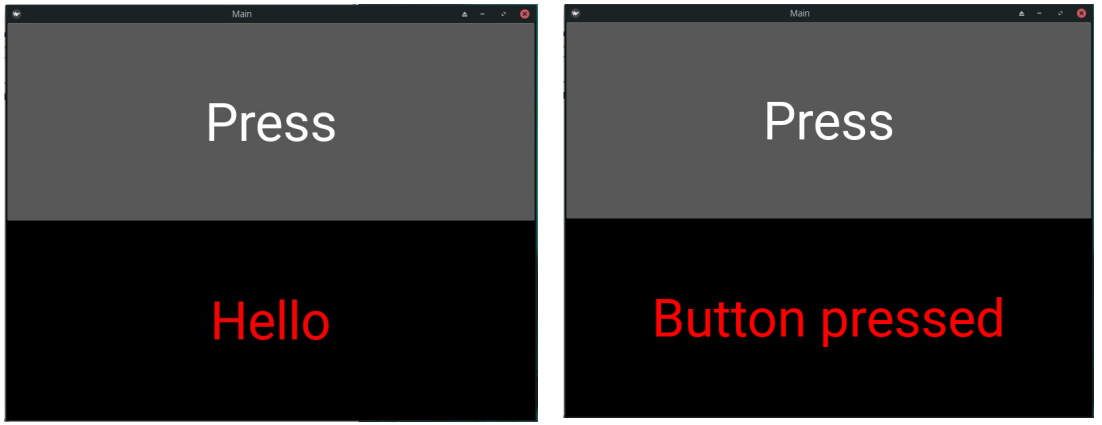
- Trigger esemény

trigger=Clock.create_trigger(fv) → beállítja a meghívandó függvényt
...
trigger() → biztosan csak egyszer meghívja a függvényt

12.16. Események

on_press + BoxLayout

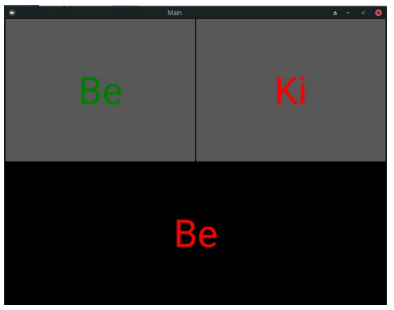
```
# 7. mintaprogram, nyomógomb lenyomására megváltoztatjuk a címke feliratát
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout

class Helloka(BoxLayout):                       # most már külön osztályban hozzuk létre
        def __init__(self,**kwargs):            # a widgeteket
                super(Helloka,self).__init__(**kwargs)
                self.orientation='vertical'
                self.butt1 = Button(text='Press', font_size=80, on_press=self.doing)
                self.lab1 = Label(text='Hello', font_size=80, color ='red')
                self.add_widget(self.butt1)
                self.add_widget(self.lab1)

        def doing(self,ev):             # ez a metódus van az on_press eseményhez rendelve
                self.lab1.text ='Button pressed'

class MainApp(App):
        def build(self):
                hello = Helloka()
                return hello

if __name__ == '__main__':
        app = MainApp()
        app.run()
```

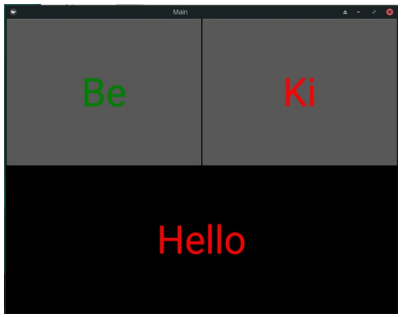


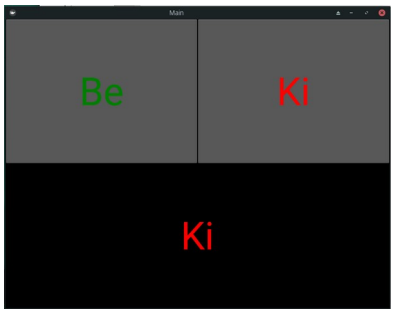
12.17. Események

on_press + FloatLayout

```
# 8. mintaprogram, nyomógombok lenyomására megváltoztatjuk a címke feliratát

from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.floatlayout import FloatLayout

class Helloka(FloatLayout):
    def __init__(self,**kwargs):
        super(Helloka,self).__init__(**kwargs)
        self.butt1 = Button(text='Be', font_size=80, color='green',
                            size_hint=(0.5, 0.5),pos_hint={'x': 0,'y':0.5}, on_press=self.do_on)
        self.butt2 = Button(text='Ki', font_size=80, color='red',
                            size_hint=(0.5, 0.5),pos_hint={'x': 0.5,'y':0.5}, on_press=self.do_off)
        self.lab1 = Label(text='Hello', font_size=80, color ='red',
                            size_hint=(1, 0.5),pos_hint={'x': 0,'y':0})
        self.add_widget(self.butt1)
        self.add_widget(self.butt2)
        self.add_widget(self.lab1)

    def do_on(self,ev):
        self.lab1.text ='Be'

    def do_off(self,ev):
        self.lab1.text ='Ki'

class MainApp(App):
    def build(self):
        return Helloka()

if __name__ == '__main__':
    MainApp().run()
```

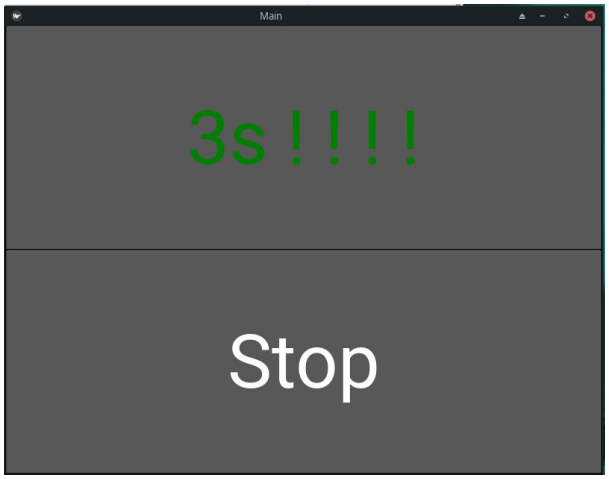






12.18. Események

Clock_schedule_interval

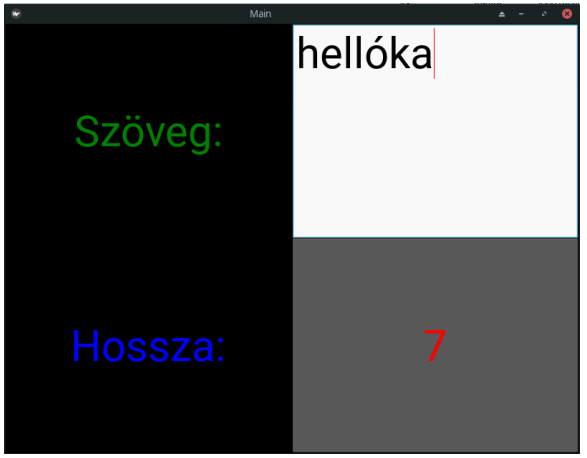



```
# 9. mintaprogram, 3 másodpercenként módosítja a gomb feliratát
( hozzáad egy plusz felkiáltó jelet)

from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
from kivy.clock import Clock

class HelloLabel(BoxLayout):
        def __init__(self,**kwargs):
                super(HelloLabel,self).__init__(**kwargs)
                self.orientation='vertical'
                self.butt1 = Button(text='3s', font_size=100, color='green')
                self.butt2 = Button(text='Stop', font_size=100, on_press=self.doing)
                self.add_widget(self.butt1)
                self.add_widget(self.butt2)

        def doing(self,ev):                 # ez a metódus leállítja az ütemezést
                app.event.cancel()

        def my_callback(self,dt):           # callback függvény, ez a metódus hívódik meg ütemezetten
                self.butt1.text+= " !"

class MainApp(App):
        def build(self):
                helloka = HelloLabel()
                self.event = Clock.schedule_interval(helloka.my_callback,3)      # ütemezés megadása
                return helloka

if __name__ == '__main__':
        app = MainApp()
        app.run()
```

12.19. Események

on_text + TextInput

```
# 10. mintaprogram, számolja és kiírja a szövegmezőbe beírt szöveg hosszát
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.floatlayout import FloatLayout

class Szamolok(FloatLayout):
        def __init__(self,**kwargs):
                super(Szamolok,self).__init__(**kwargs)
                self.lab1 = Label(text='Szöveg:', font_size=60, color='green', size_hint=(.5,.5), pos_hint={'x': 0,'y':0.5})
                self.text1 = TextInput(text='', font_size=60, size_hint=(.5,.5), pos_hint={'x': 0.5,'y':0.5})
                self.lab2 = Label(text='Hossza:', font_size=60, color ='blue', size_hint=(.5,.5), pos_hint={'x': 0,'y':0})
                self.butt1 = Button(text='0', font_size=60, color='red', size_hint=(.5,.5), pos_hint={'x': 0.5,'y':0})
                self.text1.bind(text=self.on_text)    # widget kötése az eseményt lekezelő metódushoz
                self.add_widget(self.lab1)
                self.add_widget(self.text1)
                self.add_widget(self.lab2)
                self.add_widget(self.butt1)

        def on_text(self,ins,val):              # az eseményt lekezelő metódus
                hossz=len(self.text1.text)      # beírt szöveg hosszának kiszámítása
                self.butt1.text = str(hossz)     # a hossz kiírása a nyomógombra

class MainApp(App):
        def build(self):
                return Szamolok()

if __name__ == '__main__':
        MainApp().run()
```



12.20. Kivy nyelv

Kv

Létezik Kivyben egy leíró nyelv (dizájn nyelv)→ Kv
A grafikus felület (widgetek elrendezése, tulajdonságaik) leírása lehetséges használatával.
(És igazából ennél egy picit több is!)
Így kicsit egyszerűbb és átláthatóbb a kód ezen része, és Pythonban csak a logikai részt kell megírni.

Lehet külön fájlban → .kv kiterjesztéssel
vagy magában a python fájlban is sztringként.

Betöltése → Builder osztály kell hozzá a 'lang' modulból

```
from kivy.lang import Builder

Builder.load_string('''
…
''')
```

vagy

```
Builder.load_file('path/to/file.kv')
```

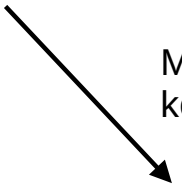
12.21. Kivy nyelv

Kv alapok

Ez egy kv leírás

```
<BoxLayout>:
    id: box1
    orientation: 'vertical'
    Label:
        id: lab1
        text: "Hello Kivy!"
        font_size: 60
        size_hint: .5, .5
    Button:
        id: but1
        text: "OK"
        font_size: 120
        color: 'red'
```

Jellemzői:
- a widgetek hierarchiája behúzásokkal jelölve
- egy adott widget tulajdonságai szintén behúzással
- a tulajdonság-érték párok között '=' helyett ':'
- az 'id' megadása nem feltétlen szükséges, akkor kell ha a widget tulajdonságait később akarjuk lekérdezni vagy módosítani (mert ezzel tudunk hivatkozni rá!)



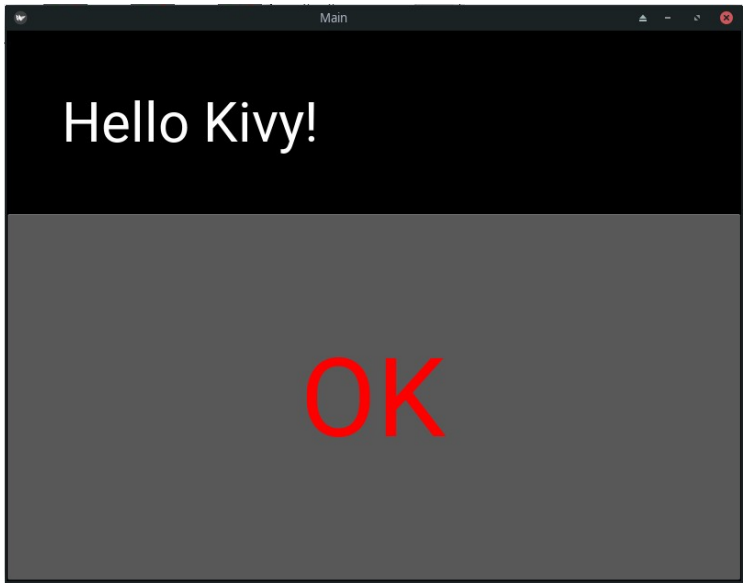
Megfelel ennek a python kódnak

```
box1=BoxLayout(orientation='vertical')
lab1 = Label(text='Hello Kivy!',font_size=60, size_hint=(.5, .5))
box1.add_widget(lab1)
but1 = Button(text='OK',font_size=120,color='red')
box1.add_widget(but1)
```

12.22. Kivy nyelv

Kv alapok

```
# 11. mintaprogram

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder

Builder.load_string("""
<HelloLabel>:
        orientation: 'vertical'
        Label:
                text: "Hello Kivy!"
                font_size: 60
                size_hint: .5, .5
        Button:
                text: "OK"
                font_size: 120
                color: 'red'
""")

class HelloLabel(BoxLayout):
        pass        # Kv  nyelven adjuk meg

class MainApp(App):
        def build(self):
                return HelloLabel()

if __name__ == '__main__':
        app = MainApp()
        app.run()
```



12.23. Kivy nyelv

Kv alapok

```
# 12. mintaprogram (a 8. mintaprogram kv nyelvvel)

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder

Builder.load_string("""
<Helloka>:
        lab1: lab1   # kell, ha pythonból akarjuk látni
        Button:
                text: "BE"
                font_size: 80
                color: 'green'
                size_hint: .5, .5
                pos_hint: {'x': 0,'y':0.5}
                on_press: root.do_on()
        Button:
                text: "KI"
                font_size: 80
                color: 'red'
                size_hint: .5, .5
                pos_hint: {'x': 0.5,'y':0.5}
                on_press: root.do_off()
        Label:
                id: lab1
                text: "Hello!"
                font_size: 80
                color: 'red'
                size_hint: 1, .5
                pos_hint: {'x': 0,'y':0}
""")

# folytatás

class Helloka(FloatLayout):

        def do_on(self):
                self.lab1.text ='Be'

        def do_off(self):
                self.lab1.text ='Ki'

class MainApp(App):

        def build(self):
                return Helloka()

if __name__ == '__main__':
        app = MainApp()
        app.run()


#  root.do_on()  root.do_off() →  mert  'Helloka' a root
#  lehetnének ezek a metódusok a 'MainApp' osztályban is
# → de akkor app.do_on()  app.do_off()  néven lehetne
# hivatkozni rájuk
```

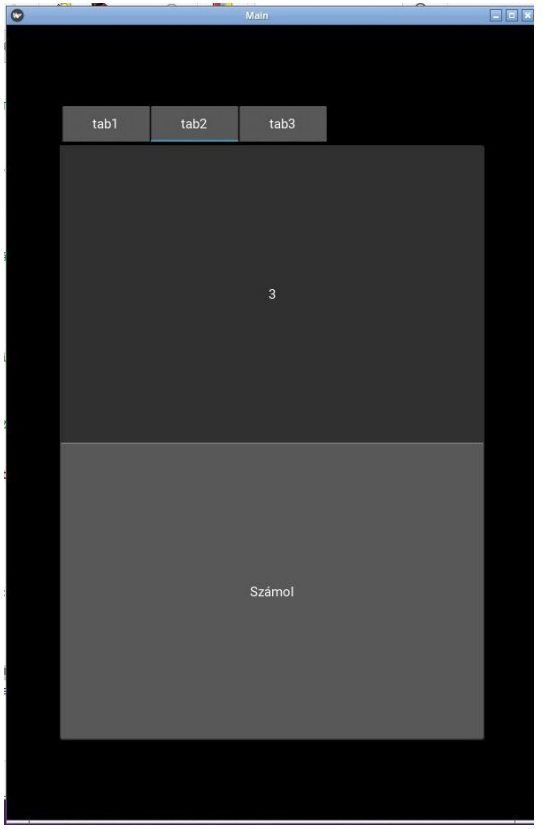
12.24. Egyéb Widgetek

TabbedPanel

Többoldalas panel, widgetek tárolására

```
# 13. mintaprogram
from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.lang import Builder
from kivy.properties import NumericProperty

Builder.load_string("""
<Test>:
        lab2:lab2
        size_hint: .8, .8
        pos_hint: {'center_x': .5, 'center_y': .5}
        do_default_tab: False
        TabbedPanelItem:
                text: 'tab1'
                Label:
                        text: 'First Tab'
        TabbedPanelItem:
                text: 'tab2'
                BoxLayout:
                        orientation: 'vertical'
                        Label:
                                id: lab2
                                text: '0'
                        Button:
                                text: 'Számol'
                                on_press: root.szamol()

        TabbedPanelItem:
                text: 'tab3'
                TextInput:
                        text: 'Hello in the third Tab ! '
""")
```

```
# folytatás
class Test(TabbedPanel):
        szam=NumericProperty(0)
        def szamol(self):
                self.szam+=1
                self.lab2.text =str(self.szam)

class MainApp(App):
        def build(self):
                return Test()

if __name__ == '__main__':
        app = MainApp()
        app.run()
```



12.25. Egyéb Widgetek

Slider (csúszka)

```
# 14. mintaprogram

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder
from kivy.properties import NumericProperty

Builder.load_string("""

""")

class Szamolo(FloatLayout):

        szam=NumericProperty(0)

        def szamol(self):
                self.szam+=self.sl1.value
                self.lab2.text =str(self.szam)

class MainApp(App):

        def build(self):
                return Szamolo()

if __name__ == '__main__':
        app = MainApp()
        app.run()

#  a csúszka által beállított számot a gomb
# lenyomására hozzáadja a 'szam' property
# értékéhez, és kiírja a lenti Label-re is
# a felső Label a csúszka aktuális értékét
# mutatja
```

```
# Kv  leírás
<Szamolo>:
        lab2: lab2
        sl1: sl1
        Button:
                text: "Számol"
                font_size: 80
                size_hint: .6, .2
                pos_hint: {'x': 0.2,'y':0.4}
                on_press: root.szamol()
        Slider:
                id: sl1
                size_hint: .6, .1
                pos_hint: {'x': 0.2,'y':0.8}
                min: -10
                max: 10
                value: 1
                step: 1
        Label:
                size_hint: .2, .2
                pos_hint: {'x': 0.4,'y':0.7}
                text: str(sl1.value)
                font_size: 50
        Label:
                id: lab2
                text: "0"
                font_size: 80
                color: 'red'
                size_hint: 1, .3
                pos_hint: {'x': 0,'y':0}
```

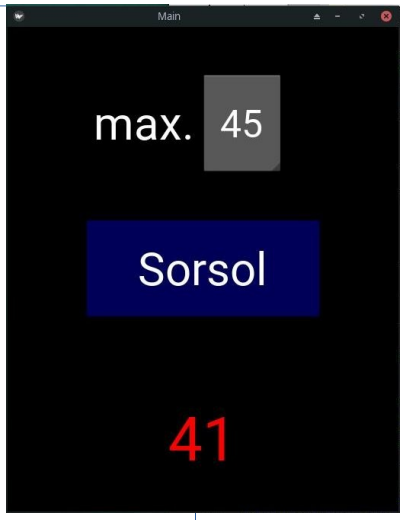
12.26. Egyéb Widgetek

Spinner (legördülő lista)

```
# 15. mintaprogram

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder
import random    #  véletlen számhoz

Builder.load_string("""

""")

class Szamolo(FloatLayout):
        def szamol(self):
                maxszam=int(self.sp1.text)
                number=random.randint(1,maxszam)
                self.lab2.text =str(number)

class MainApp(App):
        def build(self):
                return Szamolo()

if __name__ == '__main__':
        app = MainApp()
        app.run()

#  1-maxszam közötti véletlen szám generálása
#  maxszam-ot  a spinner-rel tudjuk módosítani
# szám sorsolása → 'Sorsol' nyomógomb
# megnyomására, vagy ha a spinner értéke
# megváltozik ! (ont_text)
```

```
# Kv  leírás
<Szamolo>:
        lab2: lab2
        sp1: sp1
        Label:
                size_hint: .3, .2
                pos_hint: {'x': 0.2,'y':0.7}
                text: 'max.'
                font_size: 60
        Spinner:
                id: sp1
                size_hint: .2, .2
                pos_hint: {'x': 0.5,'y':0.7}
                text: "6"       #  a kiválasztott érték
                values:'4','6','12','20','35','45','50','80','90'
                font_size: 50
                on_text: root.szamol()
        Button:
                text: "Sorsol"
                font_size: 60
                background_color: 'blue'
                size_hint: .6, .2
                pos_hint: {'x': 0.2,'y':0.4}
                on_press: root.szamol()
        Label:
                id: lab2
                text: "Press Sorsol !"
                font_size: 80
                color: 'red'
                size_hint: .6, .3
                pos_hint: {'x': 0.2,'y':0}
```

ezekből
lehet
választani



12.27. Egyéb Widgetek

ToggleButton

```
# 16. mintaprogram

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder
from kivy.properties import NumericProperty
from kivy.clock import Clock  # ütemezés

Builder.load_string("""

""")
class Szamolo(FloatLayout):
    szam=NumericProperty(0)

    def szamol(self,dt):
        self.szam+=self.sl1.value
        self.lab3.text =str(self.szam)

    def kapcs(self):
        if self.tb1.state=='down':  # ha le van nyomva
            self.event = Clock.schedule_interval(self.szamol,1)
        else:          # ha tb1.state==’normal’
            self.event.cancel()

class MainApp(App):

    def build(self):
        return Szamolo()

if __name__ == '__main__':
    app = MainApp()
    app.run()
```

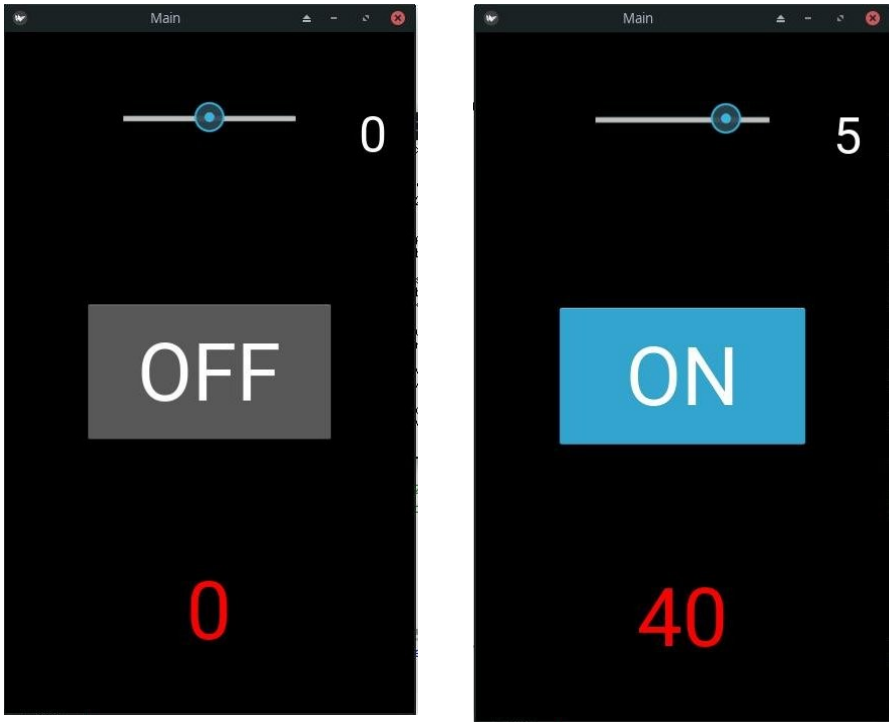
```
# Kv  leírás
<Szamolo>:
    lab3: lab3
    sl1: sl1
    tb1: tb1
    Slider:
        id: sl1
        size_hint: .5, .15
        pos_hint: {'x': 0.25,'y':0.8}
        min: -10
        max: 10
        value: 1
        step: 1
    Label:
        size_hint: .2, .2
        pos_hint: {'x': 0.8,'y':0.75}
        text: str(sl1.value)
        font_size: 50
    ToggleButton:
        id:tb1
        text: "OFF" if tb1.state=='normal' else "ON"
        state: 'normal'
        font_size: 80
        size_hint: .6, .2
        pos_hint: {'x': 0.2,'y':0.4}
        on_state: root.kapcs()
    Label:
        id: lab3
        text: "0"
        font_size: 80
        color: 'red'
        size_hint: 1, .3
        pos_hint: {'x': 0,'y':0}
```

12.28. Egyéb Widgetek

ToggleButton

16. mintaprogram

- A ToggleButton két állapotú →
'state' jellemzője lehet: 'normal' és 'down'
- Felíratát (text) az állapotától függően módosítjuk,
itt ez most a Kv nyelven van megírva !
- Ha változik az állapota → on_state esemény,
ehhez lehet metódust kötni (itt a root.kapcs)
→ és ez fog elindítani ill. leállítani egy időzítést, amely
1 másodpercenként számoltat → a slider által beállított
számmal növeli a számlálót




```
# Kv  leírás
<Szamolo>:
        lab3: lab3
        sl1: sl1
        tb1: tb1
        Slider:
                id: sl1
                size_hint: .5, .15
                pos_hint: {'x': 0.25,'y':0.8}
                min: -10
                max: 10
                value: 1
                step: 1
        Label:
                size_hint: .2, .2
                pos_hint: {'x': 0.8,'y':0.75}
                text: str(sl1.value)
                font_size: 50
        ToggleButton:
                id:tb1
                text: "OFF" if tb1.state=='normal' else "ON"
                state: 'normal'
                font_size: 80
                size_hint: .6, .2
                pos_hint: {'x': 0.2,'y':0.4}
                on_state: root.kapcs()
        Label:
                id: lab3
                text: "0"
                font_size: 80
                color: 'red'
                size_hint: 1, .3
                pos_hint: {'x': 0,'y':0}
```

12.29. Kivy konfigurálása

Kivy config file

- Az user mappában (pl. linux esetén → */home/user-név/*) a kivy létrehoz egy .kivy mappát
- ezen mappában van egy config.ini fájl, ez tartalmaz nagyon sok beállítást, csoportosítva
  pl. [graphics] → grafikus beállítások
- ezen beállítások befolyásolják a futtatott kivy programok jellemzőit
- nem célszerű ezt a fájlt szerkeszteni ! →
- ezeket a jellemzőket kivy programból is lehet módosítani !
- kivy.config modul Config objektuma kell hozzá → Config.set() metódusokkal
  Config.set(csoport,jellemző,érték)

```
# a kivy program legelején  kell ezeket az utasításokat beírni !!
from kivy.config import Config
Config.set('graphics','width',500)      # Alkalmazás ablak szélessége 500 pixel
Config.set('graphics','height',800)     # Alkalmazás ablak magassága 800 pixel
Config.set('graphics','resizable',0)    # Alkalmazás ablak nem átméretezhető
Config.set('graphics','resizable',1)    # Alkalmazás ablak átméretezhető (ez az alapértelmezett)
```

Környezeti változók beállítása

- Az operációs rendszer környezeti változóit is lehet módosítani, a Kivy-vel kapcsolatosakat is.
A program elejére kell ezeknek is kerülnie.

```
import os
os.environ['KIVY_HOME'] = '/abcd/xyz'          # Kivy alapmappa, ahová a config.ini is kerül
os.environ['KIVY_NO_CONSOLELOG'] = '1'         # Kivy konzol log kikapcsolása
…
import kivy. ….
```

12.30. Rajzolás

Canvas

- „Festővászon” amelyre rajzolni tudunk. Minden widget-nek van ilyen rajzvászna !
- a kivy.graphics modul kell hozzá
- lehet használni Kv nyelven is, és Kv nyelv nélkül is !
- mindeg először kell megadni a canvast, és utána a többi gyermek widgetet !
- igazából 3-féle canvas van: canvas.before, canvas, canvas.after
  a canvas.before tartalma kerül legalulra, a canvas.after tartalma kerül legfelülre

Canvas fontosabb metódusai

Line(points=(x1,y1,x2,y2,…),width=...)
egyenes(ek) rajzolása, (x1,y1) ponttól (x2,y2), …. pontig, width → vonalvastagság

Rectangle(pos=(x1,y1), size=(dx,dy))
téglalap rajzolása, (x1,y1)→ bal alsó sarok, (dx,dy) → szélesség, magasság

Ellipse(pos=(x1,y1), size=(dx,dy))
ellipszis,kör rajzolása, (a köré rajzolt téglalap koordinátáit kell megadni)
(x1,y1)→ bal alsó sarok, (dx,dy) → szélesség, magasság

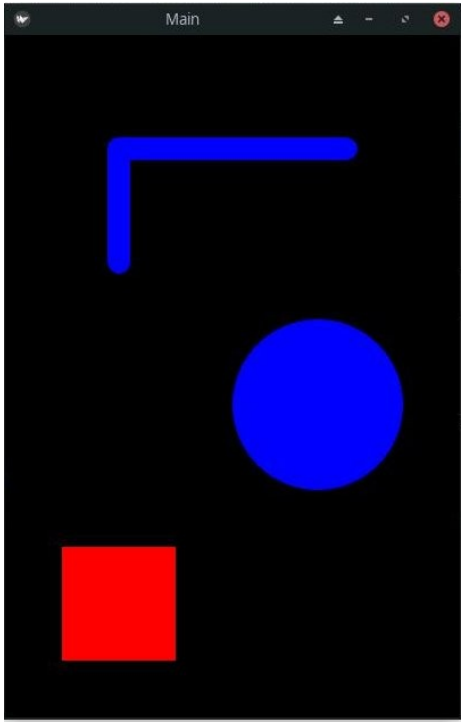
Color(r, g, b, mode='rgb')
szín megadása, mód: rgb vagy rgba, … (az értékek 0-1 között)

Clear() canvas törlése

12.31. Rajzolás

Canvas alapok

```
# 17. mintaprogram, Kv nyelv nélkül

from kivy.config import Config
Config.set('graphics','width',400)        # Alkalmazás ablak szélessége
Config.set('graphics','height',600)       # Alkalmazás ablak magassága

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Canvas, Color, Rectangle, Ellipse, Line

class Rajzolo(FloatLayout):
        def __init__(self,**kwargs):
                super(Rajzolo,self).__init__(**kwargs)
                with self.canvas:
                        Color(1, 0, 0, mode='rgb')
                        Rectangle(pos=(50,50), size=(100,100))
                        Color(0, 0, 1, mode='rgb')
                        Ellipse(pos=(200,200), size=(150,150))
                        Line(points=(100,400,100,500,300,500), width=10)

class MainApp(App):
        def build(self):
                return Rajzolo()

if __name__ == '__main__':
        app = MainApp()
        app.run()
```



12.32. Rajzolás

Canvas alapok

```
# 17. mintaprogram másképpen, Kv nyelven

from kivy.config import Config
Config.set('graphics','width',400)
Config.set('graphics','height',600)

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder

Builder.load_string("""



""")

class Rajzolo(FloatLayout):
        pass

class MainApp(App):
        def build(self):
                return Rajzolo()

if __name__ == '__main__':
        app = MainApp()
        app.run()
```

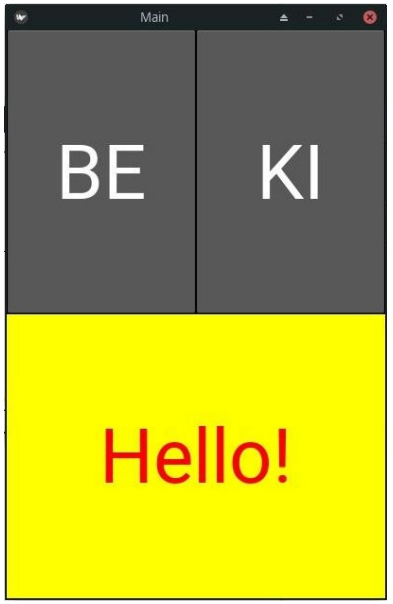
```
# Kv  leírás
<Rajzolo>:
        canvas:
                Color:
                        rgb: 1,0,0
                Rectangle:
                        pos: 50,50
                        size: 100,100
                Color:
                        rgb: 0,0,1
                Ellipse:
                        pos: 200,200
                        size: 150,150
                Line:
                        points: 100,400,100,500,300,500
                        width: 10
```

12.33. Rajzolás

Canvas alapok

18. mintaprogram
a 8. mintaprogram,csak a Label kapott háttérszínt

```
from kivy.config import Config
Config.set('graphics','width',400)
Config.set('graphics','height',600)

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.lang import Builder

Builder.load_string("""



""")

class Rajzolo(FloatLayout):
        def do_on(self):
                self.lab1.text ='Be'

        def do_off(self):
                self.lab1.text ='Ki'

class MainApp(App):
        def build(self):
                return Rajzolo()

if __name__ == '__main__':
        app = MainApp()
        app.run()
```




```
# Kv  leírás
<Rajzolo>:
        lab1: lab1
        Button:
                text: "BE"
                font_size: 80
                size_hint: .5, .5
                pos_hint: {'x': 0,'y':0.5}
                on_press: root.do_on()
        Button:
                text: "KI"
                font_size: 80
                size_hint: .5, .5
                pos_hint: {'x': 0.5,'y':0.5}
                on_press: root.do_off()
        Label:
                canvas.before:
                        Color:
                                rgb: 1,1,0
                        Rectangle:
                                pos: 0, 0
                                size: self.width, self.height

                id: lab1
                text: "Hello!"
                font_size: 80
                color: 'red'
                size_hint: 1, .5
                pos_hint: {'x': 0,'y':0}
```

13.1. NumPy

NumPy library

Numerical Python
- Tömbök (array) kezelésére szolgáló Python függvénykönyvtár → numpy modul
- Véletlenszámok generálása, valószínűségszámítás → numpy.random almodul
- Nagy mennyiségű adatok esetén gyorsabb, mintha listákat használnánk.
- Az array objektumok típusa → ndarray

Nem része az alap Python telepítésnek, külön fel kell telepíteni !

Telepítése a Python pip csomagkezelőjével: pip install numpy

NumPy modul

importálása → import numpy as np # általában 'np' néven szokás importálni

Több adat típust lehet használni, mint az alap Pythonban, típus jelölés 1 betűvel →

- i – egész
- b – logikai
- u – előjel nélküli egész
- f – lebegőpontos valós
- c – komplex (float)
- m – timedelta
- M – datetime
- O – objektum
- S – sztring
- U – unicode sztring
- V – fix memóriadarab más típusokhoz (void)

adat típust lehet pontosítani, pl. i4 → 4 byte-os egész

13.2. NumPy

Tömb létrehozása, használata

- Tömb létrehozása → array() függvénnyel

array([adat1, adat2, …])

pl.
```
import numpy as np
tomb1 = np.array([2,4,6,8,10])
```

- meg lehet adni az adat típust is → array([adat1, adat2, …], dtype= 'típus')
- Lehet több dimenziós is → array([[adat11, adat12, …],[adat21, adat22, ...]])
- a létrejött array objektum már csak egyféle adattípust fog tartalmazni !
  Ha olyan lista vagy tuple adódik át, amelyik különféle adattípusokat tartalmaz →
  egy közös adattípusra konvertálódnak !!
  ha megadtuk az adat típust, akkor arra próbálja meg konvertálni !

- A tömb elemeinek eléréséhez, módosításához ugyanúgy használható az indexelés, szeletelés, mint a listák, tuplek esetében → Az index 0-val kezdődik !!

pl.
```
import numpy as np
tomb1 = np.array([10,20,30,40,50,60,70])
tomb2 = np.array([ [2,4,6,8,10], [1,3,5,7,9] ])
print(tomb1[2])                                  # 30
print(tomb2[1,4])                                # 9
```

13.3. NumPy

Array attributumok, metódusok

- ndim → hány dimenziós a tömb
- dtype → tömb elemeinek típusa
- base → None, ha eredeti tömb,
  → egy tömböt ad vissza (az eredetit), ha csak egy hivatkozás egy tömbre
- shape → tömb elemeinek számát adja meg, dimenziónként tuple-ben
- astype('típus') → meglévő tömbről másolatot csinál, elemeinek konvertálásával más típusúvá
- copy() → meglévő tömbről másolatot csinál (saját, új adatok)
- view() → meglévő tömbre csinál egy új hivatkozást (nincsenek saját adatok)

pl.

```
import numpy as np
t1 = np.array([10,20,30,40,50,60,70])
print(t1.dtype)                    # int64
t2 = t1.astype('f')
print(t1)                          # [10 20 30 40 50 60 70]
print(t2)                          #  [10. 20. 30. 40. 50. 60. 70.]
print(t2.dtype)                    #  float32
t3 = np.array([ [2,4,6,8,10], [1,3,5,7,9] ])
t4 = t3.copy()
print(t3.shape)                    #  (2,5) → 2 sor, soronként 5 elem
```

13.4. NumPy

NumPy modul függvényei

- array([adat1, adat2, …]) vagy array((adat1, adat2, …))
  → tömb létrehozása, a megadott paraméter lehet lista, tupla, vagy más, tömbszerű adatsor

  adat típus megadás → array([adat1, adat2, …], dtype= 'típus')

- nditer(tömb) → meglévő tömb bejárása, visszaadja egyenként az elemeit

- ndenumerate(tömb) → meglévő tömb indexelt bejárása, visszaadja egyenként az elemeit az indexeikkel !

- concatenate((tömb1, tömb2, ...)) → meglévő tömbök összerakása 1 tömbbe

- where(tömb == érték) → 'érték' keresése meglévő tömbben, visszaadja az indexeket

- sort(tömb) → tömb elemeinek sorba rendezése ! DE egy új tömböt hoz létre → azt adja vissza rendezetten

pl.

```
import numpy as np
t1 = np.array(((10,20,30,40),(50,60,70,80)))
for x in np.nditer(t1):
        print(x)      # 10 20 30 40 50 60 70 80
for i,x in np.ndenumerate(t1):
        print(i,x)    # (0,0) 10  (0,1) 20 (0,2) 30 (0,3) 40
                      # (1,0) 50 (1,1) 60 (1,2) 70 (1,3) 80
talalt= np.where(t1 == 70)
print(talalt)         # (array([1], array([2])
t2 = np.array(['f','b','c','a','k'])
t3 = np.sort(t2)
print(t3)             #  ['a','b','c','f','k']
```

14.1. Matplotlib

Matplotlib library

Graph plotting library
- Függvények , grafikonok kirajzolására, sokféle funkcióval → nagyítás, eltolás ...

Nem része az alap Python telepítésnek, külön fel kell telepíteni !

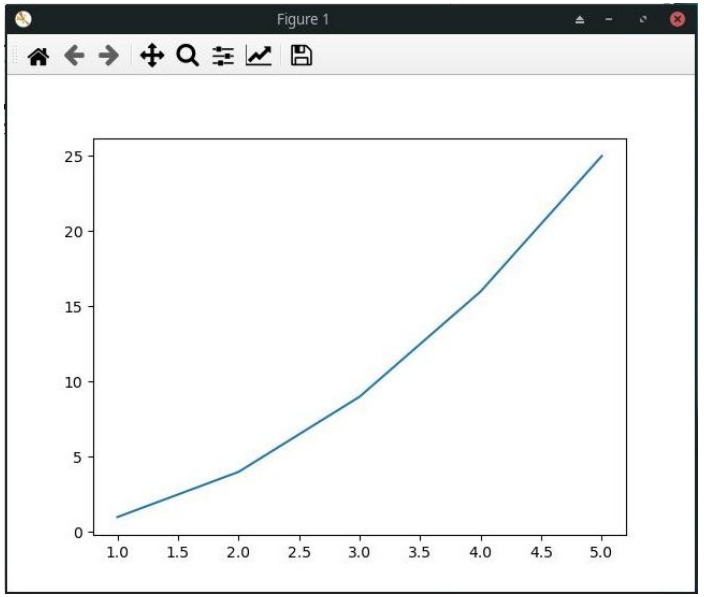
Telepítése a Python pip csomagkezelőjével: pip install matplotlib
A matplotlib legfontosabb része a pyplot almodul (matplotlib.pyplot)

Pyplot almodul függvényei

import matplotlib.pyplot as plt # általában 'plt' néven szokás importálni

- plot(xpoints, ypoints) → görbe kirajzolása
- plot(xpoints, ypoints,'o') → csak a pontok kirajzolása
- show() → megjelenítés

```
# 1. mintaprogram
import matplotlib.pyplot as plt

xpoints = (1, 2, 3, 4, 5)
ypoints = (1, 4, 9, 16, 25)

plt.plot(xpoints, ypoints)
plt.show()
```

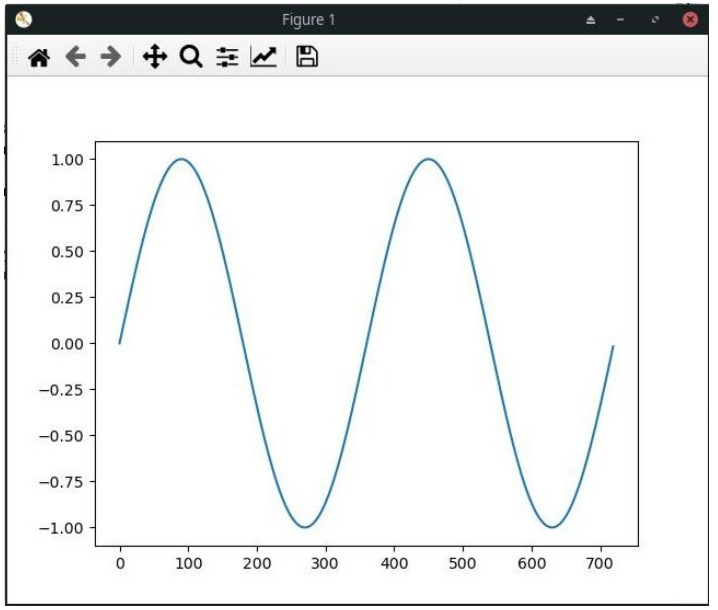


14.2. Pyplot

Pyplot almodul függvényei

- plot(ypoints) → az ’x’ értékeit nem muszáj megadni,de akkor azok a következők lesznek: 0, 1, 2, 3, ...

```
# 2. mintaprogram
import matplotlib.pyplot as plt
import math as mt

ypoints = [ ]

for i in range(720):          # szög 0 - 720 fokig
        x= i*mt.pi/180        # szög átszámítása radiánba !
        y= mt.sin(x)
        ypoints.append(y)

plt.plot(ypoints)
plt.show()
 # rad= fok*2*Pi/360
```



- plot(xpoints, ypoints,marker='o') → a pontok helyén jelek (markerek) sokféle lehet →

14.3. Pyplot, marker

Markerek

marker (a kirajzolt pontok !) formátuma

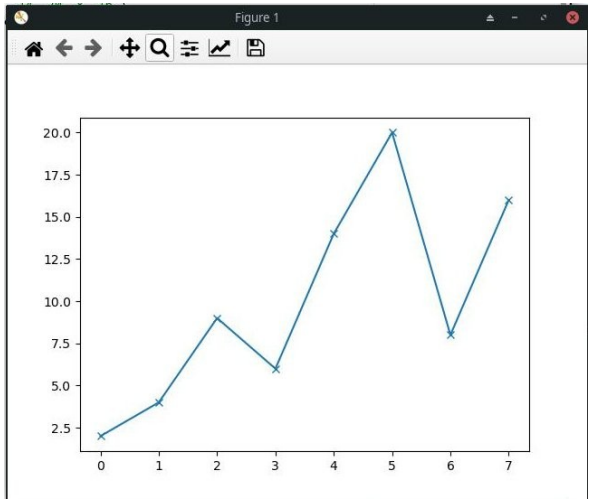
'o' Circle '*' Star '.' Point ',' Pixel 'x' X 'X' X (filled)

'+' Plus 'P' Plus (filled) 's' Square 'D' Diamond 'd' Diamond (thin)

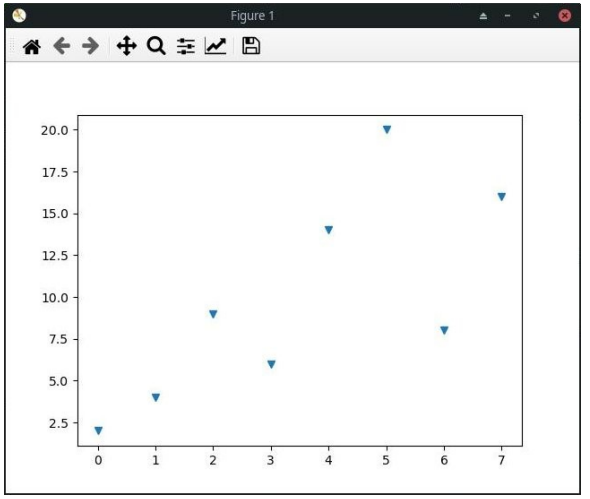
'p' Pentagon 'h' Hexagon 'v' Triangle Down '^' Up '>' right '<' left

'-' '|' '1' '2' '3' '4'

```
# 3. mintaprogram
import matplotlib.pyplot as plt

ypoints = (2, 4, 9, 6, 14, 20, 8, 16 )
plt.plot(ypoints,marker='x')
plt.show()
```




```
# 4. mintaprogram
import matplotlib.pyplot as plt

ypoints = (2, 4, 9, 6, 14, 20, 8, 16)
plt.plot(ypoints,'o', marker='v')
plt.show()
```



14.4. Pyplot, marker

Format strings

fmt → a marker + vonal + szín megadása egy sztringben (marker-line-color)
pl. ’o:r’ → circle+dotted+red

Line

'-' solid ':' dotted '--' dashed '-.' Dashed/dotted ' ' None

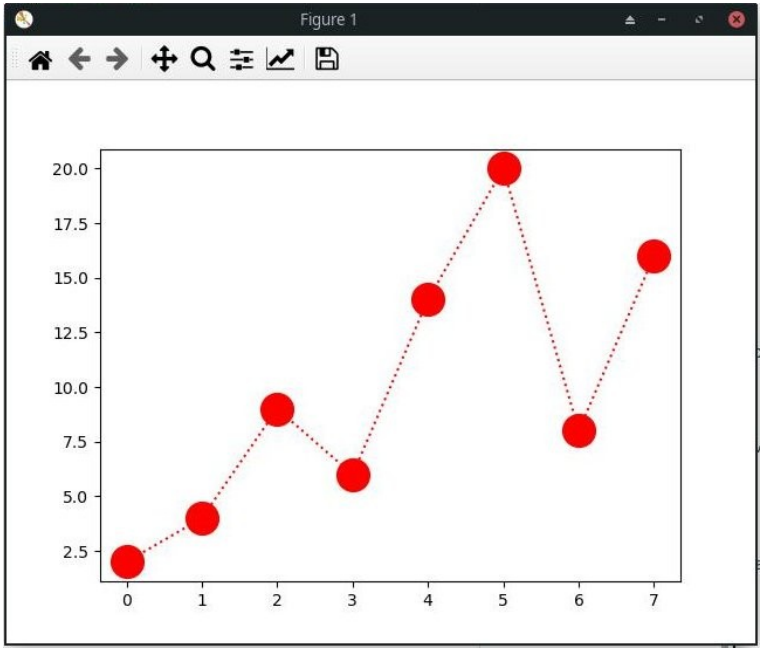
Color

'r' red 'g' green 'b' blue 'c' cyan 'm' magenta
'y' yellow 'k' black 'w' white

Marker mérete

markersize (vagy ms) kulcsszó → pl. ms=30

```
# 5. mintaprogram
import matplotlib.pyplot as plt

ypoints = (2, 4, 9, 6, 14, 20, 8, 16)
plt.plot(ypoints,'o:r', ms=20)
plt.show()
```



14.5. Pyplot, marker

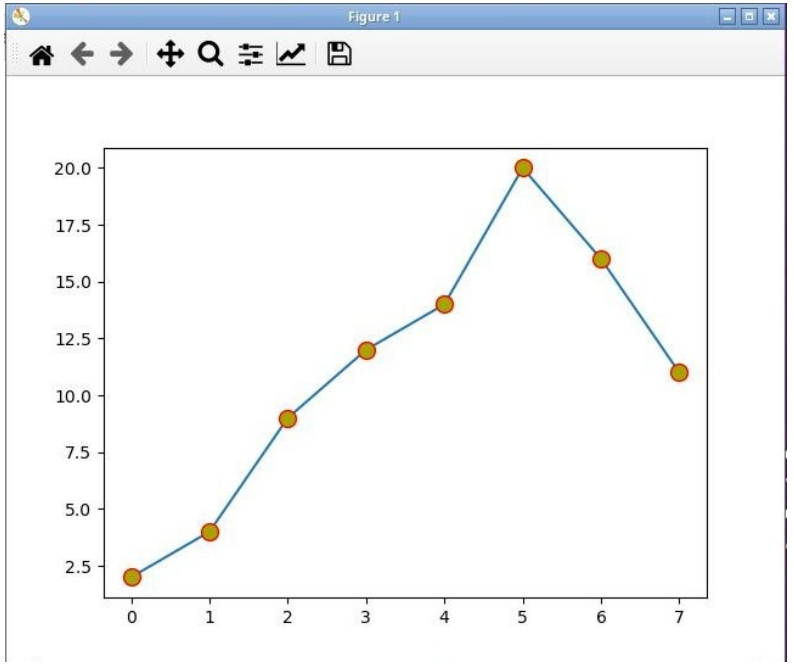
Marker szine

- markerfacecolor (vagy mfc) kulcsszó → marker belsejének színe, pl. mfc=’b’
- markeredgecolor (vagy mec) kulcsszó → marker körvonalának színe, pl. mec=’r’

Szin megadása kóddal:

Hexadecimális számmal is megadhatjuk a szín RGB összetevőit → #RRGGBB
RR (red), GG (green) and BB (blue) → mindegyik 00 – FF közötti érték (0-255)

```
# 6. mintaprogram
import matplotlib.pyplot as plt

ypoints = (2, 4, 9, 12, 14, 20, 16, 11)
plt.plot(ypoints, marker='o',  ms=10,  mec = 'r', mfc = '#a9a300')
plt.show()
```



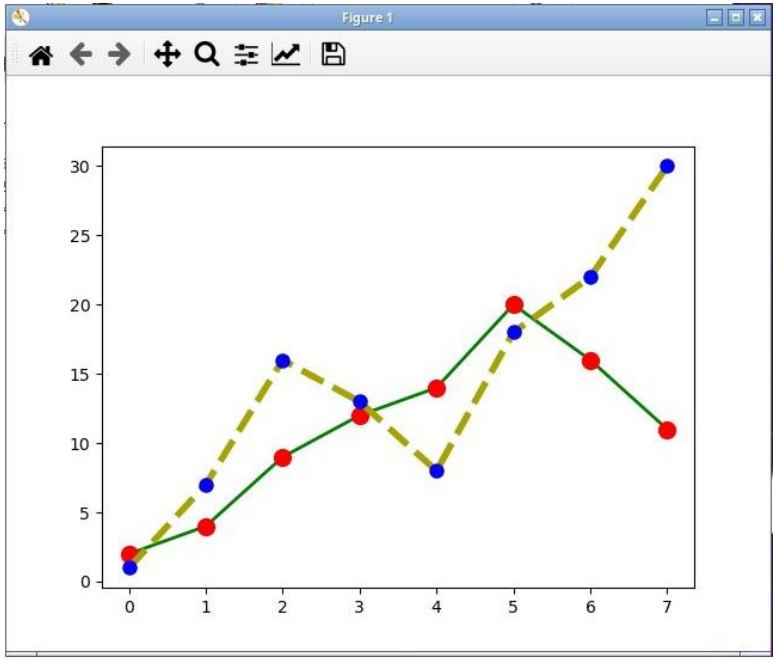
14.6. Pyplot, line

Line tulajdonságok

- color (vagy c) kulcsszó → összekötő vonal színe, pl. color=’b’ vagy c='#a9a300'
- linewidth (vagy lw) kulcsszó → összekötő vonal vastagsága, pl. lw=’3’
- linestyle (vagy ls) kulcsszó → összekötő vonal típusa, pl. ls=’dashed’

'-' solid ':' dotted '--' dashed '-.' dashdot ' ' None

```
# 7. mintaprogram
import matplotlib.pyplot as plt

yp1 = (2, 4, 9, 12, 14, 20, 16, 11)
yp2 = (1, 7, 16, 13, 8, 18, 22, 30)
plt.plot(yp1, marker='o', ms=10, mec = 'r', mfc = 'r', ls='-', lw = '2', c = 'g')
plt.plot(yp2, marker='o', ms=8, mec = 'b', mfc = 'b', ls='dashed', lw = '4', c = '#a9a300')
plt.show()
```



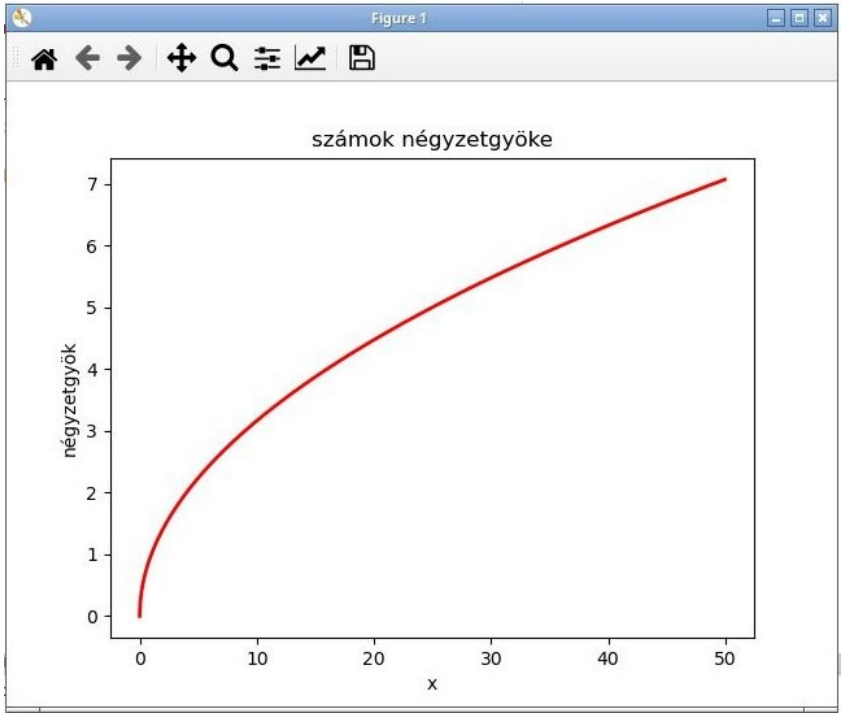
14.7. Pyplot, label

Cím és felíratok

A Pyplot almodul függvényeivel tudjuk a tengelyeket felíratozni, és címet adni az ábrának

- xlabel('xtengely_címe') → az ’x’ tengely címét adjuk meg
- ylabel('ytengely_címe') → az ’y’ tengely címét adjuk meg
- title('ábra_címe') → az ábra címét adjuk meg

```
# 8. mintaprogram
import matplotlib.pyplot as plt
import math as mt

xpoints = [ ]
ypoints = [ ]

for i in range(1000):
        x= i/20                    # x értéke 0-50
        xpoints.append(x)
        y= mt.sqrt(x)     # négyzetgyök számítása
        ypoints.append(y)

plt.plot(xpoints,ypoints, lw = '2', c = 'r')
plt.title('számok négyzetgyöke')
plt.xlabel('x')
plt.ylabel('négyzetgyök')
plt.show()
```



15.1. Pygame

Pygame

A Pygame library segítségével játék programokat, multimédiás programokat lehet készíteni.
Nem része az alap Python telepítésnek, külön fel kell telepíteni !

Telepítése a Python pip csomagkezelőjével: pip install pygame

Pygame moduljai

- pygame.display → ablak, képernyő vezérlése
- pygame.draw → rajzolás felületekre
- pygamey.event → események
- pygame.mouse → egér kezelés
- pygame.key → billentyűzet kezelés
- pygame.joystick → joystick kezelés
- pygame.image → képek kezelése
- pygame.time → idő vezérlés
- pygame.transform → skála, elforgatás, tükrözés

...

15.2. Pygame indítása

Pygame importálása, indítása, beállítások

- Importálása:

```
import pygame
```

- Indítás, inicializálás → init() metódus

```
pygame.init()
```

- Program ablak létrehozása → display modul, set_mode() metódusa

```
screen = pygame.display.set_mode((WIDTH, HEIGHT))
```

- Szín keverés, szín objektum → pygame, Color osztály

```
szin1 = pygame.Color(R, G, B)
```

- Program ablak háttérszín → fill() metódus

```
screen.fill((R, G, B))        vagy   screen.fill(szin1)
```

- Képernyő frissítése → display modul, update() metódusa

```
pygame.display.update()
```

- A puffer megjelenítése → display modul, flip() metódusa

```
pygame.display.flip()
```

15.3. Pygame ciklus, időzítések

Pygame fő ciklus

Egy pygame program központi része egy végtelen ciklus,
itt várunk a különböző eseményekre (billentyű lenyomás, egérklikk, …),
és kezeljük azokat

```
while True:
    ...
    ...
    ...
```

Időzítések

- Clock objektum létrehozása → time modul, Clock() metódusa

```
clock1 = pygame.time.Clock()
```

- Képkocka frissítés sebessége → time modul, Clock objektum, tick() metódusa

```
clock1.tick(20)    → 20 kép/másodperc
```

- késleltetés → time modul, delay() metódusa

```
pygame.time.delay(time_msec)
```

15.4. Pygame események

Events

- Billentyű lenyomás, egérkattintás, ablakbezárás,… eseményeket detektálni kell,
  és kezelni azokat (már, amelyiket szeretnénk)

- Pygame modulban előre definiált konstansok !

- események objektumok lekérdezése → pygame.event.get() metódussal
- események típusa → type attributum
- lenyomott billentyű → key attributum
- egér koordináta → pos attributum → (x,y)

Billentyű események

pygame.KEYDOWN	→ billentyű lenyomás történt
pygame.K_ESCAPE	→ Escape billentyű lenyomása volt
pygame.K_UP	→ FEL nyíl billentyű lenyomása volt
pygame.K_DOWN	→ LE nyíl billentyű lenyomása volt
pygame.K_LEFT	→ BALRA nyíl billentyű lenyomása volt
pygame.K_RIGHT	→ JOBBRA nyíl billentyű lenyomása volt
…	

Egér események

pygame.MOUSEBUTTONDOWN	→ egérgomb lenyomás történt
pygame.MOUSEBUTTONUP	→ egérgomb felengedése történt
pygame.MOUSEMOTION	→ egér mozgás történt
…	

Egyéb események

pygame.QUIT	→ kilépés programból (ablak bezárás)

15.5. Pygame program

Pygame program felépítése, minta 1.

```
import pygame          # Import the pygame modul

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600


pygame.init()          # Initialize pygame

screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))


running = True

while running:                              # Main loop
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:                # Billentyű lenyomás történt ?
            if event.key == pygame.K_ESCAPE:    # Esc lenyomása volt ? →  exit
                running = False
        elif event.type == pygame.QUIT:             #  QUIT esemény volt ? →  exit
            running = False

    screen.fill((0, 50, 50))            # Fill the screen

    pygame.display.update()             # Update the display
```

15.6. Pygame program

Pygame program felépítése, minta 2.
osztály használatával

```
import pygame          # Import the pygame modul

class Jatek(object):                    # osztály létrehozása
        def main(self, kepernyo):
                clock = pygame.time.Clock()

                while True:
                        for event in pygame.event.get():
                                if event.type == pygame.QUIT:
                                        pygame.quit()
                                        quit()
                                if event.type == pygame.KEYDOWN:
                                        if event.key == pygame.K_UP:
                                                print("Fel")
                                        if event.key == pygame.K_DOWN:
                                                print("Le")

                        pygame.display.update()
                        clock.tick(1)


if __name__ == '__main__':      # itt indul a program
        pygame.init()
        kepernyo = pygame.display.set_mode((400, 250))
        Jatek().main(kepernyo)
```

15.7. Rect

Pygame modul, Rect class

A Rect objektumok → téglalap (rectangle) koordinátáinak tárolására, manipulására.
Sok objektumnak van saját 'rect' attribútuma.

- Rect objektum létrehozása → Rect()

```
rect_obj = pygame.Rect(left, top, width, height)
rect_obj = pygame.Rect((left, top), (width, height))
rect_obj = pygame.Rect(object)
```

pl. `rect1 = pygame.Rect(20, 20, 100, 100)`

- Rect objektum másolása → copy()

```
rect2 = rect1.copy()
```

vagy `rect2 = pygame.Rect.copy(rect1)`

- Rect objektum mozgatása

→ move() új rectangle jön létre !! az eredeti nem változik !!

```
rect3 = rect1.move(x_offset,y_offset)
```

→ move_ip() az eredeti változik !!

```
rect1.move_ip(x_offset,y_offset)
```

15.8. Surface

Pygame modul, Surface class

A Surface osztály hoz létre rajzolási felületet.
Az eddig létrehozott program ablak is egy ilyen surface, csak speciális,
az a fő rajzolási felület

- Rajzolási felület (surface) létrehozása →

```
surface_obj = pygame.Surface((WIDTH, HEIGHT))
```

- Surface objektum háttérszín → fill() metódus

```
surface_obj.fill((R, G, B))        vagy   surfaceobj.fill(szin1)
```

- kép rajzolási felületre (surface) helyezése → surface, blit() metódusa

```
surface_obj.blit(image_obj,(x_kor, y_kor))
```

- Surface objektum téglalap területének lekérdezése → surface, get_rect() metódusa

```
rect_obj = surface_obj.get_rect()
```

15.9. Képek kezelése

Kép kirajzolása

- kép betöltése → image modul load() metódusa

```
kep1 = pygame.image.load('virag.jpg')       // virag.jpg mondjuk 64x64 pixeles
```

- kép kirajzolása

pl.

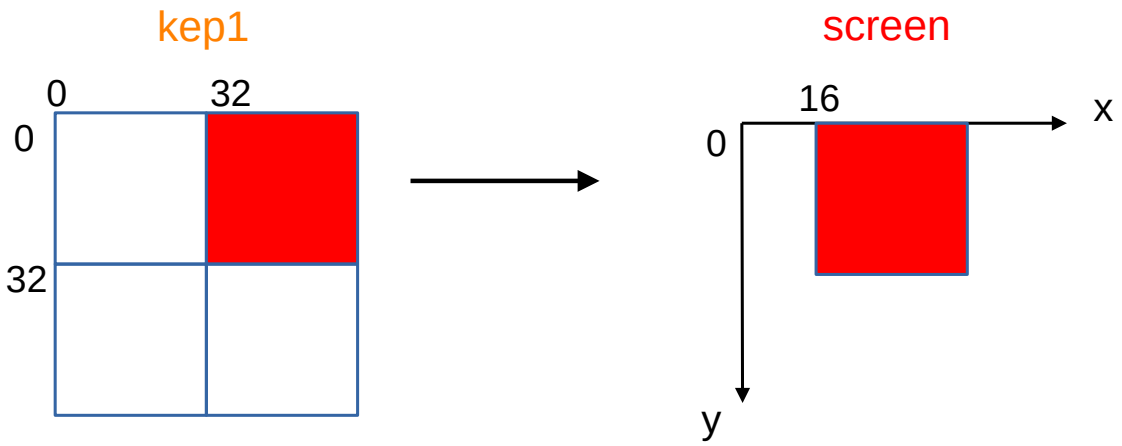
```
screen = pygame.display.set_mode((WIDTH, HEIGHT))       // surface
screen.blit(kep1,(10, 10))       // surface-re kép elhelyezése
```

- képrészlet kirajzolása, rajzolási felületre (surface) helyezése

```
surface_obj.blit(image_obj,(x_kor, y_kor),(x, y, width, height))
```

pl.

```
screen.blit(kep1,(16, 0),(32, 0, 32, 32))    // surface-re képrészlet elhelyezése
```



15.10. Sprite

Sprite modul, Sprite és Group class

Sprite → játék objektum, játék „lény” (kobold, manó)

- Sprite létrehozása → sprite modul, Sprite osztályból örökléssel

```
class Player(pygame.sprite.Sprite):
    def __init__(self, *groups):
        super(Player, self).__init__(*groups)
        self.surf = pygame.Surface((75, 25))
        self.surf.fill((255, 50, 50))
        self.rect = self.surf.get_rect()
```

- Sprite tároló (konténer) → sprite modul, Group osztály

  pl. sprites1 = pygame.sprite.Group()

- Sprite hozzáadása Group-hoz → add() metódus (Sprite és Group objektumnak is van !)

- Sprite eltávolítása Group-ból → remove() metódus (Sprite és Group objektumnak is van !)

- Sprite milyen Group-okban van benne → Sprite objektum, groups() metódusa

- Group-ban milyen Sprite-k vannak → Group objektum, sprites() metódusa

- Group-ból az összes Sprite törlése → Group objektum, empty() metódusa

15.11. Pygame program

Pygame program felépítése, minta 3.

```
import pygame           # Import the pygame modul

class Jatek(object):
    def main(self, kepernyo):
        clock = pygame.time.Clock()

        kep = pygame.image.load('kep2bb.jpg')
        kep_x = 50
        kep_y = 50

        while True:
            clock.tick(20)
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    return
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_LEFT:
                        print("Balra")
                    if event.key == pygame.K_UP:
                        print("Fel")
                if event.type == pygame.KEYDOWN and \
                        event.key == pygame.K_ESCAPE:
                    return
            kep_x += 10             # kép mozgatása
            kepernyo.fill((200, 200, 200))
            kepernyo.blit(kep, (kep_x, kep_y))
            pygame.display.flip()

if __name__ == '__main__':
    pygame.init()
    kepernyo = pygame.display.set_mode((640, 480))
    Jatek().main(kepernyo)
```

15.12. Pygame program

Pygame program felépítése, minta 4.

```
import pygame          # Import the pygame modul

SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600

class Player(pygame.sprite.Sprite):          # Sprite → játék objektum
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25))
        self.surf.fill((255, 50, 50))
        self.rect = self.surf.get_rect()

pygame.init()          # Initialize pygame

screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
player = Player()

running = True

while running:                              # Main loop
    for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:                # Check for KEYDOWN event
            if event.key == pygame.K_ESCAPE:    # If the Esc key is pressed →  exit
                running = False
        elif event.type == pygame.QUIT:             # If QUIT event →  exit
            running = False

    screen.fill((0, 50, 50))          # Fill the screen
    screen.blit(player.surf, (SCREEN_WIDTH/2, SCREEN_HEIGHT/2))      # sprite a képernyőre
    pygame.display.flip()          # Update the display
```