

# PYTHON FUNDAMENTALS

Curs interactiv de python

# STRUCTURA SEDINTA 8 : MODULE AVANSAT

- ▢ Tema sedinta anterioara
- ▢ Kivy widget-uri
- ▢ Kivy widget-uri avansate
- ▢ Limbajul Kivy
- ▢ Prezentare tema proiect final

# TEMA CLASA BOXLAYOUT

INFOACADEMY.NET

Exercitiu pe baza programului Boxlayout0.py:

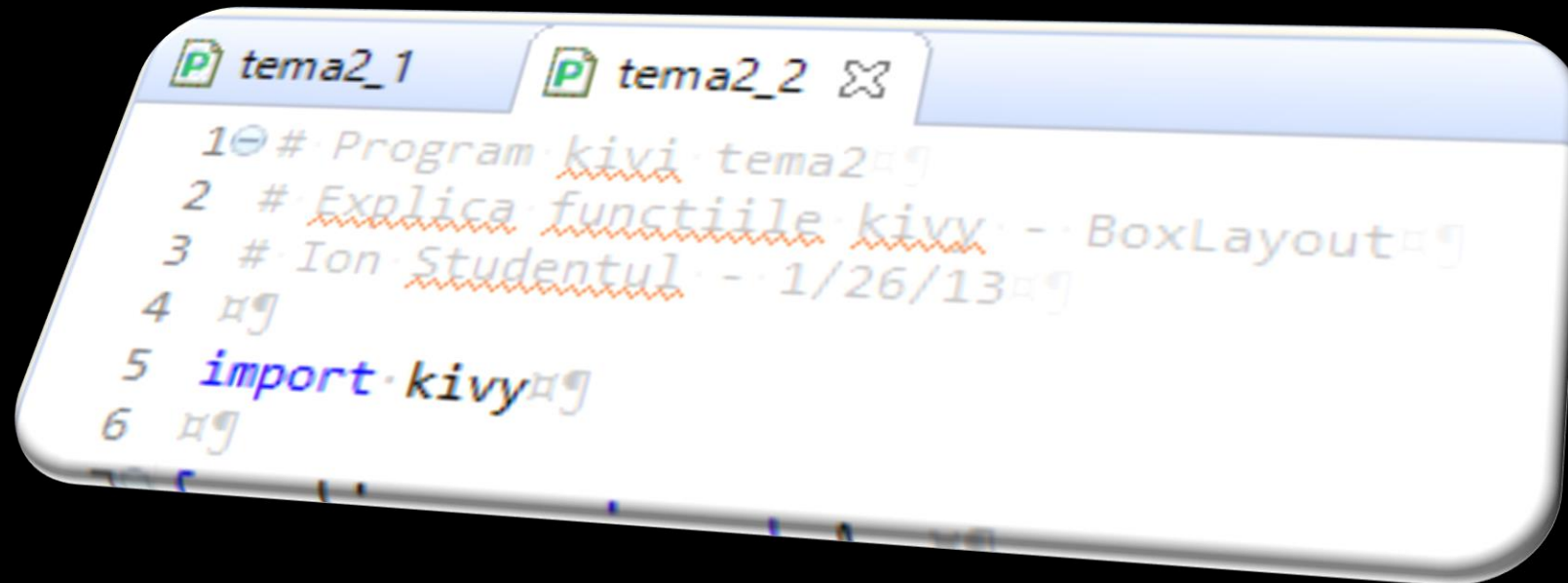
- Sa modificati padding la 20
- Sa modificati orientarea la "vertical"
- Sa adaugati spacing cu valoarea 20
- Sa adaugati size\_hint\_x cu valoarea 0.5 și size\_hint\_y cu valoarea 0.3, apoi să rulați

```
tema1 ✕  
1 # Program kivi tema1  
2 # Explica functiile kivy -- BoxLayout  
3 # Ion Studentul -- 1/26/13  
4  
5 import kivy  
6
```

Creati un program kivy cu un layout de tip boxlayout care sa afiseze 2 widget-uri de tip buton cu textele:

- <<Am plecat>>
- <<Am venit>>

Acestea sa aiba culoarea de background albastra. La apasare (evenimentul `object.on_press`) sa printeze textul butonului.



```
1 # Program kivi tema2
2 # Explica functiile kivy -- BoxLayout
3 # Ion Studentul -- 1/26/13
4
5 import kivy
6
```



Creati un program kivy cu un layout de tip boxlayout care sa afiseze 2 widget-uri de tip buton cu textele:

- <<Am plecat>>
- <<Am venit>>

Acestea sa aiba culoarea de background albastra. La apasarea butonului cu textul <<Am plecat>> (evenimentul `obiect.on_press`) sa schimbam culoarea scrisului butoanelor in rosu (`self.buton.color=(1,0,0,1)`)

La apasarea butonului cu textul <<Am venit>> (evenimentul `obiect.on_press`) sa schimbam culoarea scrisului butoanelor in verde(`self.buton.color=(0,1,0,1)`)

```
tema3 ✖  
1 # Program kivi tema3  
2 # Explica functiile kivy -- BoxLayout  
3 # Ion Studentul -- 1/26/13  
4  
5 import kivy  
6  
7 from kivy.app import App  
8 from kivy.uix.button import Button
```



# KIVY: PERSONALIZAREA APLICATIEI

- Putem modifica dimensiunile root-ului. Pt. acest task trebuie sa importam Config

```
from kivy.config import Config
```

- Pentru a seta efectiv aceasta valoare trebuie sa folosim Config.set cu ajutorul caruia setam dimensiunile aplicatiei inainte de a a rula aplicatia:

```
Config.set('graphics', 'width', '300')
```

```
Config.set('graphics', 'height', '300')
```

# KIVY: PERSONALIZAREA APLICATIEI

- In cazul in care dorim sa adaugam propriul icon la aceasta aplicatie kivy ne ofera aceasta flexibilitate. In corpul metodei build putem seta și iconita aplicatiei. Aceata iconita reprezinta un piton (sarpe) și poate fi vazuta în coltul din stanga sus. Mai jos se regaseste extrasa linia care seteaza iconita.

```
self.icon = "python1.ico"
```

# KIVY: PERSONALIZAREA APLICATIEI

INFOACADEMY.NET

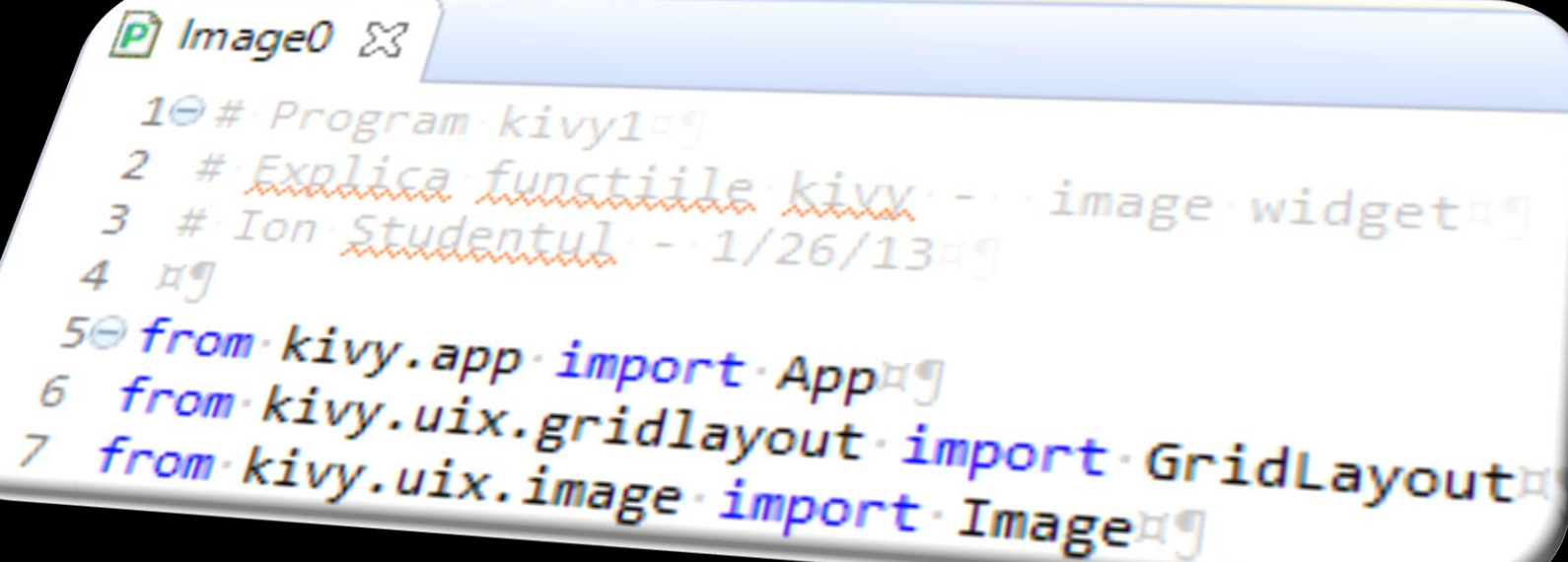
```
personalizarea_aplicatiei3 ✕  
1 # Program ico si dimensiune  
2 # Explica functiile kivy  
3 # Ion Studentul - 1/26/13  
4  
5 from kivy.app import App  
6 from kivy.uix.gridlayout import GridLa  
7 from kivy.uix.label import Label  
8 from kivy.uix.textinput import TextInp  
9 from kivy.config import Config
```

# KIVY: WIDGET-UL IMAGE

- Urmatorul widget este Image. Utilizarea Image implica importarea Image ce se regaseste sub modulul `kivy.uix.image`. Utilizarea se face prin crearea unui obiect de tip `image` care primeste parametrul `source` ce indica o imagine.
- Kivy poate incarca diverse tipuri de imagini cum ar fi PNG, GIF sau JPG.

# KIVY: WIDGET-UL IMAGE

INFOACADEMY.NET



```
1 # Program kivy1
2 # Explica funtiile kivy - - - image widget
3 # Ion Studentul - - 1/26/13
4
5 from kivy.app import App
6 from kivy.uix.gridlayout import GridLayout
7 from kivy.uix.image import Image
```

- Cream obiecte Image si le adaugam ca si child la un layout creat

```
self.imag1 = Image(source="infoacademy1.gif")
self.add_widget(self.imag1)
self.imag2 = Image(source="snake1.jpg")
self.add_widget(self.imag2)
self.imag3 = Image(source="snake2.png")
self.add_widget(self.imag3)
```



# KIVY: WIDGET-UL IMAGE

Vom reutiliza codul programului de mai sus pentru a se vedea cum reusim să manipulam imaginile statice sau dinamice(GIF).

Cu ajutorul proprietatii opacity setam gradul de opacitate al pozei. Standard aceasta este 1 adica opaca. Dacă scadem la 0 atunci este total transparenta.

```
self.imag1.opacity = 0.4 #default este 1
```

Cu ajutorul proprietatii anim\_delay vom seta cat de rapida să fie animatia noastra. Default este 0.25 adica 4 FPS. Dacă o micșorăm animatia va fi mai rapida, dacă o creștem animatia va fi mai lenta.

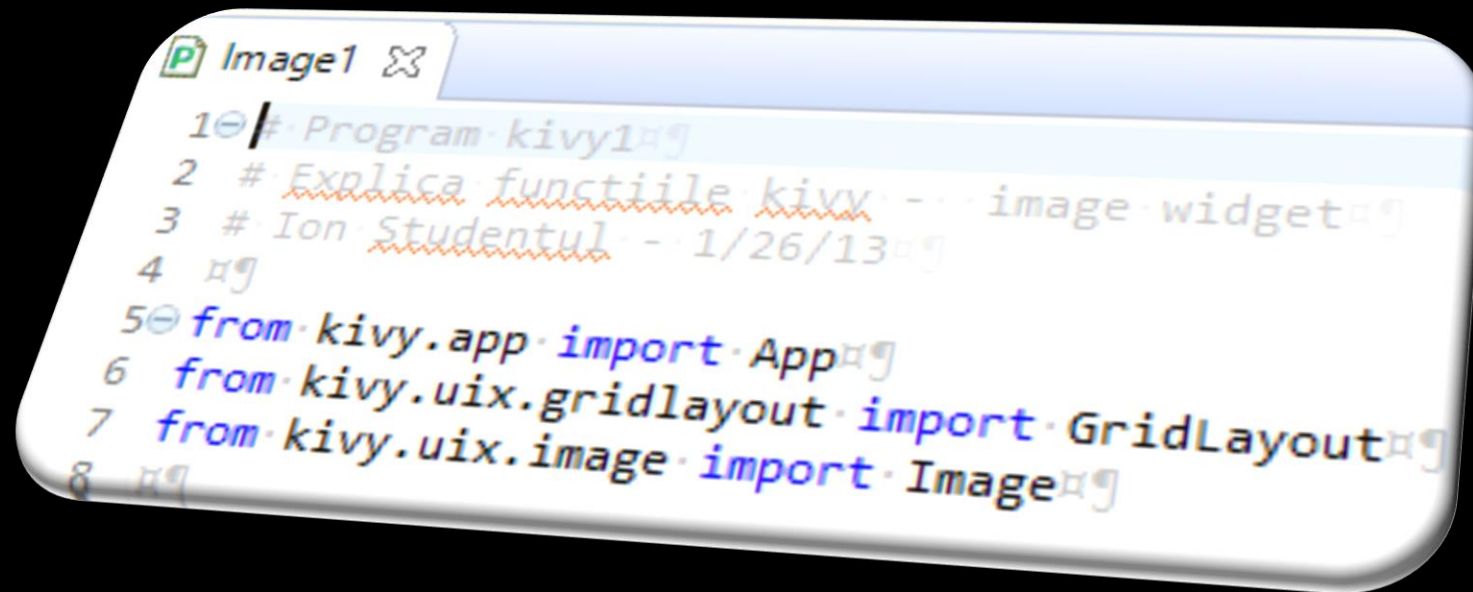
```
self.imag2.anim_delay = 0.04 # default este 0.25 (4fps)
```

Cu ajutorul proprietatii color putem seta o culoare peste o imagine, astfel ii schimbam aspectul. Standard aceasta nu are o culoare definita.

```
self.imag3.color = (0,1,0,1)
```

# KIVY: WIDGET-UL IMAGE

INFOACADEMY.NET



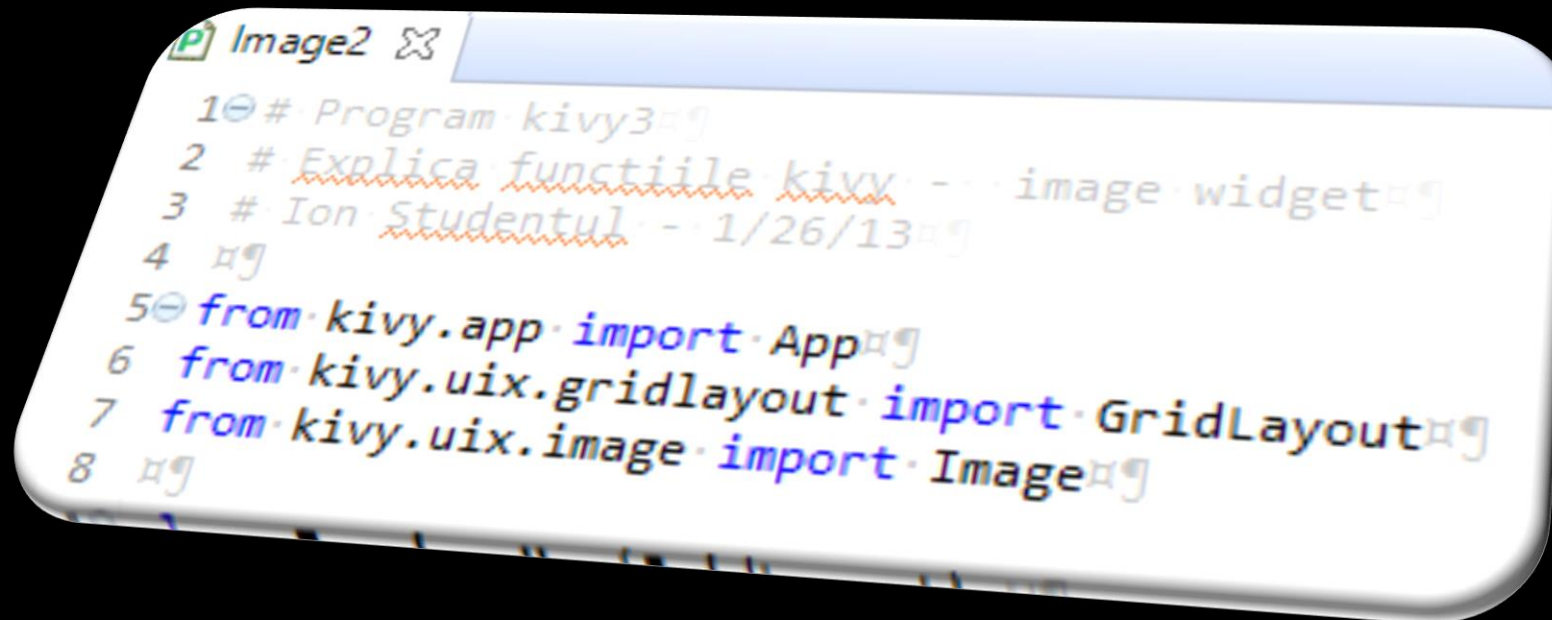
```
1 # Program kivy1
2 # Explica functiile kivy -- image widget
3 # Ion Studentul -- 1/26/13
4
5 from kivy.app import App
6 from kivy.uix.gridlayout import GridLayout
7 from kivy.uix.image import Image
8
```

# KIVY: WIDGET-UL IMAGE

Putem să reducem dimensiunea unei poze adaugate cu atributul `size_hint`.

```
self.imag2.size_hint_x = 0.1 #(scala de la 0 la 1)
```

```
self.imag2.size_hint_y = 0.1 #(scala de la 0 la 1)
```



# KIVY: WIDGET-UL IMAGE

Cum să pun o imagine de fundal aplicatiei mele?

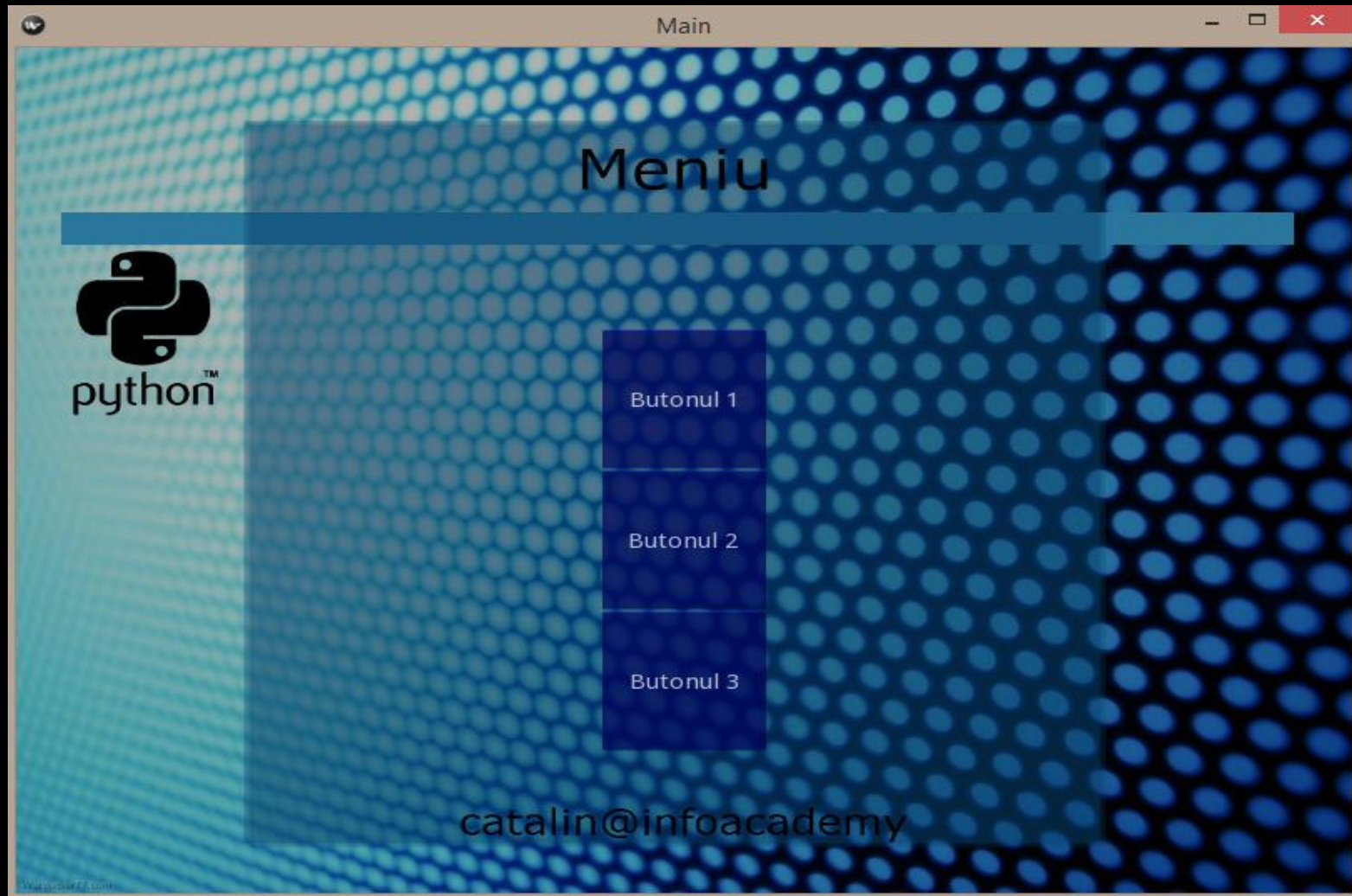
Putem sa cream un obiect de tip imagine pe care sa-l facem child la layout, iar toate celelalte elemente copii la imagine. Astfel am creat un fundal.

```
background_layout_image_without_resize ✕  
1 # Program kivy4  
2 # Explica functiile kivy -- image background  
3 # Ion Studentul -- 1/26/13  
4  
5 from kivy.app import App  
6 from kivy.uix.floatlayout import FloatLayout  
7 from kivy.uix.image import Image  
8 from kivy.uix.button import Button
```



# KIVY: WIDGET-UL IMAGE

INFOACADEMY.NET



# KIVY: WIDGET-UL IMAGE

În metoda `init` regăsim crearea unei imagini ce are ca sursă fișierul `fundal.jpg` ce se regăsește în aceeași locație cu programul (locație relativă). Următorul pas este să modificăm o proprietate a imaginii și anume `opacity` (gradul de opacitate). Astfel imaginea devine complet opacă la 1 și complet transparentă la 0. Aici se regăsește valoarea 0.7. Apoi adăugăm această imagine la layout-ul părinte.

```
self.imag1 = Image(source="fundal.jpg")
```

```
self.imag1.opacity = 0.7
```

```
self.add_widget(self.imag1)
```



Urmeaza crearea a trei butoane la care setam un text inteligibil, setam proprietatea bold la True și aceeași valoare a culorii - albastru(0,0,1,1). Toate butoanele sunt adaugate ca copii la obiectul Image creat anterior. De asemenea setam și un size\_hint la butonul 1 și butonul 3 cu scopul de a modifica dimensiunea acestuia.

```
self.but1 = Button(text = "Butonul 1", bold = True, background_color = (0,0,1,1))
self.but1.pos = (350,300)
self.but1.size_hint = (1,0.05)
self.imag1.add_widget(self.but1)
self.but2 = Button(text = "Butonul 2", bold = True, background_color = (0,0,1,1))
self.but2.pos = (350,200)
self.imag1.add_widget(self.but2)
```

```
self.but3 = Button(text = "Butonul 3", bold = True, background_color = (0,0,1,1))  
self.but3.pos = (350,100)  
self.but3.size_hint = (0.3,0.05)  
self.imag1.add_widget(self.but3)
```

Din pacate aceasta metoda ce nu include metoda de redesenare creaza o solutie ce nu permite setarea dimensiunii unui buton, deci size\_hint nu este luat în calcul. Daca ar fi luat în calcul but1 ar trebui să aiba dimensiunea orizontala cat imaginea(1,0.05). De asemenea, vedem ca copii la imag1 mostenesc caracteristici precum opacity.

# KIVY: WIDGET-UL IMAGE

INFOACADEMY.NET

Solutia presupune ca in metoda `init` a clasei `CostumLayout` regasim crearea unei imagini și redimensionarea layout-ului.

with `self.canvas.before`:

```
self.imag1 = Image(source="fundal.jpg")
```

```
self.imag1.opacity = 0.7
```

```
self.add_widget(self.imag1)
```

```
self.rect = Rectangle(size=self.size, pos=self.pos)
```

```
self.bind(size=self._update_rect, pos=self._update_rect)
```

```
def _update_rect(self, instance, value):
```

```
    self.rect.pos = instance.pos
```

```
    self.rect.size = instance.size
```

În MainApp sub metoda build începem prin a apela CustomLayout, deci ne va rezulta un layout ce are o imagine ce poate fi redimensionată.

```
c = CustomLayout()
```

La acest widget cream cele trei butoale, modificând caracteristici precum dimensiunea, gradul de opacitate sau culoarea, pe care ulterior le adăugăm la acest layout creat anterior.

```
self.but1 = Button(text = "Butonul 1", bold = True, background_color = (0,0,1,1))
```

```
self.but1.pos = (290,380)
```

```
self.but1.size_hint = (0.25,0.1)
```

```
c.add_widget(self.but1)
```

```
self.but2 = Button(text = "Butonul 2", bold = True, background_color = (0,0,1,1))
```

```
self.but2.pos = (290,280)
self.but2.size_hint = (0.25,0.1)
c.add_widget(self.but2)
self.but3 = Button(text = "Butonul 2",bold =True, background_color = (0,0,1,1))
self.but3.pos = (290,180)
self.but3.size_hint = (0.25,0.1)
self.but3.opacity = 0.7
c.add_widget(self.but3)
```

Datorita faptului ca cream aceste butoane sub metoda build trebuie să returnam layout-ul pt. a se aplica schimbarile facute.

```
return c
```

```
background_layout_image ✕  
14    
15  def __init__(self, **kwargs):  
16      super(CustomLayout, self).__init__(**kwargs)  
17    
18      with self.canvas.before:  
19          self.imag1 = Image(source="fundal.jpg")  
20          self.imag1.opacity = 0.7  
21          self.add_widget(self.imag1)  
22          self.rect = Rectangle(size=self.size, pos=
```



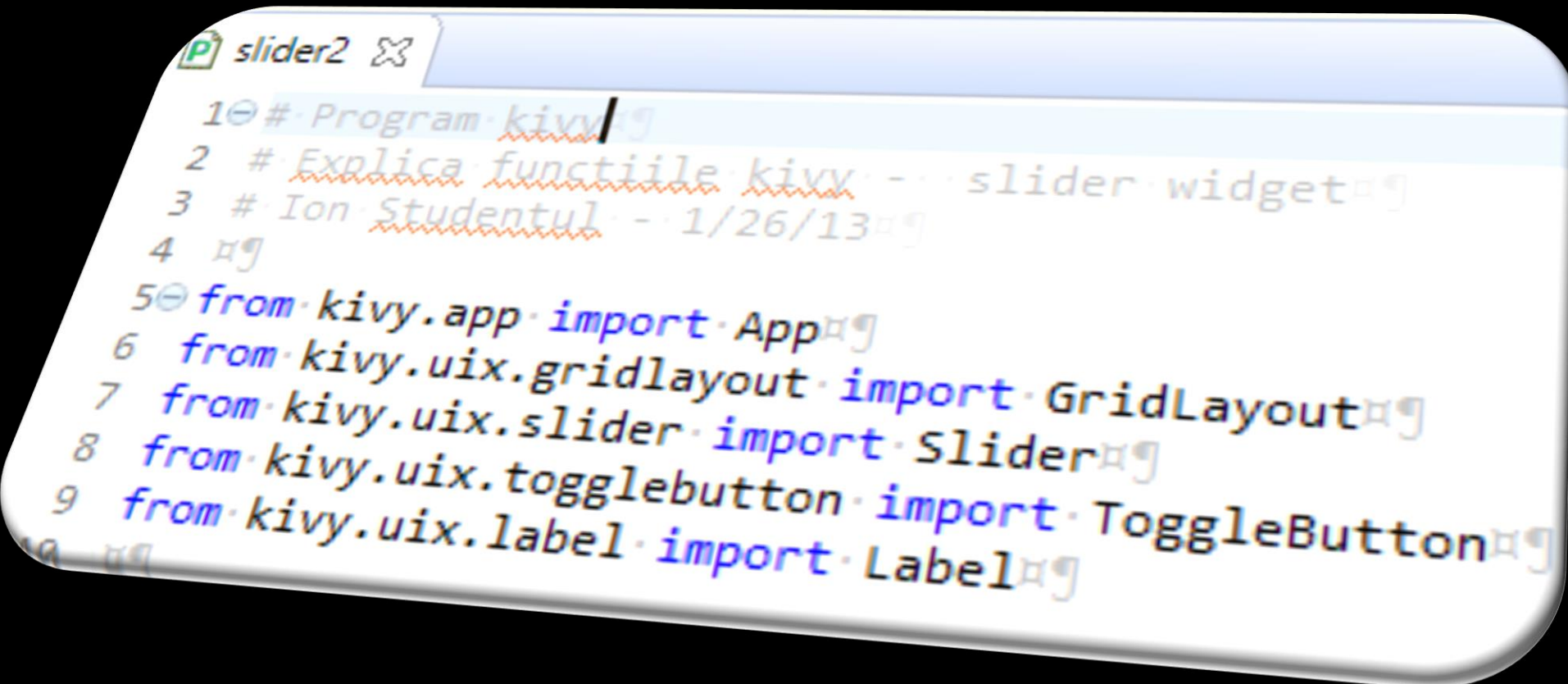
- Urmatorul widget pe care dorim sa-l studiem este slider. Acesta include posibilitatea să reglam ceva în trepte. Spre exemplu volumul poate fi oprit sau pornit, dar poate avea și trepte de volum. Putem utiliza acest widget creand un obiect de tip Slider ce are obligatoriu campurile min,max si value(valoare initiala) .
- `self.slide_muzica = Slider(min=0, max=100, value=10)`
- Pentru a seta un range trebuie să initializam valoarea de minimum (min=0),valoarea de maximum (max=100) și o valoare ce va fi afisata la inceputul aplicatiei (value=10).
- Cu ajutorul proprietatii step vom seta pasul dintre minim și maxim. Aici este 10.
- Putem utiliza și padding ca slider-ul să nu ajunga la marginea ferestrei, setand astfel o zona tampon intre margine layout-ului și slider.
- Orientarea default este orizontala, dar putem schimba slider-ul să aiba o orientare verticala („vertical”). Pentru a lega un eveniment de slider trebuie să utilizam în bind evenimentul <<value>> deoarece aceasta isi schimba valoarea la miscarea slider-ului.

```
slider0 23
1 Program kivy4.py
2 Explica functiile kivy -- slider widget.py
3 Ion Studentul -- 1/26/13.py
4
5 om kivy.app import App.py
6 om kivy.uix.gridlayout import GridLayout.py
7 om kivy.uix.slider import Slider.py
8 om kivy.uix.togglebutton import ToggleButton.py
9 om kivy.uix.label import Label.py
10 zica_activa = 0.py
```

În următorul exemplu vom lega un slider de un toggle button. Astfel va fi activ doar dacă starea toggle button este ridicată.

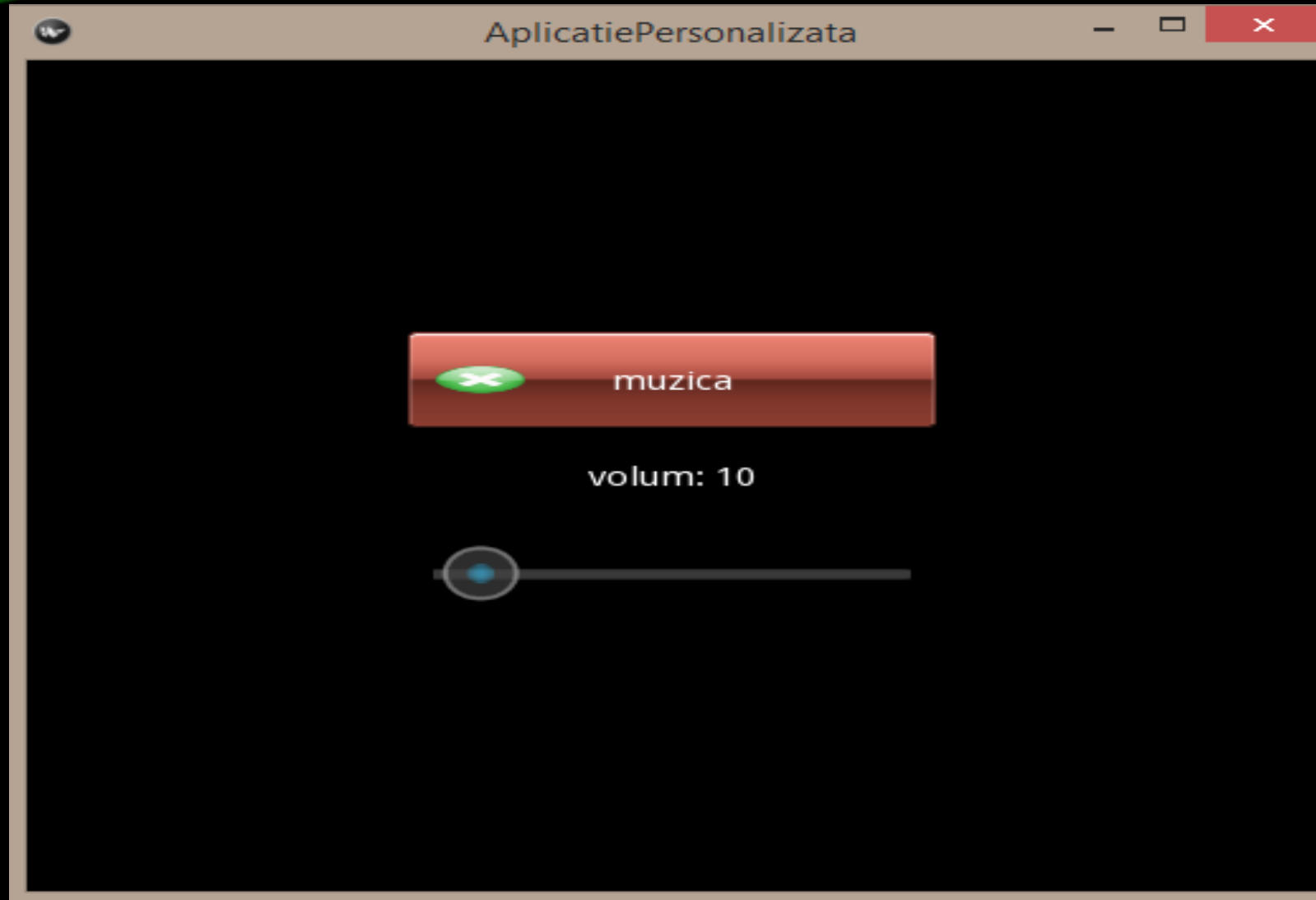
```
slider1 ✖  
1 # Program kivy4  
2 # Explica functiile kivy --- slider widget  
3 # Ion Studentul -- 1/26/13  
4  
5 from kivy.app import App  
6 from kivy.uix.gridlayout import GridLayout  
7 from kivy.uix.slider import Slider  
8 from kivy.uix.togglebutton import ToggleButton  
9 from kivy.uix.label import Label  
10
```

Slider2.py realizeaza acelasi lucru dar utilizand atributul state al ToggleButton





```
1 # Program kivy
2 # Explica functiile kivy - slider widget
3 # Ion Studentul - 1/26/13
4
5 from kivy.app import App
6 from kivy.uix.gridlayout import GridLayout
7 from kivy.uix.slider import Slider
8 from kivy.uix.togglebutton import ToggleButton
9 from kivy.uix.label import Label
```





Urmatorul widget este Switch. Switch este un buton cu doua stari: on si off. Nu este cu mult diferit de toggle button ca si creare sau utilizare. Pentru a utiliza acest widget creand un obiect de tip Switch. Vom demonstra functionalitatea switch prin modificarea programului anterior.

Asa cum se poate vedea în programul de mai jos, am inlocuit toggle cu switch. De asemenea am setat ca starea lui de inceput să fie activ. (active= True). Standard acest buton este off (active = False).

```
switch    
1 # Program kivy5  
2 # Explica functiile kivy --- slider widget  
3 # Ion Studentul -- 1/26/13  
4  
5 from kivy.app import App  
6 from kivy.uix.gridlayout import GridLayout  
7 from kivy.uix.slider import Slider  
8 from kivy.uix.switch import Switch  
9 from kivy.uix.label import Label  
10 muzica_activa = 0
```



# KIVY: WIDGET-UL SOUNDLOADER INFOACADEMY.NET

Cu ajutorul importului de SoundLoader si widget creand un obiect de tip SoundLoader vom putea rula o melodie.

```
from kivy.core.audio import SoundLoader
```

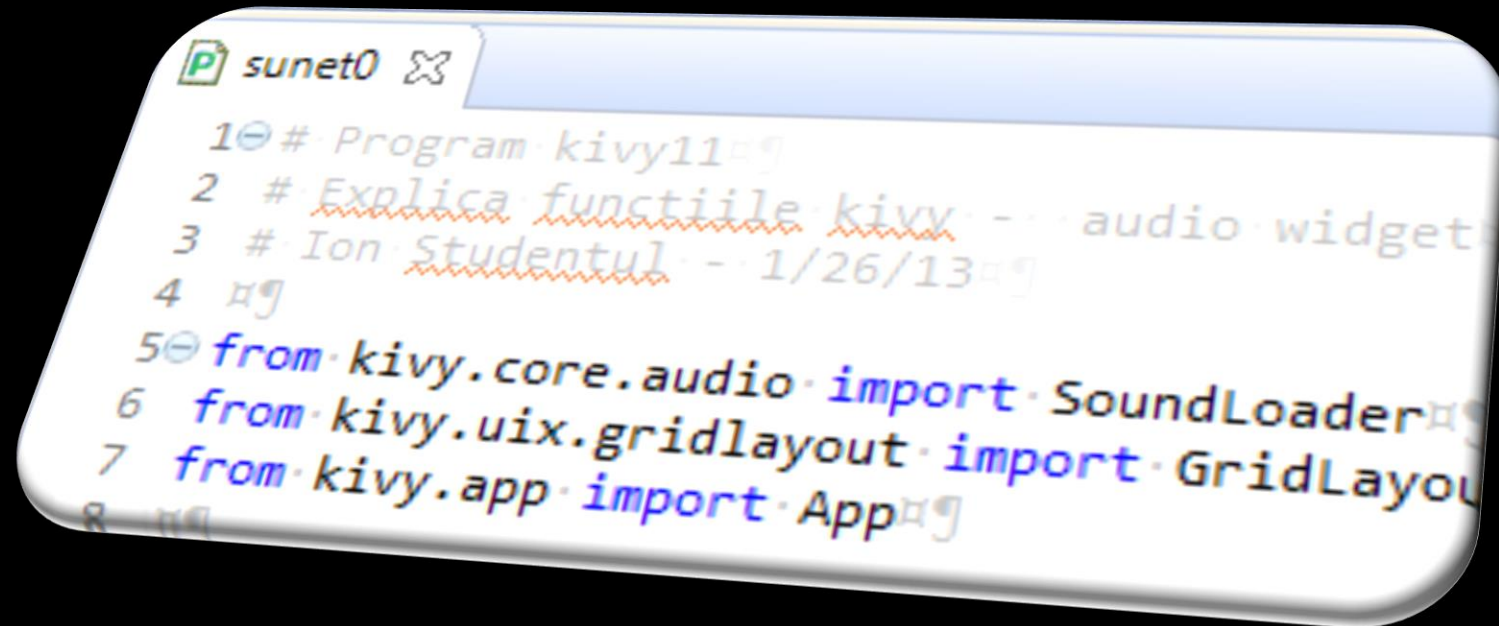
Trebuie să cream un layout deoarece dacă nu avem nimic de afisat kivy nu ruleaza. În fond, doar prin simpla utilizare Kivy se doreste crearea unei aplicatii grafice. Nu am atasat la acest program o figura deoarece nu este nimic de afisat cu exceptia unei ferestre negre.

Cu ajutorul SoundLoader.load incarcam un fisier de tip wav,ogg sau mp3. Incarcam un fisier cu load(fisier) și eliminam un fisier din memorie cu unload(). Astfel putem crea un player audio.

# KIVY: WIDGET-UL SOUNDLOADER INFOACADEMY.NET

- Incarcarea și rularea fișierelor de sunet sunt date de GStreamer pentru windows sau linux.
- În mod normal ar trebui să poată utiliza ogg și mp3 pentru noua versiune kivy 1.9. Mai multe detalii despre versiunea de GStreamer găsiți [aici](#). Deci vom rămâne în aceste exemple la rularea de ogg-uri. Dacă rulăm `obiect.play()` acel fișier va fi rulat. `Obiect.source` arată fișierul rulat și `obiect.length` arată durata fișierului.

# KIVY: WIDGET-UL SOUNDLOADER INFOACADEMY.NET



```
sunet0 ✖  
1 # Program kivy11  
2 # Explica functiile kivy --- audio widget  
3 # Ion Studentul -- 1/26/13  
4  
5 from kivy.core.audio import SoundLoader  
6 from kivy.uix.gridlayout import GridLayout  
7 from kivy.app import App  
8
```

# KIVY: WIDGET-UL SOUNDLOADER INFOACADEMY.NET

```
sunet1.py - D:\Catalin\Predare Python\carte\Se  
Run Options Windows Help  
m kivy11  
a functiile kivy - audio widget  
udentul - 1/26/13  
y.core.audio import SoundLoader
```

# KIVY: WIDGET-UL SOUNDLOADER INFOACADEMY.NET

- In programul de mai jos vom incerca să unim programul în care am aratat functionalitatea widget-ului switch cu programul în care am aratat functionalitatea widget-ului SoundLoader.

```
sunet2 ✕  
9 from kivy.uix.switch import Switch  
10 from kivy.uix.label import Label  
11  
12 class Switch_implementare(GridLayout):  
13
```



# KIVY: WIDGET-UL SOUNDLOADER INFOACADEMY.NET

Cu ajutorul proprietatii `loop` vom seta ca aceasta melodie să se repete la infinit, dacă valoarea este setata la `True`. Cu ajutorul proprietatii `volume` putem seta volumul melodiei.

```
self.sound.loop = True
```

```
self.sound.volume=0.5
```

De fiecare data cand mutam cursorul slide-ului atunci vom modifica și proprietatea `volume`

```
def valoare_volum (self,x,y):
```

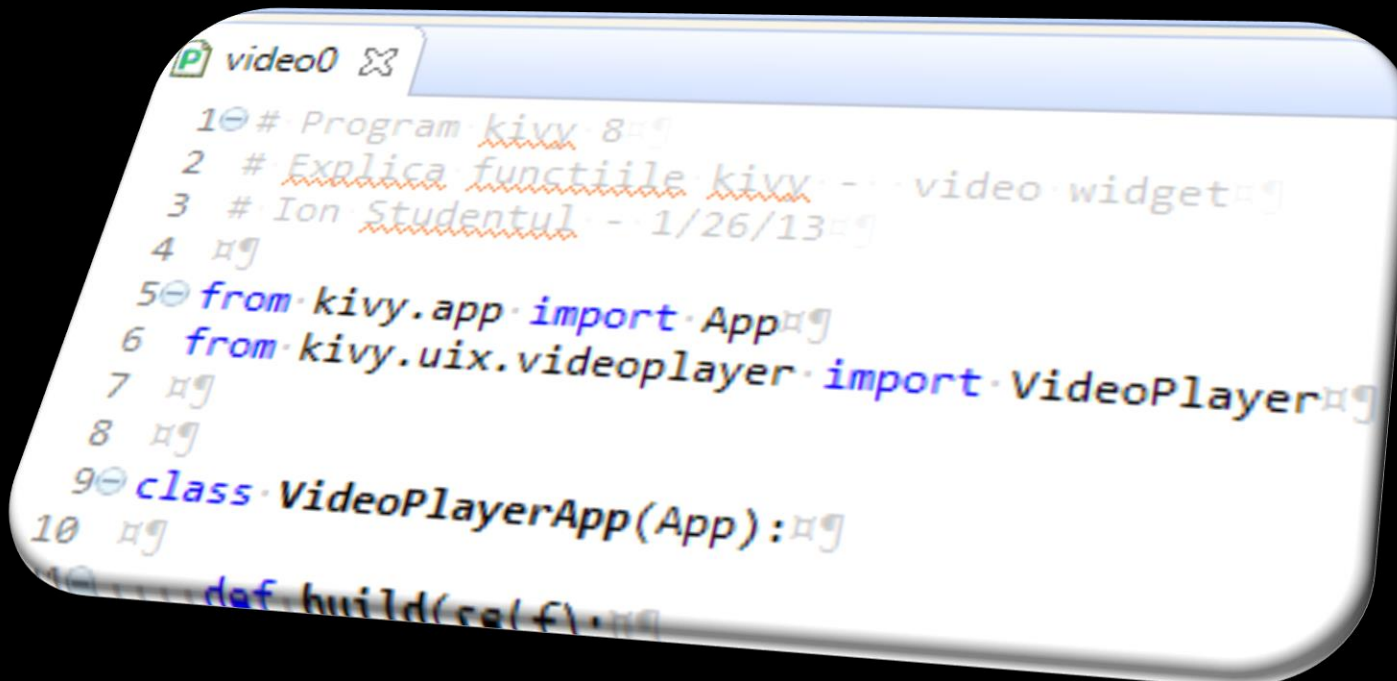
```
    '''utiliziat pentru a seta volumul'''
```

```
    self.arata_volum.text = "volum: "+str(int(self.slide_muzica.value))
```

```
    self.sound.volume = self.slide_muzica.value/100
```

Cand dorim să oprim sonorul atunci apelam și `self.sound.stop()`. Daca dorim sa-l pornim atunci apelam `self.sound.play()`. Cand sonorul este repornit melodia este rulata de la inceput.

Urmatorul widget studiat este video. Pentru a utiliza acest widget cream un obiect de tip VideoPlayer. Mai jos regasim un exemplu simplu cu un astfel de widget. Trebuie să reținem ca acest widget este dependent de codecurile instalate pe acel sistem de operare, dar și de GStreamer( similar widget-ului sound ). Sursa filmului utilizat este site-ul youtube.ro și se poate regasi [aici](#).



```
1 # Program kivy 8
2 # Explica functiile kivy -- video widget
3 # Ion Studentul -- 1/26/13
4
5 from kivy.app import App
6 from kivy.uix.videoplayer import VideoPlayer
7
8
9 class VideoPlayerApp(App):
10
11     def build(self):
```

Vedem ca Kivy vine cu un player ce are incluse butoane de play, pauza, de alegerea sectiunii de rulare sau de volum. Mai jos regasim un program care utilizeaza widget-ul video și cele mai uzuale proprietati ale lui.

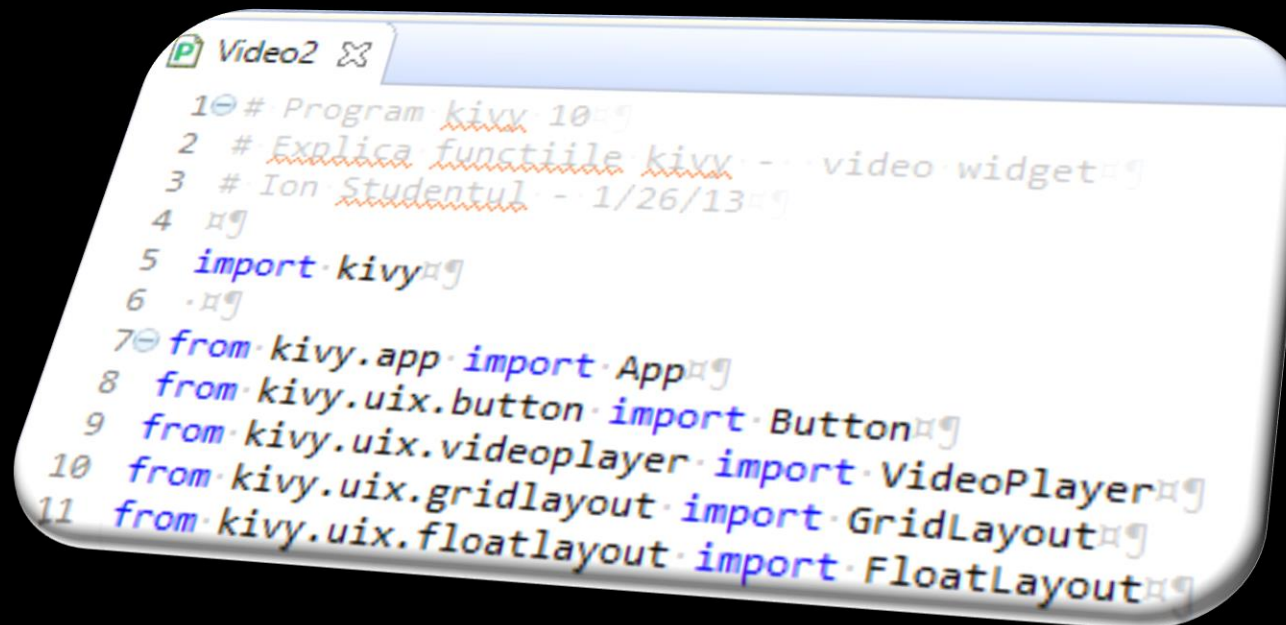
**Cu ajutorul proprietatii state oprim , pornim sau punem pauza.**

**Cu ajutorul seek setam un salt al rularii (intre 0-inceput si 1 - final).**

**Cu ajutorul proprietatii volume setam volumul intre 0-1**

```
Video1 ✖
1 # Program kivy 9
2 # Explica functiile kivy --- video widget
3 # Ion Studentul --- 1/26/13
4
5 import kivy
6
7 from kivy.app import App
8 from kivy.uix.videoplayer import VideoPlayer
9 from kivy.uix.gridlayout import GridLayout
10 from kivy.uix.button import Button
```

Urmatorul program face trecerea la formarea unui meniu.



```
1 # Program kivy 10
2 # Explica functiile kivy --- video widget
3 # Ion Studentul -- 1/26/13
4
5 import kivy
6
7 from kivy.app import App
8 from kivy.uix.button import Button
9 from kivy.uix.videoplayer import VideoPlayer
10 from kivy.uix.gridlayout import GridLayout
11 from kivy.uix.floatlayout import FloatLayout
```



Programul de mai sus este format din doua layout-uri. Unul este primar și ramane pe tot parcursul programului(`self.parent`). Al doilea layout este un child la primul și putem sa-l eliminam.(`layout1`, `layout 2` și `layout3`).

Initial rulam `self.Menu(None)`. Am adaugat `None` în interiorul apelarii deoarece dacă apelam aceasta metoda printr-un buton transmitem și butonul. Aici, avand în vedere ca nu folosim functionalitatea variabilei buton declarata la inceputul metodei `Menu`, putem trece `None`. Pentru ca aceste functii să creeze schimbarile grafice trebuie să returnam tot mereu `self.parent`. Acest lucru este necesar deoarece nu am mai creat o alta clasa care să importe toata functionalitatea ei asa cum am invatat în sedinta precedenta sau in exemplele precedente. Deci dacă foloseam o structura ca cea de mai jos, nu trebuia să returnam nimic.

```
class VideoApp(GridLayout):
```

```
    def __init__(self, **kwargs):  
        super(VideoApp, self).__init__(**kwargs)
```

Toate celelalte metode incep cu `self.parent.clear_widgets()`. Acesta linie sterge toate celelalte widget-uri. În interiorul metodei Menu regasim :

```
try:  
    self.video1.state="stop"  
except:  
    pass  
try:  
    self.video2.state="stop"  
except:  
    pass
```

Aceste linii au rolul de a incerca să seteze obiectele de tip video cu starea “stop”. Acesta practica este necesara deoarece chiar dacă stergem widget-urile de la un parinte, acestea inca exista initializate, deci nu sunt sterse. Prin urmare, fara aceste linii filmul nu se mai vedea, dar se auzea.

. În urmatoarele linii cream doua butoane ce au legate prin bind evenimentele de apasare a acestora la doua metode distincte.

```
button1 = Button(text='Play video1', font_size=14)
button1.bind(on_press=self.on_button_press1)
self.layout1.add_widget(button1) #add button
button2 = Button(text='Play video2', font_size=14)
button2.bind(on_press=self.on_button_press2)
self.layout1.add_widget(button2)
self.parent.add_widget(self.layout1)
```

Fiecare din aceste metode distincte (on\_button\_press1 sau on\_button\_press2 ) sunt identice ca formare; difera doar filmul rulat.



. Incepem prin a elimina widget-urile de la layout-ul self.parent. Apoi cream un nou layout care va deveni child la self. parent. Cream un buton și un widget de tip video. Ce vor deveni copiii layout-ului nou format.

```
def on_button_press1(self, buton):  
    self.parent.clear_widgets()  
    self.layout2 = GridLayout(cols=2)  
    menuBack = Button(text='Menu', font_size=14)  
    menuBack.size_hint = (0.1, 0.1)  
    menuBack.bind(on_press=self.Menu)  
    self.layout2.add_widget(menuBack)  
    self.video1 = VideoPlayer(source='softboy.avi', state='play')  
    self.layout2.add_widget(self.video1)  
    self.parent.add_widget(self.layout2)  
    return self.parent
```

- Primul widget avansat este Popup. Acest widget ofera un popup(fereastra mica ce anunta ceva).
- În mod normal acesta ocupa toata dimensiunea (1,1) unei ferestre.
- El este contruit din titlu ce prezinta acest popup și o zona content ce contine un alt widget.
- De asemenea am putea seta un titlu cu `Popup.title` și un text explicativ sub titlu cu `Popup.content`.
- Popup content poate fi și un alt widget, dar din pacate nu accepta decat un singur widget.
- Dacă dorim să deschidem un popup trebuie să apelam `open()`; dacă dorim să închidem un Popup trebuie să apelam `dismiss()`.



# WIDGET-URI AVANSATE: INFOACADEMY.NET POPUP

```
popup0    
1 # Program kivy13.py  
2 # Explica functiile kivy.popup.widget.py  
3 # Ion Studentul -- 1/26/13.py  
4  
5 from kivy.uix.gridlayout import GridLayout  
6 from kivy.app import App  
7 from kivy.uix.popup import Popup  
8 from kivy.uix.button import Button  
9 from kivy.uix.label import Label  
10
```

În `init` regasim un buton ce va ocupa toata suprafata layout-ului. Acest buton afiseaza textul “deschide un popup” și are legat un eveniment de apasarea lui; va rula metoda `un_popup`.

```
def un_popup(self, buton):  
    """Creaza un popup"""  
    inchide=Button(text='Inchide! ')  
    self.popup = Popup()  
    self.popup.title='Testam un popup'  
    self.popup.content=inchide  
    self.popup.open()  
    inchide.bind(on_press=self.inchide_popup)
```

Metoda `un_popup` creaza un buton numit `inchide` și un popup numit `self.popup`.

Setam apoi la popup un titlu (`self.popup.title='Testam un popup'`) și ca și continut adaugam acest buton. Deschidem apoi popup-ul cu metoda `open()`.

In ultima linie din aceasta metoda adaugam un eveniment la apasarea butonului ce va rula metoda `inchide_popup`.

```
def inchide_popup(self, Buton):
```

```
    """inchide popup"""
```

```
    self.popup.dismiss()
```

In metoda `inchide_popup` regasim linia `self.popup.dismiss()` ce va inchide popup-ul pentru a putea reveni la fereastra initiala a programului.

Vom modifica programul de mai sus pentru a putea adauga în interiorul content al popup-ului mai mult de un widget.

```
popup1 ✖  
1 # Program kivy13.py  
2 # Explica functiile kivy.popup.widget.py  
3 # Ion Studentul - 1/26/13.py  
4  
5 from kivy.uix.gridlayout import GridLayout  
6 from kivy.app import App  
7 from kivy.uix.popup import Popup  
8 from kivy.uix.button import Button  
9 from kivy.uix.label import Label  
10 from kivy.uix.boxlayout import BoxLayout
```

În `__init__` regăsim un buton ce va ocupa doar 0.8 din suprafața layout-ului. Acest buton afișează textul “deschide un popup” și are legat un eveniment de apăsarea lui; va rula metoda `un_popup`.

```
def __init__(self, **kwargs):  
    super(PopupApp, self).__init__(**kwargs)  
    self.cols = 1  
    buton1 = Button(text = "deschide un popup")  
    buton1.size = (0.8,0.8)  
    buton1.bind(on_press=self.un_popup)  
    self.add_widget(buton1)
```



În cadrul metodei `un_popup` regăsim cod pentru crearea unui buton numit `inchide` caruia i-am setat dimensiunea la 1,0.1 și cod pentru crearea unui label numit `eticheta`. Eticheta va afișa text-ul "Acesta este un popup informativ".

```
def un_popup(self, buton):  
    """Creaza un popup"""  
    inchide=Button(text='Inchide!')  
    inchide.size_hint = (1,0.1)  
    eticheta=Label(text = "Acesta este un popup informativ!")
```

Urmeaza să cream un layout de tip box numit layout2. Acesta va avea afisarea verticala și va avea ca și copii label-ul eticheta și buton-ul inchide. Vom adauga și un padding la layout-ului de 40 pentru aspect vizual.

```
layout2=BoxLayout()
```

```
layout2.orientation = "vertical"
```

```
layout2.add_widget(eticheta)
```

```
layout2.add_widget(inchide)
```

```
layout2.padding = 40
```

Setam apoi un popup numit self.popup cu titlu *'Testam un popup'*

Pentru ca să putem seta un size, vom seta size hint la None apoi vom seta size. Prin urmare vom forta dimensionarea widget-ului. Content-ul widget-ului va fi acest layout. Deschidem apoi popup-ul cu metoda open().

În ultima linie din această metoda adăugăm un eveniment la apăsarea butonului ce va rula metoda inchide\_popup.

```
self.popup = Popup()
self.popup.size_hint = (None, None)
self.popup.size = (400, 400)
self.popup.title = 'Testam un popup'
self.popup.content = layout2
self.popup.open()
inchide.bind(on_press=self.inchide_popup)
```

În metoda `inchide_popup` regăsim linia `self.popup.dismiss()` ce va închide popup-ul pentru a putea reveni la fereastra inițială a programului.

```
def inchide_popup(self, Buton):  
    """inchide popup"""  
    self.popup.dismiss()
```

Un click/tap în exteriorul widget-ului popup va genera închiderea widget-ului popup similar cu apăsarea butonului închide.

În metoda `inchide_popup` regăsim linia `self.popup.dismiss()` ce va închide popup-ul pentru a putea reveni la fereastra inițială a programului.

```
def inchide_popup(self, Buton):  
    """inchide popup"""  
    self.popup.dismiss()
```

Un click/tap în exteriorul widget-ului popup va genera închiderea widget-ului popup similar cu apăsarea butonului închide.



# WIDGET-URI AVANSATE: SCATTER

- In cele ce urmeaza vom discuta despre widget-ul Scatter. Acesta nu este decat un widget comportamental pentru un widget ce il are ca și copil. Acesta nu reprezinta un layout. Marimea copilului este setata separat. Codul integreaza în metoda init crearea unui scatter ce are ca și child un widget de tip imagine.

```
scatter0 ✕  
1 # Program kivy  
2 # Explica functiile kivy --- scatter.widget  
3 # Ion Studentul -- 1/26/13  
4  
5 from kivy.app import App  
6 from kivy.uix.scatter import Scatter  
7 from kivy.uix.gridlayout import GridLayout  
8 from kivy.uix.image import Image  
9  
10 class Scatter_test(GridLayout):  
11
```

# WIDGET-URI AVANSATE: SCATTER

- Dacă dam click dr. într-un loc putem să rotim sau scalam imaginea. Când terminăm pentru a elimina acel punct roșu facut cu click dreapta și pentru a ne opri din scalat/rotit atunci dam click stanga pe acel punct roșu. De asemenea vedem că dacă dam click stanga pe imagine aceasta se transpune în aceeași poziție ca la pornirea programului.
- Vom încerca să upgradăm un pic acest widget. Prin urmare ca imaginea să nu se mai transpună în aceeași poziție ca la început când dam click pe ea folosim proprietatea `auto_bring_to_front` setat pe `False`. Dacă `scatter.auto_bring_to_front` property este setat `True` (stare default) scatter widget este eliminat și adăugat de fiecare dată la părinte când este atins (sau click stanga). Este util când manipulezi multiple scatter și nu dorești ca cel activ să fie parțial ascuns.
- Pentru a scala acest widget folosim proprietatea `scale`. Aceasta scalează dimensiunile obiectelor din acest widget automat. A nu se folosi `size`, ci `scale`.

# WIDGET-URI AVANSATE: SCATTER

In metoda init cream un layout numit self.layout1.

```
self.layout1 = GridLayout(cols=1)
```

Apoi cream o eticheta ce afiseaza "unghi: 0" .

```
self.eticheta = Label (text = "unghi: 0")
```

Urmeaza crearea unei imagini cu sursa "nature3.jpg" și un buton numit self.buton1 ce afiseaza textul "Afla Rotatie". Self.buton are proprietate size\_hint setata la (1,0.05), deci va fi un buton subtire.

```
self.imag1 = Image(source="nature3.jpg")
```

```
self.buton1=Button(text="Afla rotatie")
```

```
self.buton1.size_hint = (1,0.05)
```

# WIDGET-URI AVANSATE: SCATTER

Copiii `self.layout1` sunt eticheta și imaginea.

```
self.layout1.add_widget(self.eticheta)
```

```
self.layout1.add_widget(self.imag1)
```

Crearea unui scatter este urmatorul pas. Setăm proprietatea `auto_bring_to_front` la `False`. Adăugăm ca și copil al scatter-ului `layout1`.

```
self.scatter1 = Scatter()
```

```
self.scatter1.auto_bring_to_front = False
```

```
self.scatter1.add_widget(self.layout1)
```

# WIDGET-URI AVANSATE:SCATTER

Adaugam self.scatter1 și self.buton1 ca și copii la self.

```
self.add_widget(self.scatter1)
```

```
self.add_widget(self.buton1)
```

Setam ca de fiecare data cand butonul se apasa să se ruleze o metoda numita roteste\_ma

```
self.buton1.bind(on_press = self.roteste_ma)
```



# WIDGET-URI AVANSATE: SCATTER

Metoda `roteste_ma` presupune update-ul textului afisat de eticheta cu gradul de rotatie al `scatter1`.

```
def roteste_ma(self, buton):  
    """modifica eticheta"""  
    self.eticheta.text = "unghi: " + str(int(self.scatter1.rotation))
```

# WIDGET-URI AVANSATE: TABBEDPANEL

Widget-ul TabbedPanel administrează diferite widget-uri în tab-uri. El este contruit din header area pentru butonul de tab actual și un content area ce conține un singur widget. Dacă dorim să punem mai multe widget-uri atunci trebuie să cream un layout. Înainte de a implementa trebuie să înțelegem ce este un TabbedPanelHeader.

Un tab individual este numit TabbedPanelHeader. Acesta este un buton special ce conține un content. Pentru a folosi acest TabbedPanelHeader trebuie să îl adăugăm la TabbedPanel.

```
tp = TabbedPanel()
```

```
th = TabbedPanelHeader(text='Tab2')
```

```
th.content = un_alt_widget
```

```
tp.add_widget(th)
```

Este important să reținem că primul tab este creat automat și adăugat la instanțiere și se numește Default tab.

# WIDGET-URI AVANSATE: TABBEDPANEL

Este important să reținem ca primul tab este creat automat și adăugat la instanțiere și se numește **Default tab**.

În metoda `init` regăsim crearea unui obiect de tip `TabbedPanel`

```
tb_panel= TabbedPanel()
```

Creăm apoi un tabbed header numit `th_text_head`. Acesta va avea ca și conținut (content) un label ce afișează mesajul "Un text".

```
th_text_head = TabbedPanelHeader(text='Text tab')
```

```
th_text_head.content= Label(text='Un text')
```

# WIDGET-URI AVANSATE: TABBEDPANEL

Urmatorul tabbed header este numit `th_img_head`. Acesta va avea ca și continut (content) o imagine cu sursa `"nature4.jpg"` de o anumita marime și cu o anumita pozitie.

```
th_img_head= TabbedPanelHeader(text='Image tab')
```

```
th_img_head.content= Image(source='nature4.jpg',pos=(400, 100), size=(400, 400))
```

Urmatorul tabbed header se numeste `th_btn_head`. Acesta va avea ca și continut (content) un buton de o anumita marime.

```
th_btn_head = TabbedPanelHeader(text='Button tab')
```

```
th_btn_head.content= Button(text='Acesta este un buton',font_size=20,  
                             size_hint=(0.8, 0.8))
```

# WIDGET-URI AVANSATE: TABBEDPANEL

Pasul urmator este să adaugam fiecare tabbed header la tabbed panel. Ulterior tabbed panel va fi adaugat ca copil la layer-ului self.

```
tb_panel.add_widget(th_text_head)  
tb_panel.add_widget(th_img_head)  
tb_panel.add_widget(th_btn_head)  
self.add_widget(tb_panel)
```



# WIDGET-URI AVANSATE: TABBEDPANEL

Vom modifica exemplu anterior pentru ca un tab sa permita mai multe widget-uri si sa setam default tab. Setarea unui fundal se realizeaza cu ajutorul proprietatii `background_image` a `tabbed panel`.

```
tb_panel= TabbedPanel()
```

```
tb_panel.background_image = "fundal2.jpg"
```

De asemenea , putem modifica default tab cu un text header și un continut.

```
tb_panel.default_tab_text = "tab-ul default"
```

```
tb_panel.default_tab_content = Image(source='infoacademy3.gif',pos=(200,100), size=(200, 200))
```

In cazul în care dorim să adaugam mai multe widget-uri intr-un singur content procedam la fel ca la popup, adica cream un layout care contine tot ce dorim să adaugam în content.

# WIDGET-URI AVANSATE: TABBEDPANEL

Prin urmare mai jos regasim layout-ul numit layout care contine doua widget-uri (eticheta și buton). Apoi layout este adaugat ca și content al tabbedPanelHeader creat anterior.

```
th_btn_head = TabbedPanelHeader(text='Button tab')
layout= GridLayout(cols=1)
eticheta=Label(text='tab-ul cu buton', font_size = 40)
buton=Button(text='Acesta este un buton',font_size=20)
layout.add_widget(eticheta)
layout.add_widget(buton)
th_btn_head.content= layout
th_btn_head.content.padding = 200
```

# WIDGET-URI AVANSATE: TABBEDPANEL

```
TabbledPannel1 ✕  
1 from kivy.app import App  
2 from kivy.uix.tabbedpanel import TabbedPanel  
3 from kivy.uix.tabbedpanel import TabbedPanelHeader  
4 from kivy.uix.button import Button  
5 from kivy.uix.label import Label  
6 from kivy.uix.image import Image  
7 from kivy.uix.gridlayout import GridLayout  
8
```

# WIDGET-URI AVANSATE: CAROUSEL

- Urmatorul widget este carousel. Acesta poate afisa ca intr-un carusel imaginile pe rand. De asemenea poate afisa și alte widget-uri.
- Tot ce trebuie să facem este să glisam cu mouse-ul sau cu degetul (touch screen) pe ecran.
- Cu ajutorul proprietatii `anim_move_duration` a obiectului de tip carousel vom seta durata trecerii de la un slide la altul: Default 0.5 secunde.
- Chiar dacă nu este utilizată aici proprietatea `loop` a obiectului de tip carousel va permite ca imaginile să fie glisate chiar dacă lista se termina, reluand lista de obiecte afisate.

# WIDGET-URI AVANSATE:CAROUSEL

```
carousel ✕  
1 from kivy.app import App  
2 from kivy.uix.carousel import Carousel  
3 from kivy.uix.image import Image  
4 from kivy.uix.label import Label  
5  
6 class Example1(App):
```



# LIMBAJUL KIVY

- Un alt atu este crearea în mod automat a widget-urilor și redimensionarea lor fara a fi nevoie de artificii de design. De asemenea separa codul în doua parti, partea logica și partea GUI.
- Exista doua metode de a incarca cod kivy în aplicatia ta:
  - Prin conventia de nume: Kivy cauta un fisier cu extensia .kv cu acelasi nume ca și clasa App, dar scrisa doar cu caractere mici și fara App la final în cazul în care se termina cu App. Exemplu: AplicatiaMeaApp => aplicatiamea.kv
  - Metoda Builder: poti spune kivy să incarce un fisier sau un șir de caractere  
Exemplu:
    - `Builder.load_file('calea/catre/fisier.kv')`
    - `Builder.load_string(sir_kv)`
  - Unde `sir_kv` este o variabila de tip șir de caractere.

# LIMBAJUL KIVY

- Kivy seamana cu limbajul CSS (mai multe detalii gasiti [aici](#)) .
- Se aplica un set de reguli în limbajul kivy. O regula se aplica la un widget specific și il modifica într-un anumit fel.
- Poti utiliza regulile pt. a specifica proprietati (cum ar fi size, size\_hint etc.) sau comportament al widget-urilor (adugare de child's sau bindings). De asemenea, poti utiliza limbajul kivy să cream întreaga interfata grafica. Un fisier ky trebuie să contina doar un singur root widget. Dynamic Classes este un alt concept ce este posibil în limbajul kivy. Acesta permite să cream noi widget-uri și reguli pe parcurs, fara nici o declaratie Python.

# LIMBAJUL KIVY

Iata cum scriem cod kivy.

- Prima linie trebuie să fie tot mereu:

```
#:kivy `1.0`
```

Unde 1.0 este versiunea kivy, cea mai recenta este 1.8, puteti utiliza ce varianta doriti. Apoi urmeaza:

```
# definitia sintaxei unei reguli. De retinut ca multiple reguli pot imparti
```

```
# aceleasi definitii ca in CSS.
```

```
<Regula1, Regula 2>:
```

```
    # .. definitii ..
```

```
<Regula3>:
```

```
    # .. definitii..
```

```
# Sintaxa pentru crearea unui root widget
```

```
RootClassName:
```

```
    # .. definitii..
```

# LIMBAJUL KIVY: EXEMPLU

```
from kivy.base import runTouchApp
```

```
from kivy.lang import Builder
```

```
kv = """
```

```
Button:
```

```
    text: 'test'
```

```
    on_press: print "test"
```

```
"""
```

```
if __name__ == '__main__':
```

```
    runTouchApp(Builder.load_string(kv))
```

Exista multiple pagini ce ofera support pentru kivy, discutii de tip “how to” sau sugestii pentru a scrie kivy. Iata cateva pe care trebuie să le vizitati:

- Lista de mail kivy-users: <https://groups.google.com/forum/#!forum/kivy-users>
- Canalul IRC #kivy pe irc.freenode.net.
- github issue tracker (Pentru bug-uri și cerere de features: <https://github.com/kivy/kivy/issues>).
- <http://kivy.org/docs/gettingstarted/examples.html>
- <http://karanbalkar.com/tag/kivy/>



VA MULTUMESC PENTRU PARTICIPARE

**La revedere!**