SEDINTA 8 – KIVY AVANSAT

Widget-uri partea a doua

Cred ca una din intrebarile pe care le aveti legate de aplicatiile pe care le realizati este cum să setez o marime a ferestrei, astfel ca fiecare aplicatie să se vada asa cum doriti.

```
# Program ico si dimensiune
# Explica functiile kivy
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.config import Config



class LoginScreen(GridLayout):

    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.cols = 2
        self.add_widget(Label(text='Utilizator '))
        self.username = TextInput(multiline=False)
        self.add_widget(self.username)
        self.add_widget(Label(text='Parola'))
        self.password = TextInput(password=True, multiline=False)
        self.add_widget(self.password)
        self.password.bind(on_text_validate= self.verific_user_si_parola)

    def verific_user_si_parola(self,t):
        text_user = self.username.text
        text_pass = self.password.text
        if (text_user == "test" and text_pass=="test"):
            self.clear_widgets()
            self.add_widget(Label(text='Bine ai venit!'))
        else:
            self.username.select_all()
            self.username.delete_selection()
            self.password.select_all()
            self.password.delete_selection()



class AplicatiePersonalizata(App):

    def build(self):
        self.icon ="python1.ico"
        return LoginScreen()
```

```
#seteaza dimensiunea ferestrei
Config.set('graphics', 'width', '300')
Config.set('graphics', 'height', '300')

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

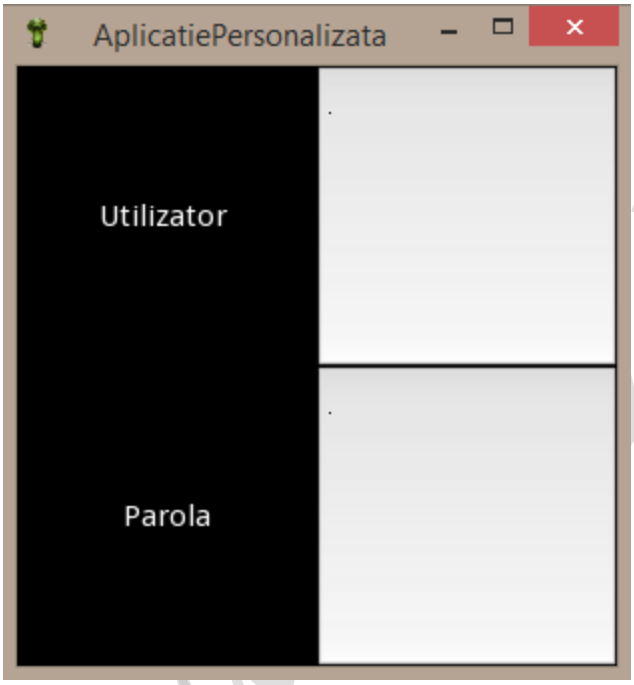


Fig.14

Asa cum se poate vedea fereastra are dimensiuni considerabili mai mici.Acest lucru se poate face cu urmatoarele linii:

```
from kivy.config import Config
Config.set('graphics', 'width', '300')
Config.set('graphics', 'height', '300')
```

Prin urmare am importat Config, cu ajutorul caruia am setat dimensiunile aplicatiei inainte de a a rula aplicatia.

Tot în aplicatia de mai sus putem vedea ca in corpul metodei build putem seta și iconita aplicatiei. Aceata iconita reprezinta un piton (sarpe) și poate fi vazuta în Fig.14 în coltul din stanga sus. Mai jos se regaseste extrasa linia care seteaza iconita.

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

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.

Iata un prim exemplu ce implica label-ul Image:

```
# Program kivy1
# Explica functiile kivy -  image widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.image import Image

class ImagineaMea(GridLayout):

    def __init__(self, **kwargs):
        super(ImagineaMea, self).__init__(**kwargs)
        self.cols = 1
        self.imag1 = Image(source="infoacademy2.gif")
        self.add_widget(self.imag1)
        self.imag2 = Image(source="infoacademy3.gif")
        self.add_widget(self.imag2)
        self.imag3 = Image(source="infoacademy4.gif")
        self.add_widget(self.imag3)

class AplicatiePersonalizata(App):

    def build(self):
        return ImagineaMea()

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

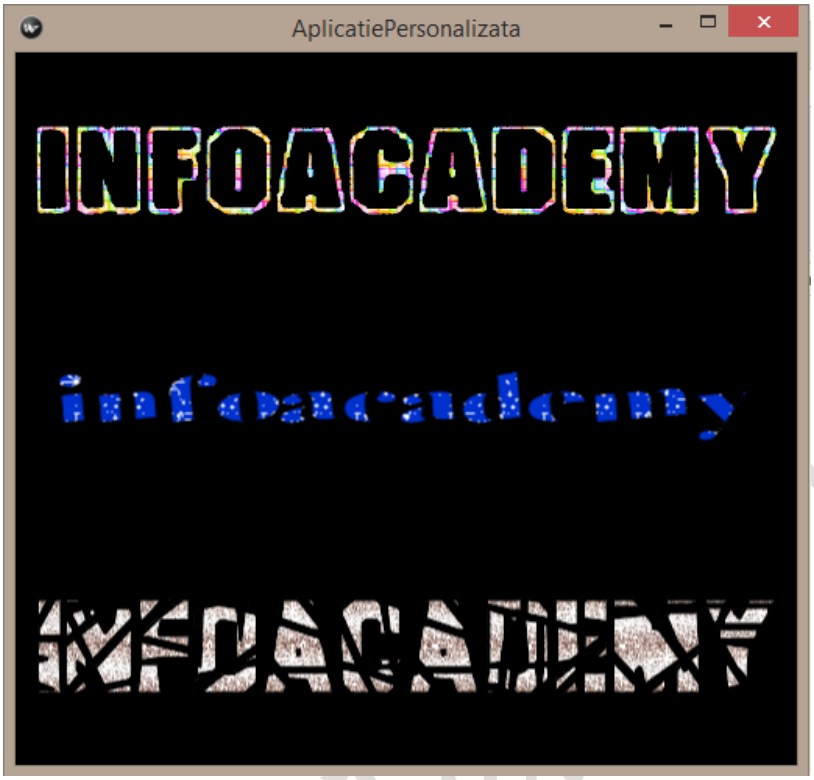


Fig.1

In programul de mai sus regasim în init crearea a trei obiecte de tip imagine (*self*.imag1, *self*.imag2 și *self*.imag3) și adaugarea lor la layout-ul self.

```
        self.imag1 = Image(source="infoacademy2.gif")
        self.add_widget(self.imag1)
        self.imag2 = Image(source="infoacademy3.gif")
        self.add_widget(self.imag2)
        self.imag3 = Image(source="infoacademy4.gif")
        self.add_widget(self.imag3)
```

Kivy poate incarca diverse tipuri de imagini cum ar fi PNG, GIF sau JPG. Totusi în exemplul de mai jos vom reutiliza codul programului de mai sus pentru a se vedea cum reusim să manipulam imaginile statice sau dinamice(GIF).

```
# Program kivy2
# Explica functiile kivy -  image widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.image import Image

class ImagineaMea(GridLayout):

    def __init__(self, **kwargs):
```

```
        super(ImagineaMea, self).__init__(**kwargs)
        self.cols = 1
        self.imag1 = Image(source="infoacademy2.gif")
        self.imag1.opacity = 0.4 #default este 1
        self.add_widget(self.imag1)
        self.imag2 = Image(source="infoacademy3.gif")
        self.imag2.anim_delay = 0.04 # default este 0.25 (4fps)
        self.add_widget(self.imag2)
        self.imag3 = Image(source="infoacademy4.gif")
        self.imag3.color = (0,1,0,1)
        self.add_widget(self.imag3)

class AplicatiePersonalizata(App):

    def build(self):
        return ImagineaMea()

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

Fig.2

In programul de mai sus regasim în init crearea a trei obiecte de tip imagine (`self.imag1`, `self.imag2` și `self.imag3`) și adaugarea lor la layout-ul self.

```
        self.imag1 = Image(source="infoacademy2.gif")
        self.add_widget(self.imag1)
        self.imag2 = Image(source="infoacademy3.gif")
        self.add_widget(self.imag2)
        self.imag3 = Image(source="infoacademy4.gif")
```

```
        self.add_widget(self.imag3)
```

Pe langa acesta vom modifica anumite proprietati ale acestor obiecte.Cu ajutorul proprieatatii opacity setam gradul de opacitate al pozei. Standard aceasta este 1 adica opaca. Dacă scadem la 0 atunci este toatal 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 micsoram animatia va fi mai rapida, dacă o cresteam 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. Sandard aceasta nu are o culoare definita.

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

De asemenea este destul de important să reducem dimensiunea unei poze adaugate. Aceasta practica se poate face cu size_hint.

```
        self.imag2.size_hint_x = 0.1
        self.imag2.size_hint_y = 0.1
```

Un exemplu se regaseste mai jos.

```
# Program kivy3
# Explica functiile kivy -  image widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.image import Image

class ImagineaMea(GridLayout):

    def __init__(self, **kwargs):
        super(ImagineaMea, self).__init__(**kwargs)
        self.cols = 1
        self.imag1 = Image(source="jaguar.jpg")
        self.add_widget(self.imag1)
        self.imag2 = Image(source="jaguar.jpg")
        self.imag2.size_hint_x = 0.1
        self.imag2.size_hint_y = 0.1
        self.add_widget(self.imag2)

class AplicatiePersonalizata(App):
```

```
    def build(self):
        return ImagineaMea()

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

Fig. 3

Cred ca fiecare dintre voi v-ati intrebat cum să pun o imagine de fundal aplicatiei mele?

Iata mai jos un exemplu ce realizeaza acest lucru.

```
# Program kivy4
# Explica functiile kivy - image background
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.image import Image
from kivy.uix.button import Button

class CustomLayout(FloatLayout):

    def __init__(self, **kwargs):
        # make sure we aren't overriding any important functionality
        super(CustomLayout, self).__init__(**kwargs)
        self.imag1 = Image(source="fundal.jpg")
        self.imag1.opacity = 0.7
```

```
        self.add_widget(self.imag1)
        self.but1 =Button(text = "Butonul 1",blod =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",blod =True, background_color =
(0,0,1,1))
        self.but2.pos = (350,200)
        self.imag1.add_widget(self.but2)
        self.but3 =Button(text = "Butonul 3",blod =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)


class MainApp(App):

    def build(self):
        return CustomLayout()

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

Fig. 4

Asa cum se poate vedea și în Fig.4, dar și în cod am atasat o imagine la layout și am facut celelalte elemente copiii la imagine. Astfel am creat un fundal.

Prin urmare în metoda init regasim crearea unei imagini ce are ca sursa fisierul fundal.jpg ce se regaseste în aceasi locatie cu programul (locatie relativa). Urmatorul pas este să modificam o proprietate a imaginii și anume opacity (gradul de opacitate). Asftel imaginea devine complet opaca la 1 și complet transparenta la 0. Aici se regaseste valoarea 0.7. Apoi adaugam acesta imagine la layout-ul parinte.

```
        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 proprieatatea bold la True (facem textul ingrosat) și aceasi valoarea culorii - albastru(0,0,1,1). Toate butoanele sunt adaugate ca copiii la obiectul Imagine 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",blod =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",blod =True, background_color =
(0,0,1,1))
        self.but2.pos = (350,200)
        self.imag1.add_widget(self.but2)
        self.but3 =Button(text = "Butonul 3",blod =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 redeseneare 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 copiii imag1 mostenensc caracteristici precum opacity.

Pentru a rezolva aceasta problema codul se transforma în programul de mai jos:

```
# Program kivy5
# Explica functiile kivy - image background
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.graphics import Color, Rectangle
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.image import Image
from kivy.uix.button import Button
```

```
class CustomLayout(FloatLayout):

    def __init__(self, **kwargs):
        super(CustomLayout, self).__init__(**kwargs)

        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


class MainApp(App):

    def build(self):
        c = CustomLayout()
        self.but1 =Button(text = "Butonul 1",blod =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",blod =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",blod =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)
        return c

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

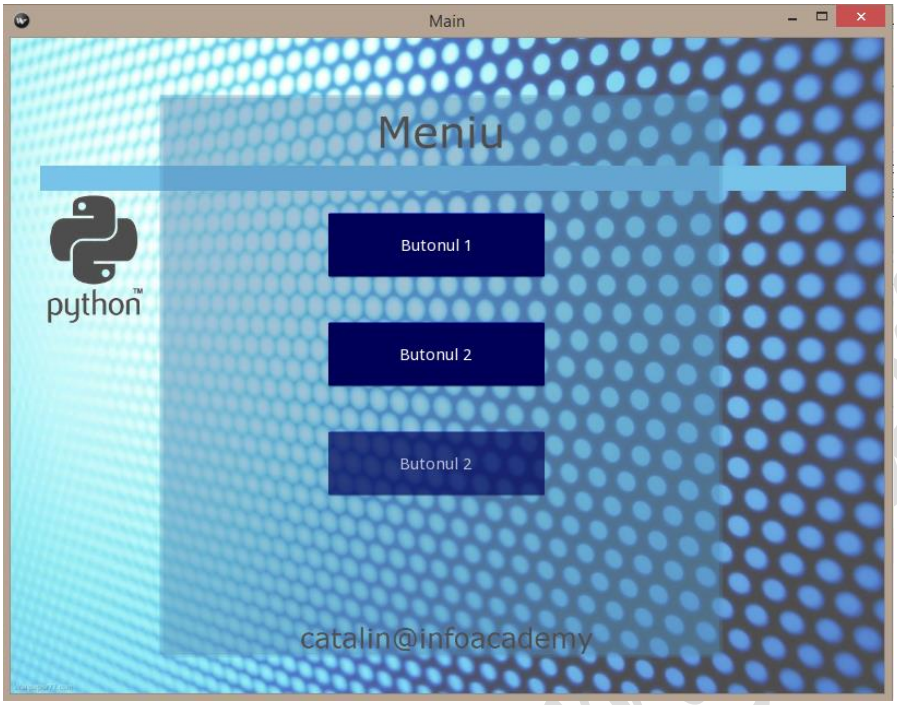


Fig.5

Acum se poate vedea ca grafica arata mult mai bine deoarece butoanele vor fi create la ce dimensiune ne-am ales. Acest lucru se datoreaza eliminarii mostenirii. Widget-ul de tip imagine nu permite setarea unei dimensiuni în comportamentul standard. Acest lucru se reflecta în lipsa butoanelor de a avea opacitatea setata la 0.7. Totusi vedem ca acum aceasta caractersitica poate fi modificata la cerere,asa cum am facut în cazul but3.

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
```

In MainApp sub metoda build incepem prin a apela CostumLayout, deci ne va rezulta un layout ce are o imagine ce poate fi redimensionata.

```
c = CustomLayout()
```

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

```
self.but1 =Button(text = "Butonul 1",blod =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",blod =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",blod =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
```

O alta modalitate de a avea posibilitatea de a scala butoanele cand acestea sunt adaugate ca copiii la o imagine este sa setam asupra copiilor urmatoarele atribute:

Copil.size_hint= (None,None)

Copil.size=(val_x, val_y)

Unde val_x este valoarea aplicata pe orizontala, respectiv val_y este valoarea aplicata pe verticala a copilului.

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).

```
# Program kivy4
# Explica functiile kivy -  slider widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.slider import Slider
from kivy.uix.togglebutton import ToggleButton
from kivy.uix.label import Label
muzica_activa = 0

class SliderulMeu (GridLayout):
```

```
    def __init__(self, **kwargs):
        super(SliderulMeu, self).__init__(**kwargs)
        self.cols = 1
        self.arata_volum = Label (text = "volum: 10")
        self.add_widget(self.arata_volum)
        self.slide_muzica = Slider(min=0, max=100, value=10)
        self.slide_muzica.padding = 20
        self.slide_muzica.orientation="horizontal" # alte optiuni 'vertical'
        self.slide_muzica.step = 10
        self.add_widget(self.slide_muzica)
        self.slide_muzica.bind(value=self.valoare_volum)

    def valoare_volum (self,x,y):
        '''utilziat pentru a vedea volumul'''
        self.arata_volum.text = "volum: "+str(int(self.slide_muzica.value))

class AplicatiePersonalizata(App):

    def build(self):
        return SliderulMeu()


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

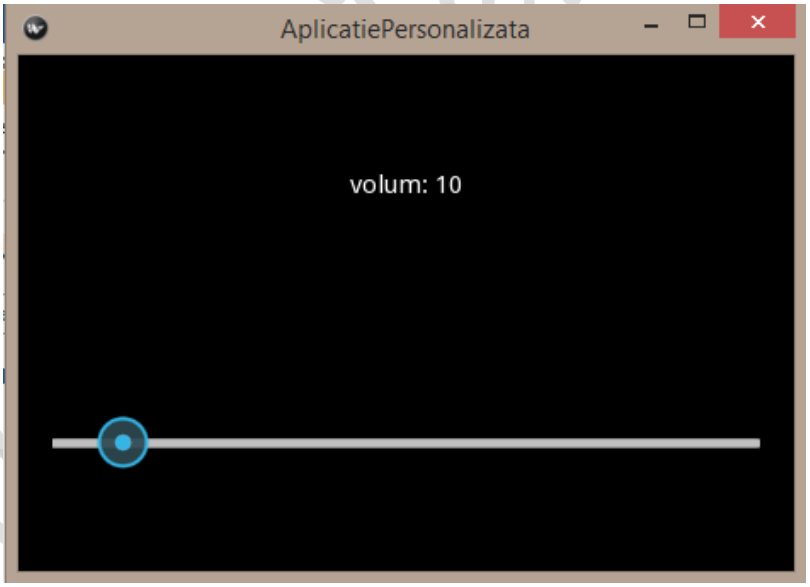


Fig.6

In clasa SliderulMeu sub metoda init regasim crearea unui label ce indica „volum:10”. Acest label este adaugat la layout-ul self.

```
        self.arata_volum = Label (text = "volum: 10")
        self.add_widget(self.arata_volum)
```

In urmatorul pas cream un slider.

```
        self.slide_muzica = Slider(min=0, max=100, value=10)
        self.slide_muzica.padding = 20
        self.slide_muzica.orientation="horizontal" # alte optiuni 'vertical'
        self.slide_muzica.step = 10
        self.add_widget(self.slide_muzica)
```

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, deci din 10 in zece putem selecta.

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 value deoarece aceasta isi schimba valoarea la miscarea slider-ului. Astfel la miscarea slider-ului vom rula metoda valoare_volum.

```
self.slide_muzica.bind(value=self.valoare_volum)
```

Sub metoda valoare_volum regasim updatarea label-ului arata_volum cu valoarea data de slider.

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

In urmatorul exemplu vom lega un slider de un toggle button. Astfel va fi activ doar dacă starea toggle button este ridicata.

```
# Program kivy4
# Explica functiile kivy -  slider widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.slider import Slider
from kivy.uix.togglebutton import ToggleButton
from kivy.uix.label import Label
muzica_activa = 0

class SliderulMeu(GridLayout):

    def __init__(self, **kwargs):
        super(SliderulMeu, self).__init__(**kwargs)
        self.cols = 1
        self.padding = 150
        self.toggle1 = ToggleButton(text="muzica")
```

```
        self.toggle1.background_normal = "on.png"
        self.toggle1.background_down = "off.png"
        self.add_widget(self.toggle1)
        self.arata_volum = Label (text = "volum: 10")
        self.add_widget(self.arata_volum)
        self.slide_muzica = Slider(min=0, max=100, value=10)
        self.slide_muzica.step = 5
        self.slide_muzica.orientation="horizontal" # alte optiuni 'vertical'
        self.add_widget(self.slide_muzica)
        self.toggle1.bind(on_press=self.dezactiveaza_volum)
        self.slide_muzica.bind(value=self.valoare_volum)

    def valoare_volum (self,x,y):
        '''utilziat pentru a vedea volumul'''
        self.arata_volum.text = "volum: "+str(int(self.slide_muzica.value))

    def dezactiveaza_volum (self,x):
        '''utilziat pentru a acctiva sau a dezactiva slider-ul'''
        global muzica_activa
        if (muzica_activa %2 == 0) :
            self.slide_muzica.disabled =True
        else:
            self.slide_muzica.disabled =False
            self.slide_muzica.value = 0
        muzica_activa += 1

class AplicatiePersonalizata(App):

    def build(self):
        return SliderulMeu()

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

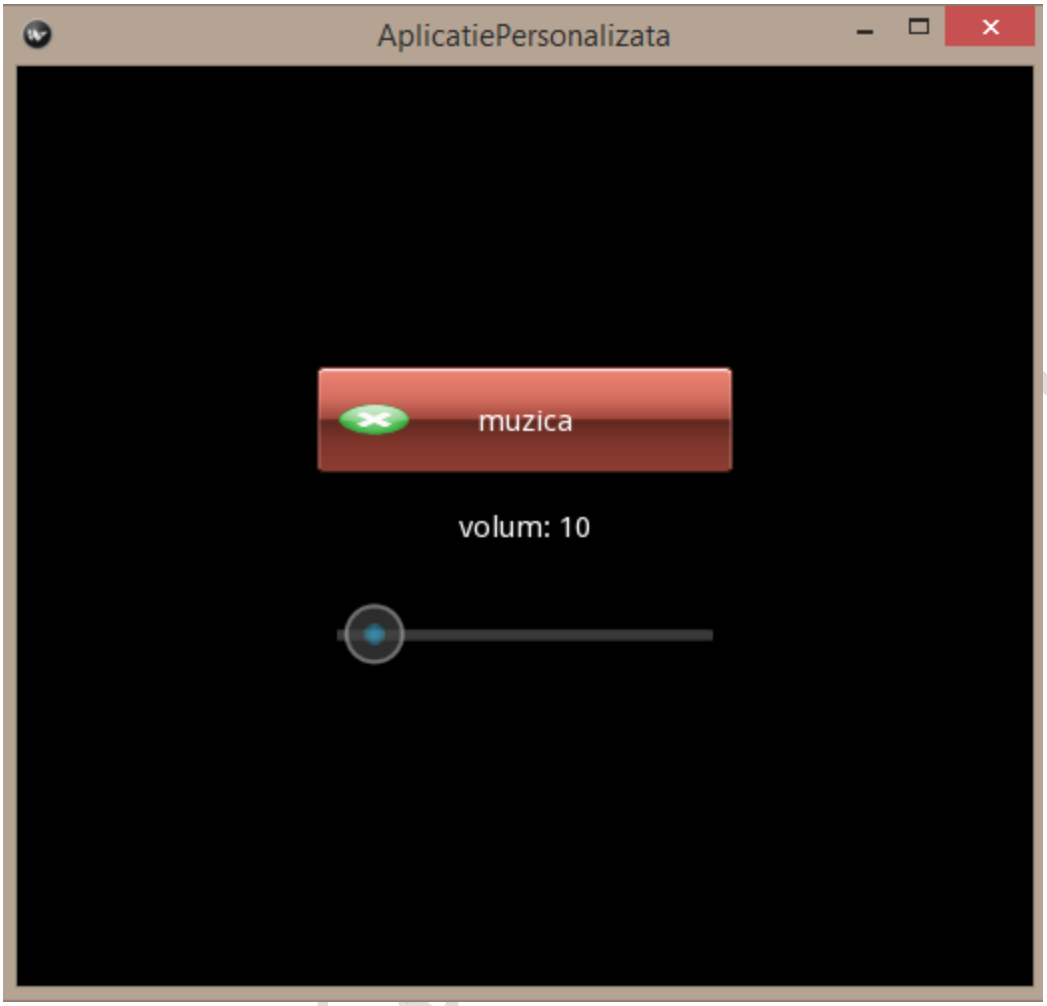


Fig.7

Incepem prin a crea o variabila numita muzica_activa=0. Aceasta va avea rolul unei evidente a starii butonului toggle.

Putem observa ca padding-ul a fost mutat la nivel de layout.

```
self.padding = 150
```

A fost introdus un toggle button caruia ii modificam backgrund-ul neapasat (normal) și apasat (down) cu doua imagini png. Vom adauga acest toggle buton la layout. De asemenea, un eveniment este legat la schimbarea starii ruland metoda dezactiveaza_volum la oricare apasare (activare sau dezactivare).

```
self.toggle1 = ToggleButton(text="muzica")
self.toggle1.background_normal = "on.png"
self.toggle1.background_down = "off.png"
self.add_widget(self.toggle1)
self.toggle1.bind(on_press=self.dezactiveaza_volum)
```

In urmatorul pas cream un slider.

```
self.slide_muzica = Slider(min=0, max=100, value=10)
self.slide_muzica.step = 5
self.slide_muzica.orientation="horizontal" # alte optiuni 'vertical'
self.add_widget(self.slide_muzica)
self.slide_muzica.bind(value=self.valoare_volum)
```

Pentru a seta un range trebuie să initializam o valoare 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 5.

Adaugam slide_muzica la layout-ul self. La fiecare miscare a slide-ului rulam metoda valoare_volum data de evenimentul value :

```
self.slide_muzica.bind(value=self.valoare_volum)
```

In metoda dezactiveaza_volum dorim să modificam valoarea variabilei globale muzica_activa, astfel ca folosim cuvantul cheie global. Astfel dacă aceasta valoare se imparte fara rest la 2 atunci putem să dezactivam slide-ul. În caz contrar il activam,si ii setam valoarea la 0. Modificarea valorii va duce implicit la rularea metodei valoare_volum. Apoi vom incrementa variabila globala muzica activa.

```
global muzica_activa
if (muzica_activa %2 == 0) :
    self.slide_muzica.disabled =True
else:
    self.slide_muzica.disabled =False
    self.slide_muzica.value = 0
muzica_activa += 1
```

Deci metoda dezactiveaza_volum va trece widget-ul slider în disabled (nu va putea fi utilizat) dacă butonul este în starea de oprit. La repornire va seta volumul la zero.

Sub metoda valoare_volum regasim updatarea label-ului arata_volum cu valoarea data de slider.

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

As dori să realizati un exercitiu simplu. Doresc să inlocuiti variabila globala muzica_activa cu state-ul butonului toggle. Prin urmare dacă self.toggle1.state este “down” sau este ”normal” să realizeze acele schimbari de disabled.

Urmatorul widget este Switch. Switch este un buton cu doua stari: on si off. Nu este cu mult diferit de toogle 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. Astfel, în loc de toggle buton, vom folosi acum un widget de tip switch.

```
# Program kivy4
# Explica functiile kivy -  slider widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.gridlayout import GridLayout
from kivy.uix.slider import Slider
from kivy.uix.switch import Switch
from kivy.uix.label import Label
muzica_activa = 0

class Switch_implementare(GridLayout):

    def __init__(self, **kwargs):
        super(Switch_implementare, self).__init__(**kwargs)
        self.cols = 1
        self.padding = 150
        self.switch1 = Switch()
        self.switch1.active =True
        self.add_widget(self.switch1)
        self.arata_volum = Label (text = "volum: 10")
        self.add_widget(self.arata_volum)
        self.slide_muzica = Slider(min=0, max=100, value=10)
        self.slide_muzica.step = 5
        self.slide_muzica.orientation="horizontal" # alte optiuni 'vertical'
        self.add_widget(self.slide_muzica)
        self.switch1.bind(active=self.dezactiveaza_volum)
        self.slide_muzica.bind(value=self.valoare_volum)

    def valoare_volum (self,x,y):
        '''utilziat pentru a vedea volumul'''
        self.arata_volum.text = "volum: "+str(int(self.slide_muzica.value))

    def dezactiveaza_volum (self,x,y):
        '''utilziat pentru a acctiva sau a dezactiva slider-ul'''
        global muzica_activa
        if (muzica_activa %2 == 0) :
            self.slide_muzica.disabled =True
        else:
            self.slide_muzica.disabled =False
            self.slide_muzica.value = 0
        muzica_activa += 1

class AplicatiePersonalizata(App):

    def build(self):
        return Switch_implementare()

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

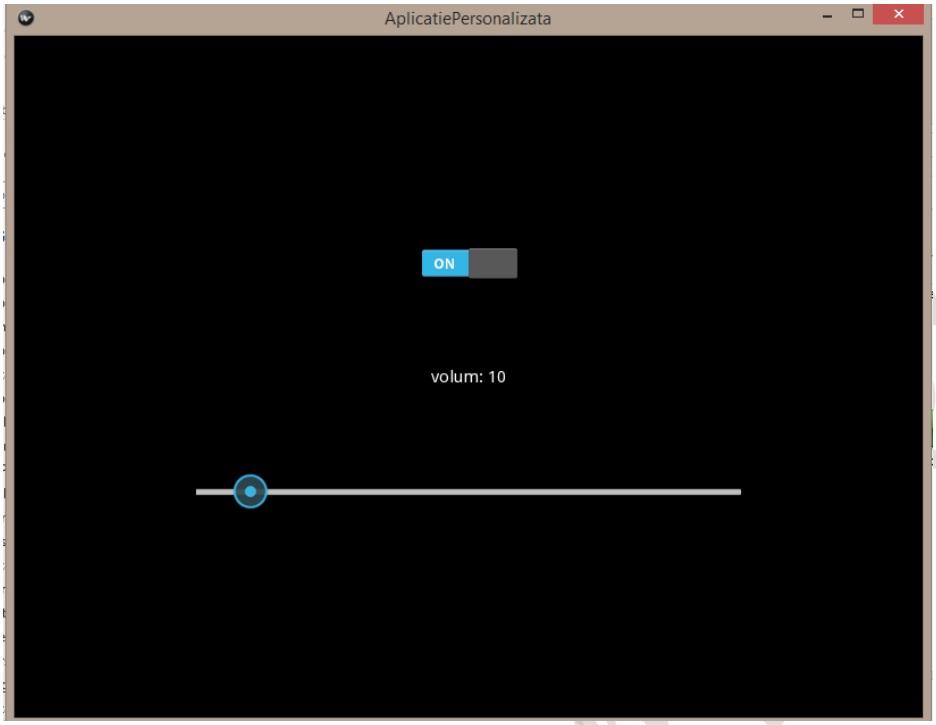


Fig. 8

Asa cum se poate vedea în programul de mai sus, 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).

Bineinteles ca ar trebui să avem și un widget care permite rularea unui sunet sau a unei melodii. Pentru a utiliza acest widget cream un obiect de tip SoundLoader. Veti regasi un exemplu în cele ce urmeaza.

```
# Program kivy11
# Explica functiile kivy - audio widget
# Ion Studentul - 1/26/13

from kivy.core.audio import SoundLoader
from kivy.uix.gridlayout import GridLayout
from kivy.app import App

class VideoPlayerApp(App):

    def build(self):
        layout= GridLayout()
        cols=1
        sound = SoundLoader.load('JO_-_09_-_Fortitude.wav')
        if sound:
            sound.play()
            print "Sursa fisierului este", sound.source
            print "Lungimea fisierului este ", sound.length
```

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

Cu ajutorul importului de SoundLoader vom putea rula o melodie. Mai jos se regaseste linia extrasa din program:

```
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 accest program o figura deaorece nu este nimic de afisat cu exceptia unei ferestre negre.

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

Incarcarea și rularea fisierelor de sunet sunt date de GStremer pentru windows sau linux. În versiunea Kivy 1.9 putem reda fisiere precum ogg și mp3. Din pacate nu functioneaza cu mp3 cu instalarea default datorita unor plugin-uri lipsa în packeage-ul default (varianta pachet de tip zip care necesita ca progamul să rulat prin dublu click sau prin sendto). Mai multe detalii despre versiunea de GStreamer gasiti aici. Deci vom ramane în aceste exemple la rularea de wav-uri. Dacă rulam obiect.play() acel fisier va fi rulat. Obiect.source arata fisierul rulat și obiect.length arata durata fisierului.

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.

```
# Program kivy12
# Explica functiile kivy - audio widget
# Ion Studentul - 1/26/13

from kivy.core.audio import SoundLoader
from kivy.uix.gridlayout import GridLayout
from kivy.app import App
from kivy.uix.slider import Slider
from kivy.uix.switch import Switch
from kivy.uix.label import Label
muzica_activa = 0

class Switch_implementare(GridLayout):

    def __init__(self, **kwargs):
        super(Switch_implementare, self).__init__(**kwargs)
        self.cols = 1
        self.padding = 150
        self.switch1 = Switch(text="muzica")
        self.switch1.active =True
```

```
        self.add_widget(self.switch1)
        self.arata_volum = Label (text = "volum: 50")
        self.add_widget(self.arata_volum)
        self.slide_muzica = Slider(min=0, max=100, value=50)
        self.slide_muzica.step = 5
        self.slide_muzica.orientation="horizontal" # alte optiuni 'vertical'
        self.add_widget(self.slide_muzica)
        self.switch1.bind(active=self.dezactiveaza_volum)
        self.slide_muzica.bind(value=self.valoare_volum)
        self.sound = SoundLoader.load('JO_-_09_-_Fortitude.wav')
        self.sound.play()
        self.sound.loop = True
        self.sound.volume=0.5

    def valoare_volum (self,x,y):
        '''utilziat pentru a vedea volumul'''
        self.arata_volum.text = "volum: "+str(int(self.slide_muzica.value))
        self.sound.volume = self.slide_muzica.value/100

    def dezactiveaza_volum (self,x,y):
        '''utilziat pentru a acctiva sau a dezactiva slider-ul'''
        global muzica_activa
        if (muzica_activa %2 == 0) :
            self.slide_muzica.disabled =True
            self.sound.stop()
        else:
            self.slide_muzica.disabled =False
            self.slide_muzica.value = 0
            self.sound.play()
        muzica_activa += 1
        self.sound.volume=0

class AplicatiePersonalizata(App):

    def build(self):
        return Switch_implementare()

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

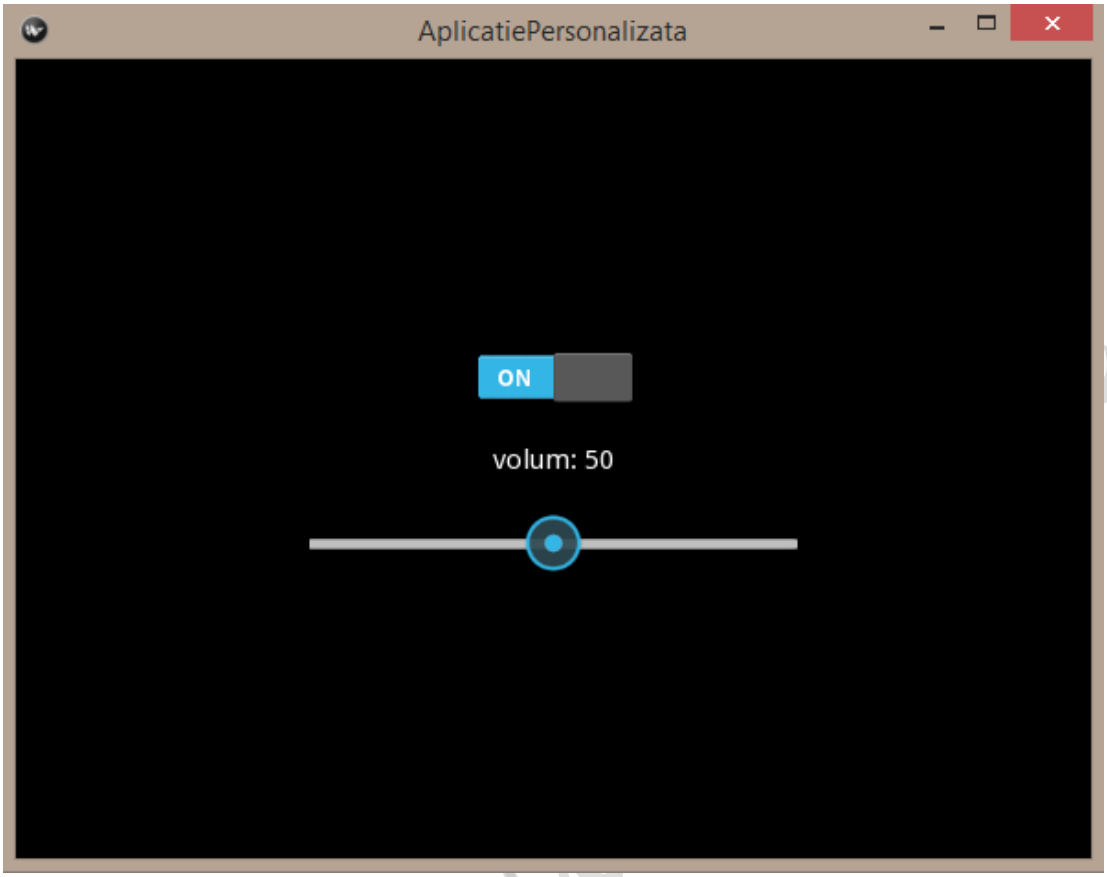


Fig.9

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 a sound:

```
def valoare_volum (self,x,y):
        '''utilziat pentru a vedea 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 retinem ca acest widget este dependent de codec-urile 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.

```
# Program kivy 8
# Explica functiile kivy -  video widget
# Ion Studentul - 1/26/13

import kivy

from kivy.app import App
from kivy.uix.videoplayer import VideoPlayer


class VideoPlayerApp(App):

    def build(self):
        filename = 'videoplayback.avi'
        return VideoPlayer(source=filename, state='play')


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

Fig.10

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

```
# Program kivy 9
# Explica functiile kivy -  video widget
# Ion Studentul - 1/26/13

import kivy

from kivy.app import App
from kivy.uix.videoplayer import VideoPlayer
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button

class VideoPlayerApp(App):

    def build(self):
        filename = 'videoplayback.avi'
        self.layout = GridLayout(cols=2)
        self.video =  VideoPlayer(source=filename, state='play')
        self.layout.add_widget(self.video)
        button1 = Button(text='Play video1', font_size=14)
        button1.bind(on_press=self.play_video1)
        button1.size_hint=(0.2,0.1)
        self.layout.add_widget(button1)
        button2 = Button(text='pauza video1', font_size=14)
        button2.size_hint=(0.2,0.1)
        button2.bind(on_press=self.pauza_video1)
        self.layout.add_widget(button2)
        button3 = Button(text='stop video1', font_size=14)
        button3.size_hint=(0.2,0.1)
        button3.bind(on_press=self.stop_video1)
        self.layout.add_widget(button3)
        button4 = Button(text='la 1/3 din film', font_size=14)
        button4.size_hint=(0.2,0.1)
        button4.bind(on_press=self.un_sfert_video1)
        self.layout.add_widget(button4)
        button5 = Button(text='la 2/3 din film', font_size=14)
        button5.size_hint=(0.2,0.1)
        button5.bind(on_press=self.doua_sferturi_video1)
        self.layout.add_widget(button5)
        button6 = Button(text='volum : -', font_size=14)
        button6.size_hint=(0.2,0.1)
        button6.bind(on_press=self.minus_video1)
        self.layout.add_widget(button6)
        button7 = Button(text='volum : +', font_size=14)
        button7.size_hint=(0.2,0.1)
        button7.bind(on_press=self.plus_video1)
        self.layout.add_widget(button7)
        return self.layout
```

```
    def play_video1 (self,buton):
        self.video.state='play'
    def pauza_video1 (self,buton):
        self.video.state='pause'
    def stop_video1 (self,buton):
        self.video.state='stop'
    def un_sfert_video1 (self,buton):
        self.video.seek(0.33)
    def doua_sferturi_video1 (self,buton):
        self.video.seek(0.66)
    def minus_video1 (self,buton):
        if (self.video.volume>0):
            self.video.volume -=0.1
    def plus_video1 (self,buton):
        if (self.video.volume<1):
            self.video.volume +=0.1


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

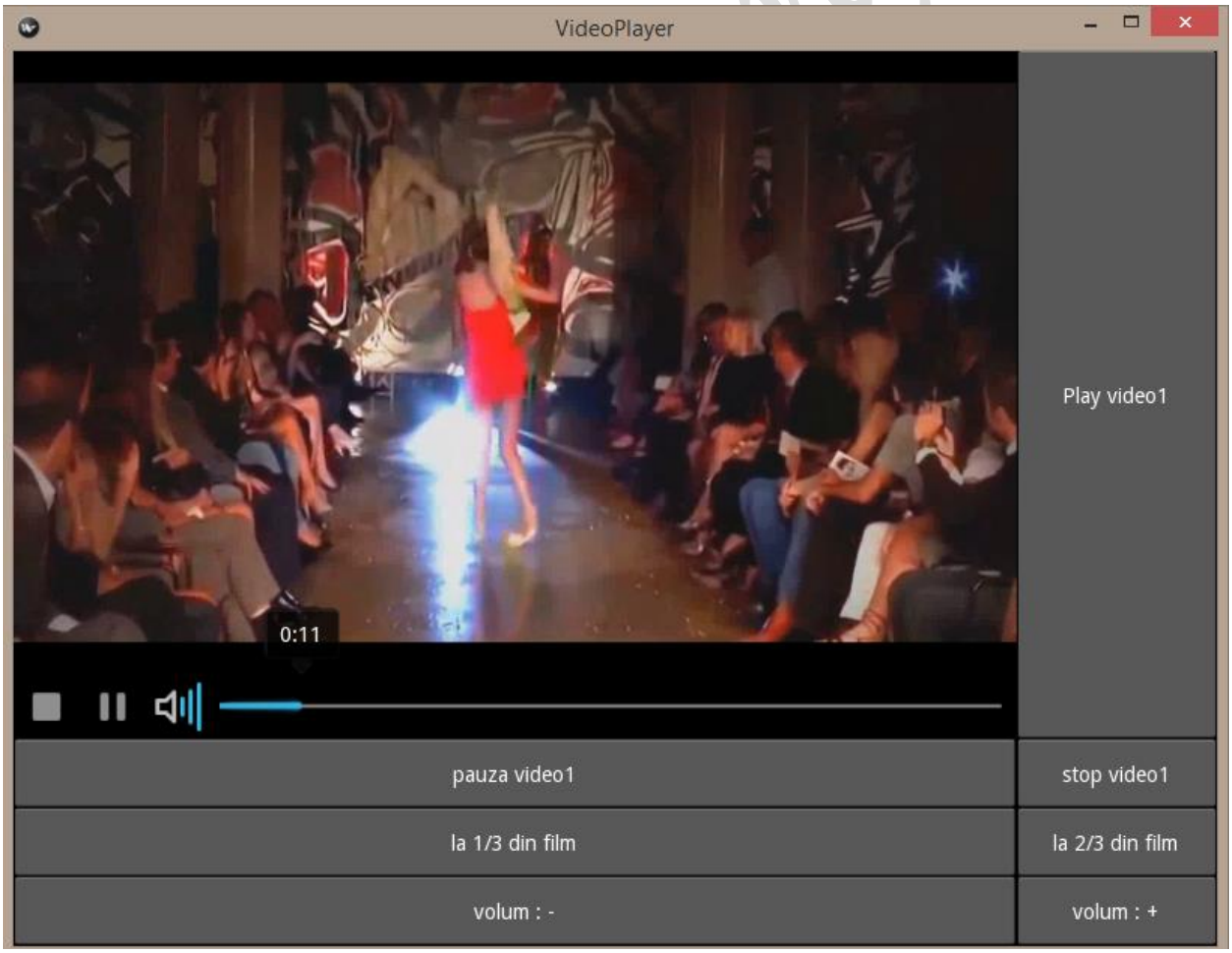


Fig.11

In programul de mai sus incercam să afisam comportamentul fiecarei proprietati pe care un widget de tip video poate să o prezinte.

Prin urmare putem vedea sub metoda build definirea unui fisier de tip avi ce o să fie folosit pentru rulare și crearea unui layout pe doua coloane de tip grid numit self.layout.

```
filename = 'videoplayback.avi'
self.layout = GridLayout(cols=2)
```

De asemenea vom crea un widget video ce va deveni copilul layout-ului self.layout. Acesta are starea play prin urmare fisierul viode va incepe să fie redat.

```
self.video =  VideoPlayer(source=filename, state='play')
self.layout.add_widget(self.video)
```

In urmatoarea sectiune vom atasa sapte butoane, ficare avand legata o functie la apasare pentru a demonstra fiecare prorpeitate a widget-ului video. Fiecare buton va face parte din layout-ul self.layout. Fiecare buton are o functie care schimba caracteristici ale redarii video prin modificarea de parametrii a widget-ului self.video conform cu numele butonului.

```
button1 = Button(text='Play video1', font_size=14)
button1.bind(on_press=self.play_video1)
button1.size_hint=(0.2,0.1)
self.layout.add_widget(button1)
button2 = Button(text='pauza video1', font_size=14)
button2.size_hint=(0.2,0.1)
button2.bind(on_press=self.pauza_video1)
self.layout.add_widget(button2)
button3 = Button(text='stop video1', font_size=14)
button3.size_hint=(0.2,0.1)
button3.bind(on_press=self.stop_video1)
self.layout.add_widget(button3)
button4 = Button(text='la 1/3 din film', font_size=14)
button4.size_hint=(0.2,0.1)
button4.bind(on_press=self.un_sfert_video1)
self.layout.add_widget(button4)
button5 = Button(text='la 2/3 din film', font_size=14)
button5.size_hint=(0.2,0.1)
button5.bind(on_press=self.doua_sferturi_video1)
self.layout.add_widget(button5)
button6 = Button(text='volum : -', font_size=14)
button6.size_hint=(0.2,0.1)
button6.bind(on_press=self.minus_video1)
self.layout.add_widget(button6)
button7 = Button(text='volum : +', font_size=14)
button7.size_hint=(0.2,0.1)
button7.bind(on_press=self.plus_video1)
self.layout.add_widget(button7)
```

Apoi cream aceste metode. Fiecare metoda este discutata individual. Metoda play_video1 schimba starea self.video.state în „play” pt a incepe redearea filmului.

```
    def play_video1 (self,buton):
        self.video.state='play'
```

Metoda pauza_video1 schimba starea self.video.state în „pause” pt a intrerupe redearea filmului.

```
    def pauza_video1 (self,buton):
        self.video.state='pause'
```

Metoda play_video1 schimba starea self.video.state în „stop” pt a opri redearea filmului.

```
    def stop_video1 (self,buton):
        self.video.state='stop'
```

Metoda un_sfert_video1 schimba proprietatea seek a self.video la 0.33 și astfel filmul va avea un salt la 1/3 din film.

```
    def un_sfert_video1 (self,buton):
        self.video.seek(0.33)
```

Metoda doua_sferturi_video1 schimba proprietatea seek a self.video la 0.66 și astfel filmul va avea un salt la 2/3 din film.

```
    def doua_sferturi_video1 (self,buton):
        self.video.seek(0.66)
```

Metoda minus_video1 schimba proprietatea volume a self.video decrementand cu 0.1 dacă valoarea volumului este mai mare decat zero. Filmul isi va diminua volumul la fiecare apasare de buton.

```
    def minus_video1 (self,buton):
        if (self.video.volume>0):
            self.video.volume -=0.1
```

Metoda plus_video1 schimba proprietatea volume a self.video incrementand cu 0.1 dacă valoarea volumului este mai mica decat unu. Filmul isi va amplifica volumul la fiecare apasare de buton.

```
    def plus_video1 (self,buton):
        if (self.video.volume<1):
            self.video.volume +=0.1
```

Urmatorul program face trecerea la formarea unui meniu.

```
# Program kivy 10
# Explica functiile kivy -  video widget
# Ion Studentul - 1/26/13
```

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.videoplayer import VideoPlayer
from kivy.uix.gridlayout import GridLayout
from kivy.uix.floatlayout import FloatLayout


class MyApp(App):
    def build(self):
        self.parent= FloatLayout()
        self.Menu(None)
        return self.parent

    def Menu(self, buton):
        self.parent.clear_widgets()
        try:
             self.video1.state="stop"
        except:
            pass
        try:
            self.video2.state="stop"
        except:
            pass
        self.layout1 =GridLayout(cols=2)
        self.layout1.spacing =100
        self.layout1.padding =100
        button1 = Button(text='Play video1', font_size=14)
        button1.bind(on_press=self.on_button_press1)
        self.layout1.add_widget(button1)
        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)
        return self.parent

    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

    def on_button_press2(self,buton):
        self.parent.clear_widgets()
        self.layout3 =GridLayout(cols=2)
        menuBack = Button(text='Menu', font_size=14)
        menuBack.size_hint = (0.1,0.1)
        menuBack.bind(on_press=self.Menu)
```

```
        self.layout3.add_widget(menuBack)
        self.video2= VideoPlayer(source='videoplayback.avi', state='play')
        self.layout3.add_widget(self.video2)
        self.parent.add_widget(self.layout3)
        return self.parent

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

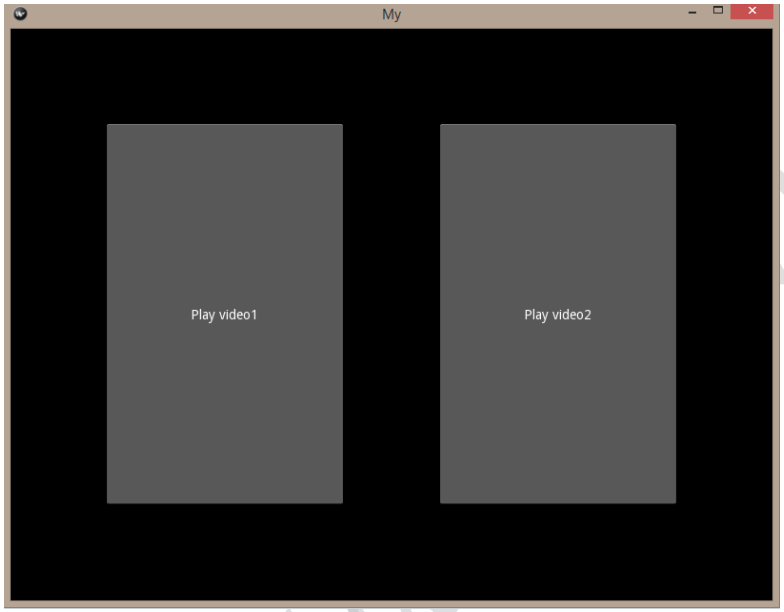


Fig.12

Fig.13

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).

Sa vorbim despre metoda build. Pentru prima data 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. Vom explica doar una din metode. 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
```

Sectiunea ce nu trebuie uitata este return self.parent pentru a updata schimbarile facute.

Widget-uri avansate

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 dechidem un popup trebuie să apelam open(); dacă dorim să inchidem un Popup trebuie să apelam dismiss().

```
# Program kivy13
# Explica functiile kivy popup widget
# Ion Studentul - 1/26/13
from kivy.uix.gridlayout import GridLayout
from kivy.app import App
from kivy.uix.popup import Popup
from kivy.uix.button import Button
from kivy.uix.label import Label

class PopupApp(GridLayout):

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

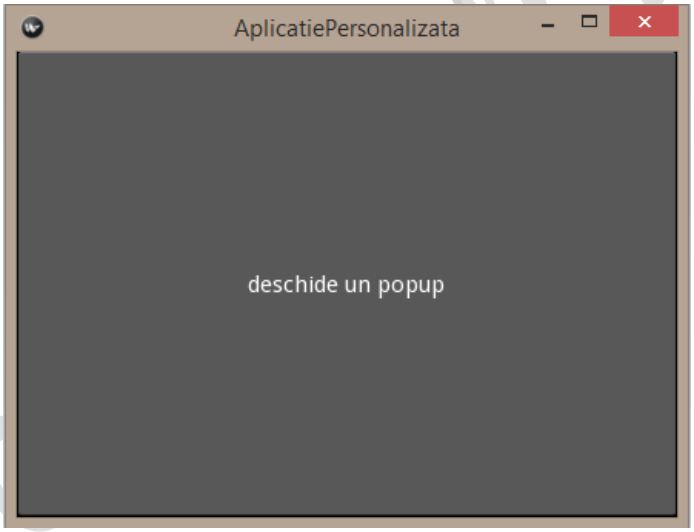
```
    def un_popup(self,buton):
        """Creaza un popup"""
        inchide=Button(text='Inchide! ')
        self.popup = Popup()
        self.popup.title='Testam un popup'
        self.popup.content=inchide
        self.popup.open()
        inchide.bind(on_press=self.inchide_popup)

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

class AplicatiePersonalizata(App):

    def build(self):
        return PopupApp()

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



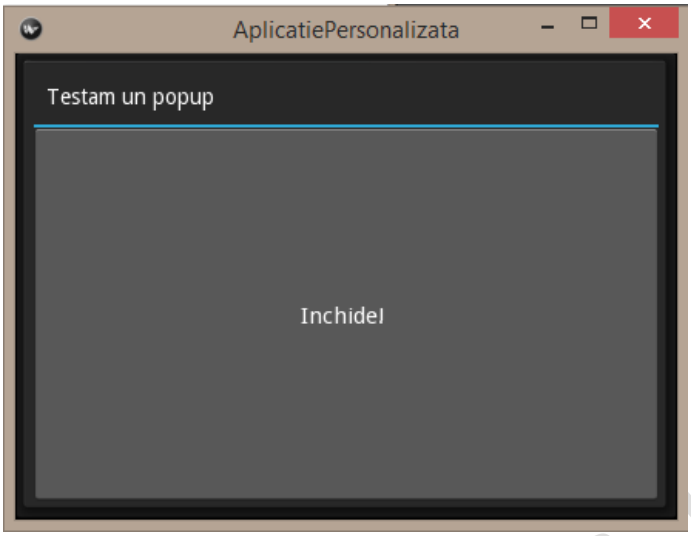


Fig.15

In cele ce urmeaza explicam ce contine programul de mai sus. 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. Acest program se regaseste mai jos.

```
# Program kivy13
# Explica functiile kivy popup widget
# Ion Studentul - 1/26/13

from kivy.uix.gridlayout import GridLayout
from kivy.app import App
from kivy.uix.popup import Popup
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout

class PopupApp(GridLayout):

    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)

    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!")
        layout2=BoxLayout()
        layout2.orientation = "vertical"
        layout2.add_widget(eticheta)
        layout2.add_widget(inchide)
        layout2.padding = 40
        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)

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

class AplicatiePersonalizata(App):

    def build(self):
        return PopupApp()

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

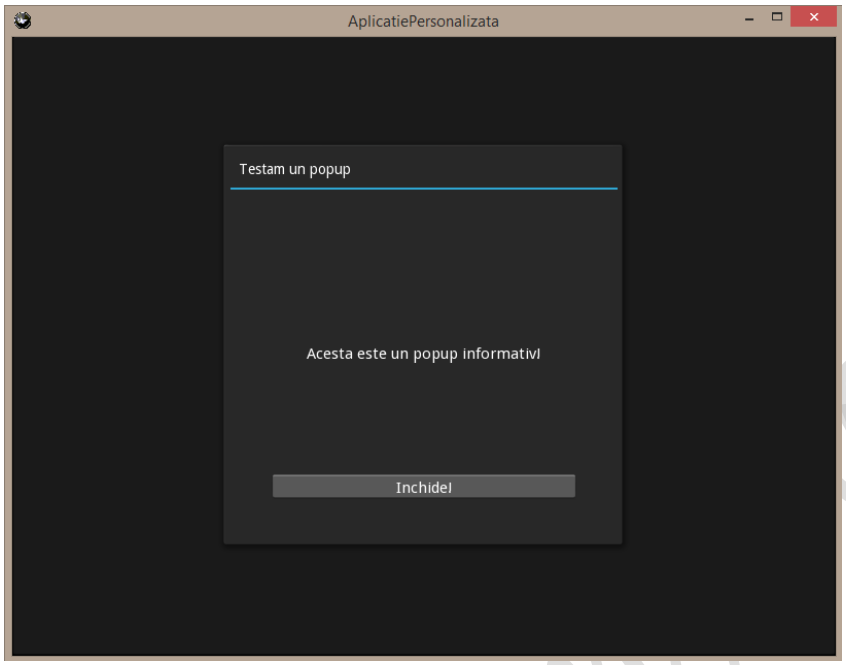


Fig.16

In init regasim un buton ce va ocupa doar 0.8 din suprafata layout-ului. Acest buton afiseaza textul “deschide un popup” și are legat un eveniment de apasarea 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)
```

In cadrul metodei un_popup regasim 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 afisa text-ul "Acesta este un popup informative”.

```
    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 copiii label-ul eticheta și buton-ul inchide. Vom adauga și un padding la layout-ului de 40 pentru aspect viziual.

```
        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().

In ultima linie din aceasta metoda adaugam un eveniment la apasarea 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)
```

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

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

Un click/tap în exteriorul widget-ului popup va generea inchiderea widget-ului popup similar cu apasarea butonului inchide.

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.

```
# Program kivy
# Explica functiile kivy -  scatter widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.scatter import Scatter
from kivy.uix.gridlayout import GridLayout
from kivy.uix.image import Image

class Scatter_test(GridLayout):

    def __init__(self, **kwargs):
        super(Scatter_test, self).__init__(**kwargs)
        self.cols = 1
```

```
        self.scatter1 = Scatter()
        self.imag1 = Image(source="nature1.jpg")
        self.scatter1.add_widget(self.imag1)
        self.add_widget(self.scatter1)

class AplicatiePersonalizata(App):

    def build(self):
        return Scatter_test(size=(400, 400), size_hint=(None, None))


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

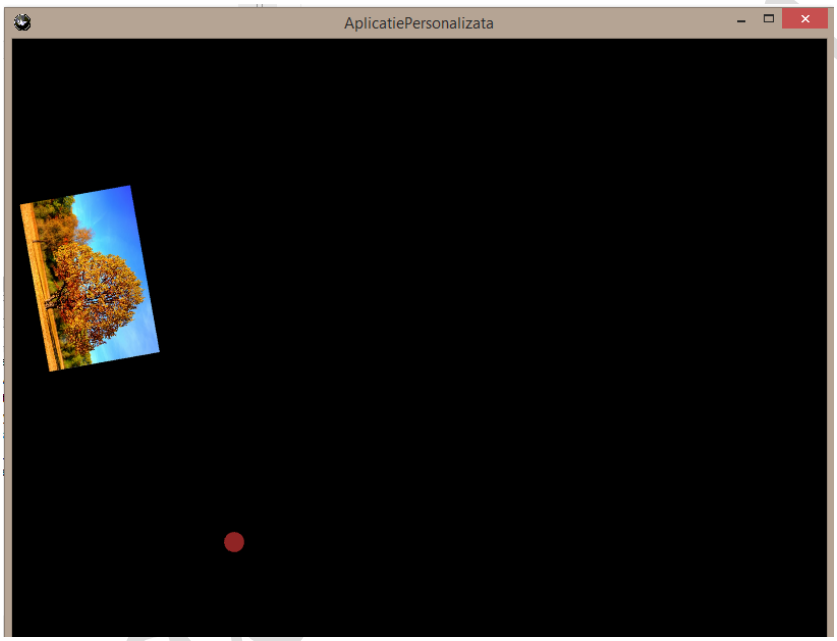


Fig.17

Daca dam click dr. intr-un loc putem să rotim sau scalam imaginea.Cand terminam pentru a elimina acel punct rosu facut cu click dreapta și pentru a ne opri din scalat/rotit atunci dam click stanga pe acel punct rosu. De asemenea, vedem ca dacă dam click stanga pe imagine aceasta se transpune în aceasi pozitie ca la pornirea programului.

Codul integreaza în metoda init crearea unui scatter ce are ca și child un widget de tip imagine.

Vom incerca să upgradam un pic acest widget.Prin urmare ca imaginea să nu se mai transpuna în aceasi pozitie ca la inceput cand 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 elimiat și adaugat de fiecare data la parinte cand este atins (sau click stanga). Este util cand manipulezi multiple scatter și nu doresti ca cel activ să fie partial ascuns.

Pentru a scala acest widget folosim proprietatea scale. Aceasta scaleaza dimensiunea obiectelor din acest widget automat. A nu se folosi size, ci scale.

Am creat un program care foloseste fiecare proprietate explicata mai sus.

```
# Program kivy1
# Explica functiile kivy -  image widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.scatter import Scatter
from kivy.uix.gridlayout import GridLayout
from kivy.uix.image import Image
from kivy.uix.label import Label
from kivy.uix.button import Button

class Scatter_test(GridLayout):

    def __init__(self, **kwargs):
        super(Scatter_test, self).__init__(**kwargs)
        self.cols = 1
        self.layout1 = GridLayout(cols=1)
        self.eticheta = Label (text = "unghi: 0")
        self.imag1 = Image(source="nature3.jpg")
        self.buton1=Button(text="Afla rotatie")
        self.buton1.size_hint = (1,0.05)
        self.layout1.add_widget(self.eticheta)
        self.layout1.add_widget(self.imag1)
        self.scatter1 = Scatter()
        self.scatter1.auto_bring_to_front = False
        self.scatter1.add_widget(self.layout1)
        self.add_widget(self.scatter1)
        self.add_widget(self.buton1)
        self.buton1.bind(on_press = self.roteste_ma)

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

class AplicatiePersonalizata(App):

    def build(self):
        return Scatter_test()

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

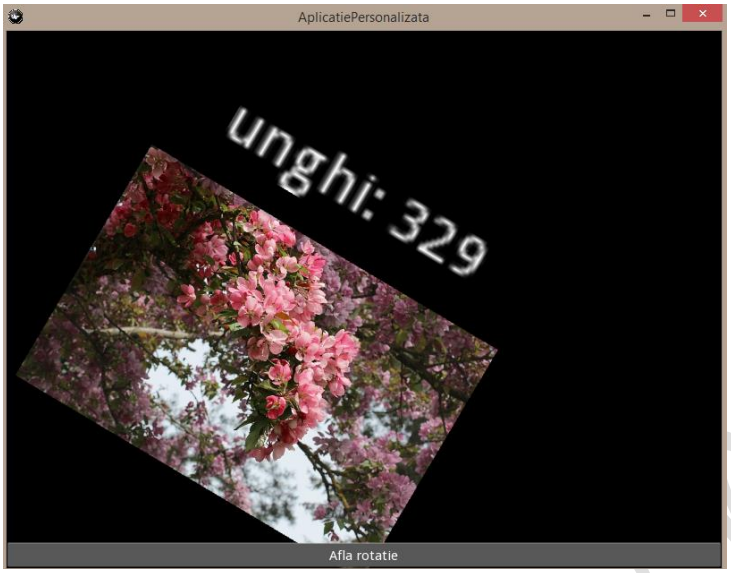


Fig.18

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)
```

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. Setam proprietatea auto_bring_to_front la False.Adaugam ca și copil al scatter-ului layout1.

```
self.scatter1 = Scatter()
self.scatter1.auto_bring_to_front = False
self.scatter1.add_widget(self.layout1)
```

Adaugam self.scatter1 și self.buton1 ca și copiii 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)
```

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-ul TabbedPanel administreaza diferite widget-uri în tab-uri. El este contruit din header area pentru butonul de tab actual și un content area ce contine un singur widget. Dacă dorim să punem mai multe widget-uri atunci trebuie să cream un layout. Inainte de a implementa trebuie să intelegem ce este un TabbedPanelHeader.
Un tab individual este numit TabbedPanelHeader. Acesta este un buton special ce contine un content. Pentru a foosi acest TabbedPanelHeader trebuie sa il adaugam la TabbedPanel.

```
tp = TabbedPanel()
th = TabbedPanelHeader(text='Tab2')
th.content = un_alt_widget
tp.add_widget(th)
```

Este important să retinem ca primul tab este creat automat și adaugat la instantiere și se numeste **Default tab.**

Iata și un exemplu simplu.

```
# Program kivy
# Explica functiile kivy -  TabbedPanel widget
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.uix.tabbedpanel import TabbedPanelHeader
from kivy.uix.gridlayout import GridLayout
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.image import Image

class TabbedPanelApp(GridLayout):
    def __init__(self, **kwargs):
        super(TabbedPanelApp, self).__init__(**kwargs)
```

```
        self.cols =1
        tb_panel= TabbedPanel()

        th_text_head = TabbedPanelHeader(text='Text tab')
        th_text_head.content= Label(text='Un text')

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

        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))

        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)

class AplicatiePersonalizata(App):

    def build(self):
        return TabbedPanelApp()

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

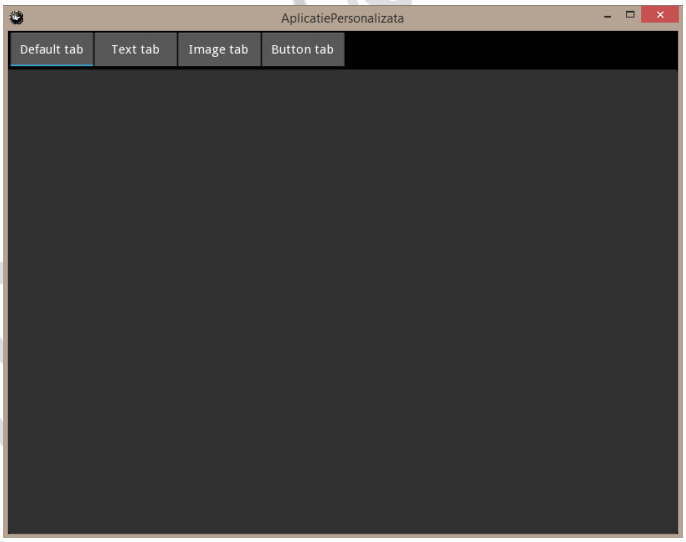


Fig.19

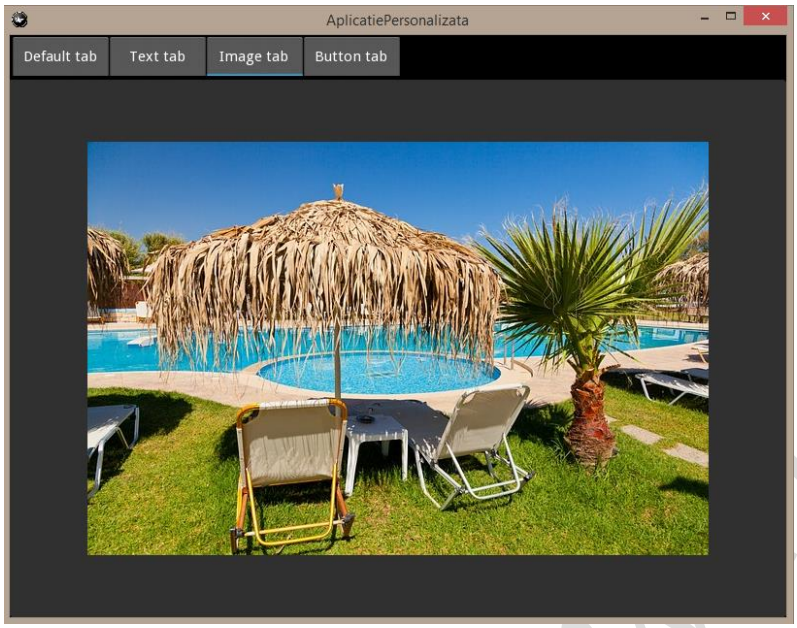


Fig.20

In metoda init regasim crearea unui obiect de tip TabbedPanel

```
tb_panel= TabbedPanel()
```

Cream apoi un tabbed header numit th_text_head. Acesta va avea ca și continut (content) un label ce afiseaza mesajul “Un text”.

```
        th_text_head = TabbedPanelHeader(text='Text tab')
        th_text_head.content= Label(text='Un text')
```

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))
```

Pasul urmator este să adaugam fiecare tabbed header la tabbed panel. Ulterior tabbed panel va fi adaugal 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)
```

Mai jos gasim un exemplu imbunatatit de tabbed panel.

```
from kivy.app import App
from kivy.uix.tabbedpanel import TabbedPanel
from kivy.uix.tabbedpanel import TabbedPanelHeader
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.image import Image
from kivy.uix.gridlayout import GridLayout

class TabbedPanelApp(App):
      def build(self):
          tb_panel= TabbedPanel()
          tb_panel.background_image ="fundal2.jpg"
          tb_panel.default_tab_text =  "tab-ul default"
          tb_panel.default_tab_content = Image(source='infoacademy3.gif',pos=(200,
100), size=(200, 200))

          th_text_head = TabbedPanelHeader(text='Text tab')
          th_text_head.content= Label(text='Infoacademy', font_size = 40)

          th_img_head= TabbedPanelHeader(text='Image tab')
          th_img_head.content= Image(source='infoacademy4.gif',pos=(400, 100),
size=(400, 400))

          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


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

          return tb_panel

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

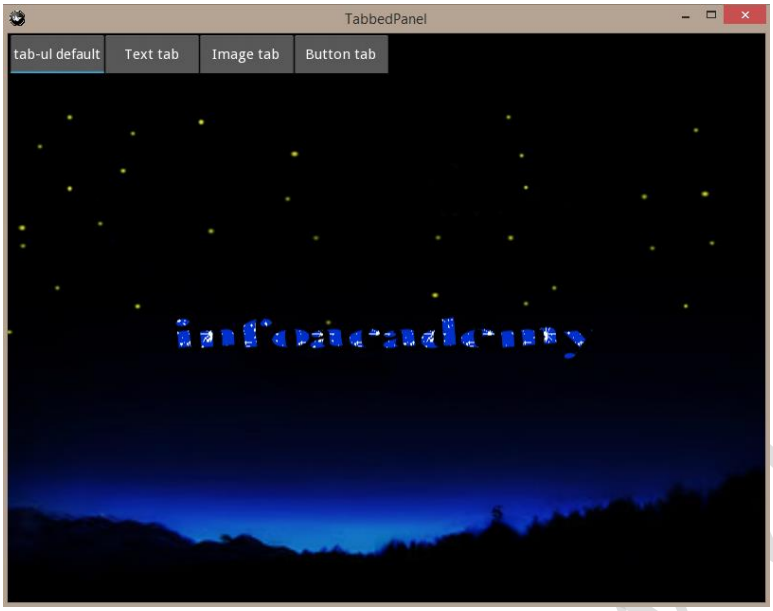


Fig. 21

Imbunatatirea consta în setarea unui fundal 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. 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
```

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. Valoarea standard a atributului anim_move_duration este 0.5 secunde. Chiar dacă nu este utilizată aici proprieatatea loop a obiectului de tip carousel va permite ca imaginile să fie glisate chiar dacă lista se termina, reluand lista de obiecte afisate.

```
from kivy.app import App
from kivy.uix.carousel import Carousel
from kivy.uix.image import Image
from kivy.uix.label import Label

class Example1(App):

    def build(self):
        #define the carousel
        carousel = Carousel(direction='right')
        carousel.anim_move_duration = 1
        carousel.padding = 40
        image1 = Image(source="nature1.jpg")
        carousel.add_widget(image1)
        image2 = Image(source="nature2.jpg")
        carousel.add_widget(image2)
        image3 = Image(source="nature3.jpg")
        carousel.add_widget(image3)
        image4 = Image(source="nature4.jpg")
        carousel.add_widget(image4)
        eticheta =  Label (text = "Am ajuns la finalul listei!", font_size = 40)
        carousel.add_widget(eticheta)

        return carousel

if __name__ == '__main__':
    Example1().run()
```

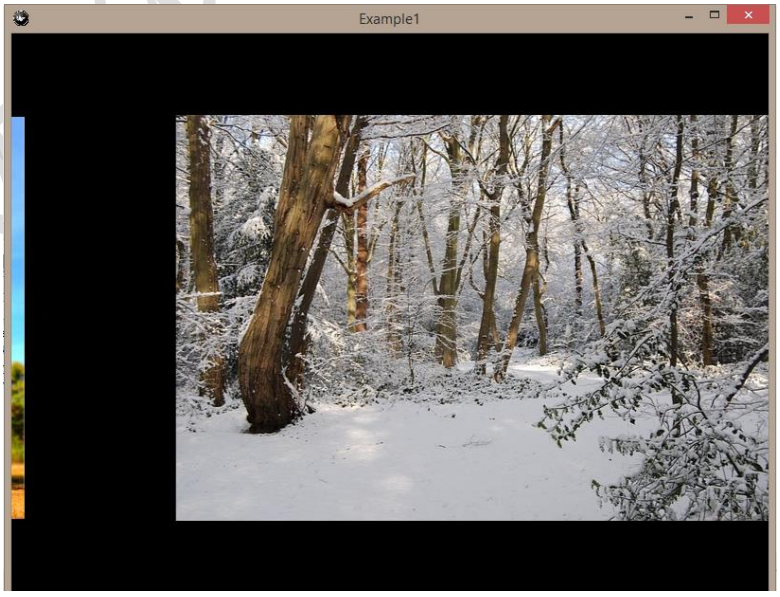


Fig.22

Limbajul Kivy

Pe masura ce aplicatia creste și devine mai complexa este o practica comuna ca contructia copacului cu declaratii și proprietati să devina mai greu de mentinut. Limbajul Kivy permite să elimine acest inpediment. 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.

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 intr-un anumit fel. Poti utiliza regulile pt. a specifica proprietati (cum ar fi size, size_hint etc.) sau comportament al widget-urilor (adugare de copiii sau bindings). De asemenea, poti utiliza limbajul kivy să cream intreaga 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.

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..

# Sintaxa pentru crearea unei dynamic class
<NewWidget@BaseClass>:
    # .. definitii..
```

Iata cum ar trebui să arate codul kivy:

```
<ClassName>:
    prop1: value1
    prop2: value2

    canvas:
        CanvasInstruction1:
            canvasprop1: value1
        CanvasInstruction2:
            canvasprop2: value2
```

Daca primul cuvand care este neindentat este inconjurat de “<>” atunci este regula, în caz contrar este root widget. Expresia “canvas:” este regasita peste tot deoarece fiecare widget are propriul canvas.

Alte observatii:

- Indentarea este importanta și trebuie să fie consistenta. Nu se poate amesteca spatii cu tab-uri (caractersitica python)
- Valoarea proprieatii trebuie data o singura data

Alte sintaxe speciale:

- Importarea unui modul:
  #:import nume x.y.z
  Este echivalent cu urmatoarea sintaxă din python:
  from x.y import z as nume
- Setarea unei variabile globale:
  #:set nume valoare
  Este echivalent cu urmatoarea sintaxă din python:
  name = value

Sa vedem cum arata un exemplu kivy:

```
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))
```

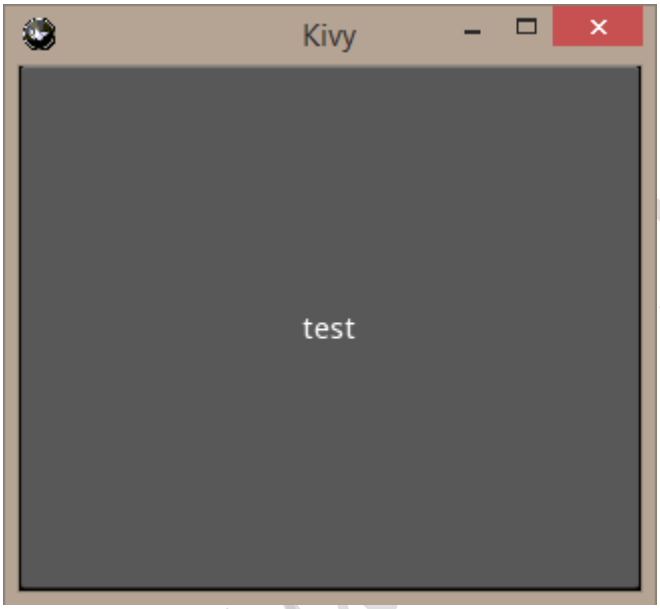


Fig.23

Mai multe detalii gasiti dand click aici.

Transformarea unei aplicatii în executabil

Primul lucru notabil este ca puteti intampina probleme. Nu ar trebui să va faceti griji dacă se intampla să intamplinati probleme,dar compilarea pentru android implica multe lucruri în spate care pot ocazional să nu functioneze asa cum ne-am dori. Dacă intampinati probleme, ar trebui să stiti ca Kivy are o comunitate foarte activa și foarte buna în a va ajuta sau rezolva aceste probleme. În particular puteti să adresati intrebarile voastre sau să reportati problema în link-urile gasite în sectiunea support.

De asemenea as dori să specific ca putem utiliza și pentru windows un bulider, dar subliniez faptul ca pot aparea situatii neprevazute și aici. Va rog sa cititi mai multe detalii aici.

Pentru toate celelalte sisteme de operare putem instala Python. Pe linux Python vine default pe mai multe distributii , iar kivy se instaleaza foarte usor pe linux. . Deci intrebarea este cum instalam un .py pe android. Una din metode este instalarea

aplicatiei Kivy Launcher , dar aceasta nu ofera portabilitate. Prin urmare e mai bine sa cream un fisier apk.

Pentru procesul de crearea a unui apk exista doua metode de a crea aplicatia kivy . Ambele utilizeaza proiectul python-for-android. Prima metoda realizeaza conversia prin apelarea scripturilor acestui proiect în mod direct printr-un mod de linie de comanda. Poti gasi intructiuni pentru aceasta medoda pe site-ul kivy aici. Aceasta metoda este buna, dar e mult mai usor să utilizam o metoda automata (metoda buildozer). Metoda buildozer ruleaza in spate tot python-for-android, dar adauga multiple fisiere de configurare penutra seta aplicatia. De asemenea adauga automat multiple dependente. Prin urmare in cele ce urmeaza vom discuta despre buildozer.

Inainte de toate trebuie sa retinem ca buildozer ruleaza doar pe Linux si OS X. Nu ruleaza pe windows. De obicei instalam o masina virtuala Linux,in general in Oracle VirtualBox. In al doilea rand, kivy ruleaza un online cloud builder ce utilizeaza python-for-android pentru a compila codul upload-at (copiat pe server) cu putini parametrii. Este mult mai flexibil si mai sigur ca functioneaza decat tool-uri locale. Iata si pagina pentru online cloud builder (click aici). Recomand varianta online datorita simplitatii.

In cazul in care nu doriti varianta online puteti merge pe varianta buildozer.

Acesta opertiune se face cu ajutorul aplicatiei pip.

Dupa ce ati instalalt linux (exemplificam pe ubuntu) trebuie sa scrieti intr-o fereastra terminal (fara semnul $) :

$ **sudo apt-get install python-pip**

Pentru a instala buildozer trebuie sa scrieti intr-o fereastra terminal (fara semnul $) :

$ **sudo pip install buildozer**

O observatie importanta este ca trebuie sa avem instalat kivy pe linux, dar si alte module ce le-am folosit in aplicatie.

Apoi trebuie sa mergem in directorul care are salvata aplicatia kivy ce o dorim sa o convertim. Trebuie sa modificam numele aplicatiei in main.py si asta datorita faptului ca aplicatia android cauta si ruleaza aplicatii main. Aceasta va fi punctul de intrare in aplicatie. Toate celelalte fisiere pot ramane cu numele neschimbat.

Urmatorul pas este sa cream un fisier buildozer.spec ce este un fisier de configurare care contine toti parametrii utili pentru conversie. Poti crea fisierul utilizand tot aplicatia buildozer cu ajutorul comenzii:

buildozer init

Aceasta comanda creaza un fisier numit buildozer.spec in directorul curent populat cu valori standard.

Populatea fisierului buildozer.spec

In cele ce urmeaza vom discuta de cele mai uzuale valori ce pot fi setate in fisierul buildozer.spec. Trebuie sa sti aceste valori in cazul in care conversia va da eroare.

- title: Numele aplicatiei tale, eu am folosit „Infoacademy.net”
- package.name: Un șir de caractere simplu fara pauza care impreuna cu package.domain trebuie sa formeze un identificator unic
- package.domain: Nu trebuie un domeniu real. Standard este org.test dar e bine sa treceti propriul domeniu.
- source.dir: Directorul ce contine fisierele sursa, incluzand fisierul main.py file. Standard este ‘.’ si este bine deoarece inseamna directorul curent.
- source.include_exts: Buildozer va include automat fisierele sursa cu extensiile respective in APK-ul tau. Multiple fisiere sunt standard incarcate.
- source.exclude_exts, source.exclude_dirs, source.exclude_patterns: Mai multe optiuni pentru a exclude anumite fisiere de a fi integrate in APK.
- version.regex, version.filename: TAceasta este metoda default ce cauta declararea versiunii. Buildozer cauta in fisierul main dupa un șir de caractere de formă __version__ = 'some_version'. Eu nu adaug acest șir si merg pe varianta a doua prin setarea manuala a versiunii comentand aceste linii.
- version: setarea manuala a versiunii. Eu am setat 1.0 pt. moment.
- requirements: este o lista separata de virgula cu modulele ce trebuie importate. Dacă nu ai module speciale trebuie sa treci doar „kivy”
- presplash.filename: Fisierul ce indica o imagine care va fi utilizată in kivy loading screen la inceputul rularii. Dacă nu setezi nimic va adauga un kivy loading screen.
- icon.filename: Un fisier ce indica o imagine ce poate fi folosita ca icon. Este comentata standard, deci va utiliza kivy icon. Se cere un PNG.
- orientation: Orientarea aplicatiei tale Poti alege dintre ‘landscape’, ‘portrait’, sau ‘all’. All inseama ca aplicatia ta se poate roti.
- fullscreen: Dacă este setat la 1 va incerca sa umple cat mai mult posibil.

Mai sunt si alti parametrii, dar care nu sunt obligatorii deoarece au un default.

Pentru a crea fisierul apk rulati din consola :

$ buildozer android debug

Mai multe detalii gasiti aici.

Suport Kivy

Exista multiple pagini ce ofera support pentru kivy, discutii de tip “how to” sau sugestii pentru a scrie kivy. Iata cateva pe catre 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/