

SEDINTA 7 – KIVY

Intro

Modulul Kivy este un modul ce poate fi utilizat pentru crearea de interfețe grafice Cross Platform. Acest lucru reprezintă ca același cod poate fi rulat pe sisteme diferite de operare precum Linux, Windows, OS X, Android sau iOS.

Totusi de ce sa-l folosim? Exista multiple toolkits ce au ca scop crearea de aplicatii grafice cum ar fi QT sau Flash?

In primul rand kivy este conceput pentru azi și pentru maine.

Kivy poate administra metode Multi-Touch, pe langa experienta metodei clasice a unui mouse.

Kivy este foarte flexibil. Poate fi rulat pe sisteme diferite de operare precum Linux, Windows, OS X, Android sau iOS. De asemenea este foarte scalabil pentru a putea de veni foarte usor de adaptat la tehnologii noi sau solutii terte.

Kivy este concentrat în sensul în care putem scrie aplicatii în cateva linii de cod.

Kivy este finantat. Acest lucru face ca kivy să fie creat de developer foarte competenti și profesionali. Kivy nu este un proiect trecator, nu este un proiect mic studentesc.

Kivy este gratuit. Nu trebuie să platesti pentru kivy, chiar dacă tu faci bani de pe urma vanzarilor de aplicatii kivy. Totusi poti dona sau poti participa la imbunatatirea produsului.

Intrebarea este de ce să folosim Python ca baza in aplicatiilor noastre? Nu este mai incet decat alte variante? Vom incerca să oferim un raspuns cat se poate de aprofundat. Python este un limbaj de programare care permite să realizezi multe lucruri intr-un timp cat mai scurt (prin comparatie cu alte limbaje). Pentru multe scenarii de dezvoltare a aplicatiilor este mult mai recomandat să scriem aplicatia repede intr-un limbaj de nivel inalt precum Python, apoi sa-l optimizam. În ceea ce priveste viteza de executie a unei implementari a aceluasi algoritm în limbaje diferite, cum ar fi C++ și Python poti observa ca Python este mai lent decat C++. Desenarea de grafice sofisticate duce la un proces destul de costisitor în ceea ce priveste resursele. Dar în aproape toate cazurile aplicatia ta va ajunge să ruleze de cele mai multe ori aceasi parte de cod. În kivy, spre exemplu, aceste parti sunt dispecerizarea de evenimente și desenarea de grafice. Acum python permite să realizam aceste parti mult mai rapid. Prin utilizarea Cython, poti compila codul tau la nivelul limbajului C și de acolo putem utiliza optimizari uzuale ale compilatorului pentru a mari viteza de executie oferind o crestere a factorului de performanta de la 1x la 1000x (depinde foarte mult de cod). Compilarea de cod la nivel de limbaj C se realizeaza oriunde eficienta este cu adevarat

critica. Pentru desenarea de grafice Kivy se bazeaza pe GPU pt a maximiza performanta.

Kivy nu a fost creat să reinventeze roata, totusi se doreste a fi inovativ pe piata limbajelor de programare. Prin urmare, concentrarea se face pe propriul cod, folosindu-se cod existent dacă librariile sunt de calitate și stabile. Pt. a suporta un set de optiuni bogate multe librarii sunt necesare. Prin urmare, iata o lista de dependente:

- [Cython](#).

Alte module optionale sunt:

- [OpenCV 2.0](#) – Camera de luat vederi.
- [PIL](#) – Afisarea de imagine și text.
- [PyCairo](#) – Afisare de text.
- [PyEnchant](#) – Corectura de text ortografic.
- [PyGST](#) – Redare audio/video și camera de luat vederi
- [Pygame](#) pt. redare audio/video

Acestea fiind spuse, trebuie să trecem la treaba!

Instalarea Kivy

Totusi instalarea kivy nu este asa de dificil pe cat ne-am asteptat deoarece dezvoltatorii de la kivy pe ofera o varianta portabila de Python pentru Windows și MacOS X cu toate modulele instalate pe care putem să o utilizam pentru dezvoltarea de aplicatii kivy. Aceasta modalitate este cea mai usoara pentru a rula Kivy fara a fi obligat să instalezi altceva în systemul de operare. Metoda de instalare este simpla deoarece cuprinde și un compiler Python și cu toate librariile necesare. Dacă doresti să instalezi Kivy într-un mediu unde exista deja Python instalat, este cu putinta și acest lucru, dar aceasta practica nu aduce imbunatatiri considerabile.

Instalarea dintr-o versiune portabila:

1. Copiaza ultima versiune de kivy accesand: <http://kivy.org/#download>





Operating System	File	Instructions	Size
 Windows 7, 8 (32/64 bit)	Kivy-1.8.0-py2.7-win32.zip (Mirror)	Installation for Windows	126MB
	Kivy-1.8.0-py3.3-win32.zip (Mirror)		129MB
 Mac OS X 10.7, 10.8, 10.9 (requires Python 2.7)	Kivy-1.8.0-osx.dmg (Mirror)	Installation for MacOSX	34 Mb
 Linux (tested Ubuntu > 13.04, 32/64 bit, Maegia, Arch)	Kivy-1.8.0.tar.gz	Installation for Ubuntu	13 Mb
 Ubuntu PPA	Stable PPA Daily PPA	How to use software from PPA	12 Mb

Fig. 1

2. Extrage fisierul copiat.

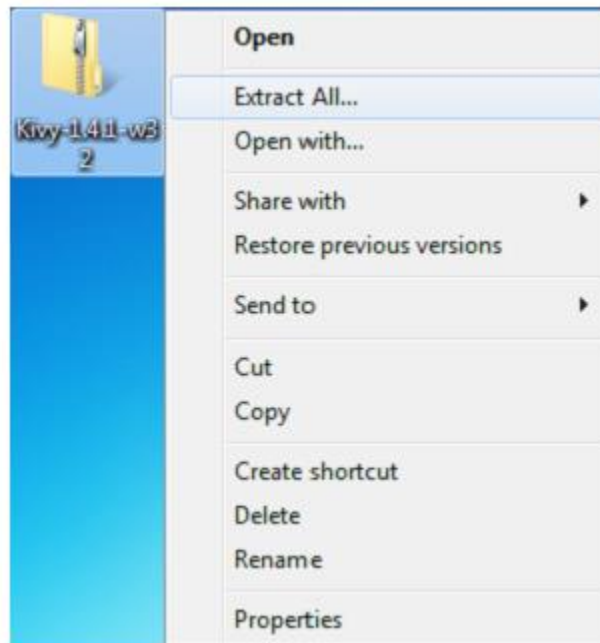


Fig. 2

3. In directorul unde ai dezarhivat fisierul zip copiat local la pasul 2 ai un fisier numit kivy.bat. Utilizeaza acest fisier pt. a lansa orice aplicatie kivy cum e descris mai jos.

In momentul de fata instalarea este realizata, Totusi pentru a putea rula aplicatii kivy avem doua posibilitati. Una este să utilizam metoda send-to, iar cealalta este să utilizam metoda duple-click.

Metoda send-to:

1. Copiaza fisierul kivy.bat în clipboard.

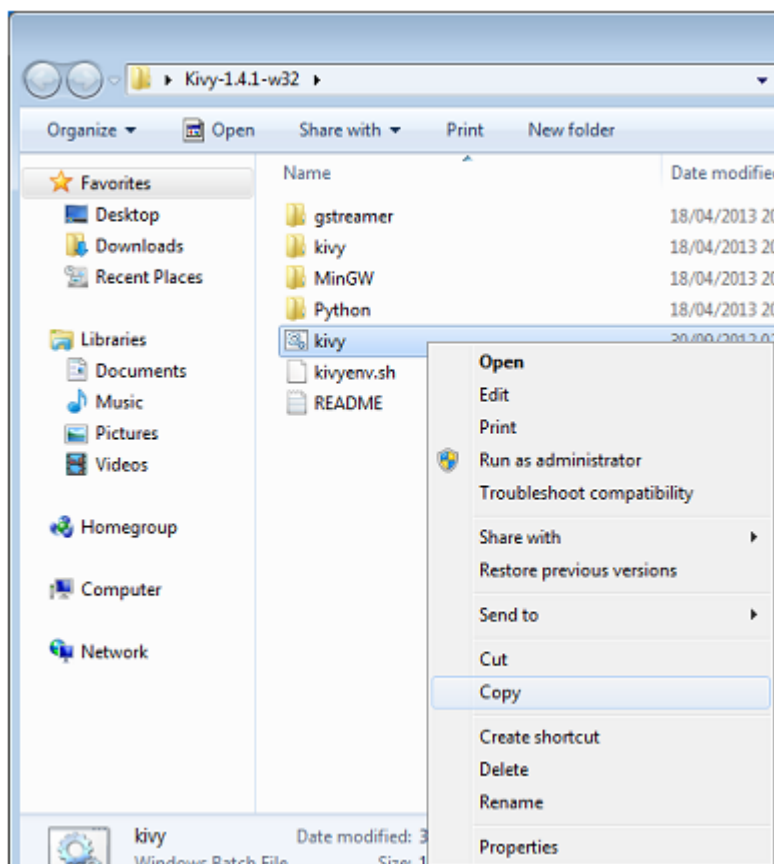


Fig. 3

2. Deschide Exploratorul de fișiere al windows-ului și scrie în address-bar <<shell:sendto>>

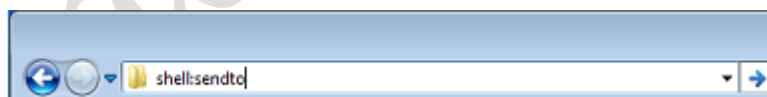


Fig. 4

3. Ar trebui să te trimită către un director special al windows-ului:

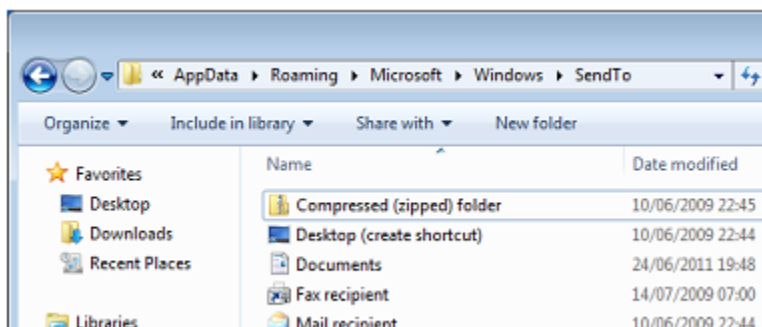


Fig. 5

4. Lipeste (paste) fisierul kivy.bat copiat anterior ca și shortcut(scurtatura):

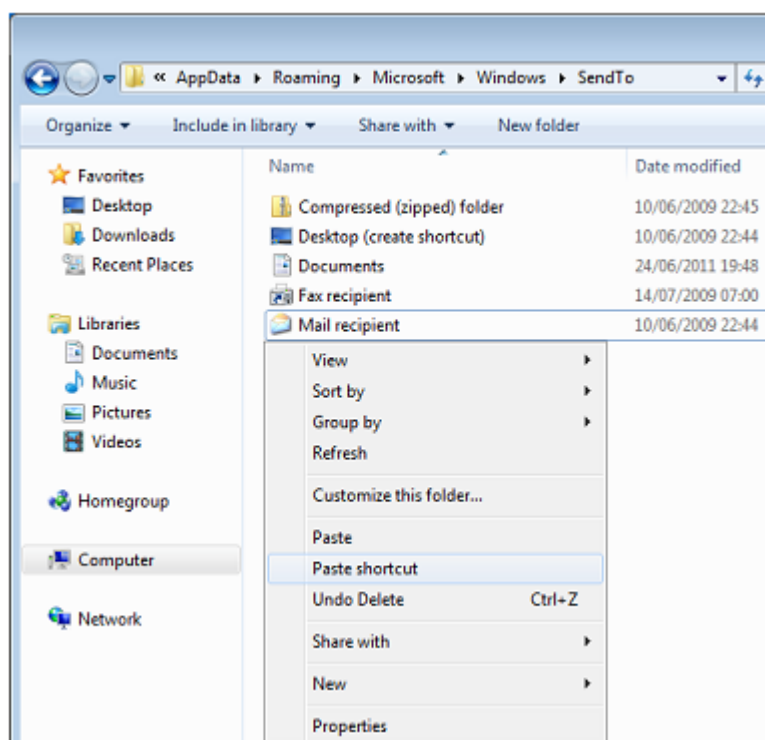


Fig.6

5. Redenumeste shortcut-ul ca kivy versiune.

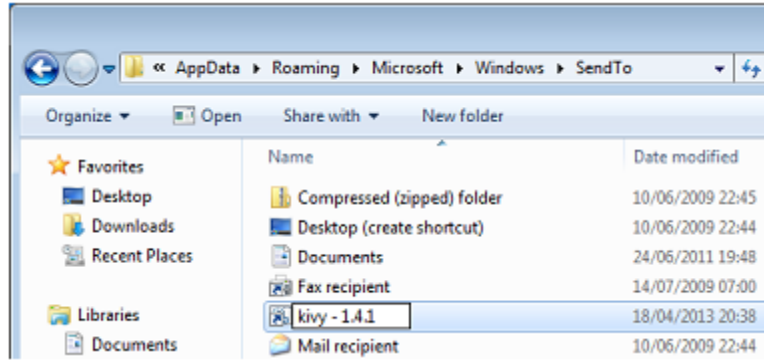


Fig.7

6. Acum poti rula kivy dand click pe aplicatia ta apoi apeland sendto și kivy.

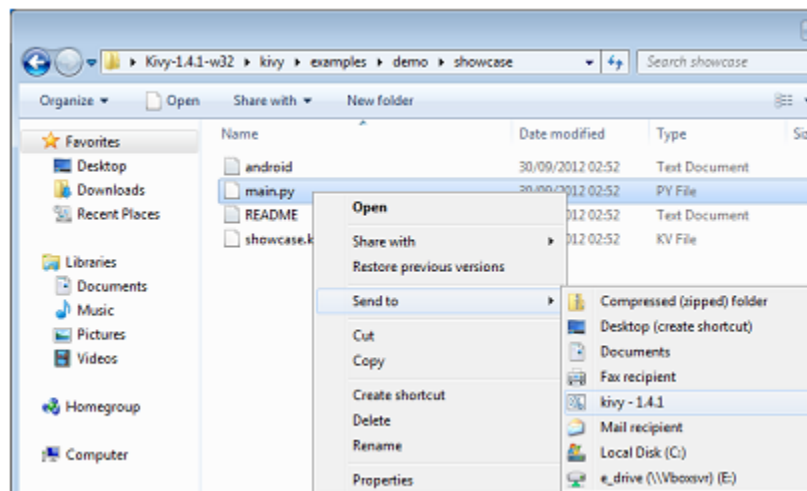


Fig.8

Metoda open-with:

1. Da click dreapta pe un fisier .py
2. Alege "Open with..." apoi "choose default program"
3. Navigheaza catre kivy.bat din fisierul dezarhivat portabil și selecteaza-l.
4. Apoi de fiecare data cand vei da dublu click pe tipul de fisier ".py" vei rula fisierul apeland kivy.bat

In cazul în care doriti să **instalati kivy manual** trebuie să aveti urmatoarele dependente (module). Aceasta metoda nu este recomandata deoarece de multe ori aceste dependente trebuiesc unite manual, altfel nu functioneaza:

- Glew 1.5.7

<http://sourceforge.net/projects/glew/files/glew/1.5.7/glew-1.5.7-win32.zip/download>

- Pygame 1.9.2

<http://pygame.org/ftp/pygame-1.9.2a0.win32-py2.7.msi>

- Cython 0.14 – necesita Visual Studio 2003/2008 pt. instalare

<https://pypi.python.org/packages/source/C/Cython/Cython-0.14.tar.gz>

- MinGW

<http://sourceforge.net/projects/mingw/files/Installer/mingw-get-setup.exe/download>

- GStreamer

<http://gstreamer.freedesktop.org/data/pkg/windows/1.4.1/gstreamer-1.0-x86-1.4.1.msi>

- Setuptools

https://bootstrap.pypa.io/ez_setup.py

O alta modalitate este **instalarea de kivy cu Python Extension Packages** realizata de Christoph Gohlke. Acestea sunt executabile de windows care instaleaza peste versiunea python curenta toate fisierele necesare pentru rularea modulelor dorite. Pagina de mai jos se ocupa de executabile 32/64 biti pentru Windows pentru pachete de extensii open-source in vederea distributiei CPython a limbajului Python . CPython reprezinta varianta standard de Python (varianta Python creata in limbajul C). Exista si alte variante precum Jython scris in java sau IronPython scris pentru Common Language Infrastructure.

Aceste executabile sunt neoficiale si au scop de testare si evaluare.

Trebuie instalate urmatoarele componente:

Pygame (varianta oficiala):

<http://pygame.org/ftp/pygame-1.9.2a0.win32-py2.7.msi>

Aceste pachete Python vin cu extensia (whl). Prin urmare aveti nevoie de wheel pentru instalare:

<https://pypi.python.org/pypi/wheel#downloads>

Kivy Python Extension Package - Christoph Gohlke extensia whl:

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#kivy>

Pentru a simplifica procedura in locul extensiei whl si a package-ului copiat de pe pagina lui Christoph Gohlke voi oferi pentru acest curs un executabil cu extensia *.exe.

Crearea unei aplicatii

Crearea unei aplicatii se realizeaza destul de simplu:

- In cadrul clasei trebuie să mostenim App()
- Trebuie să implementam metoda build() pentru a returna instanta widget (radacina tree-ului widget). Un root este acel obiect de tip graphic peste care se construiesc toata aplicatia. Un widget este un element grafic cum ar fi un buton, o fereastra etc.
Build initializeaza aplicatia și poate fi apelat doar o data. Dacă metoda returneaza un widget acesta va deveni root widget și toate celelalte widget-uri trebuie să devina copii la root widget.
- Initializarea acestei clase și apelarea metodei run pentru rulare

Iata un exemplu simplu de aplicatie ce nu are nici un widget aplicat, deci nu are root:

```
# Program kivy0
# Explica functiile kivy
# Ion Studentul - 1/26/13

from kivy.app import App

class PrimulProgramKivy(App):
    pass

PrimulProgramKivy().run()
```

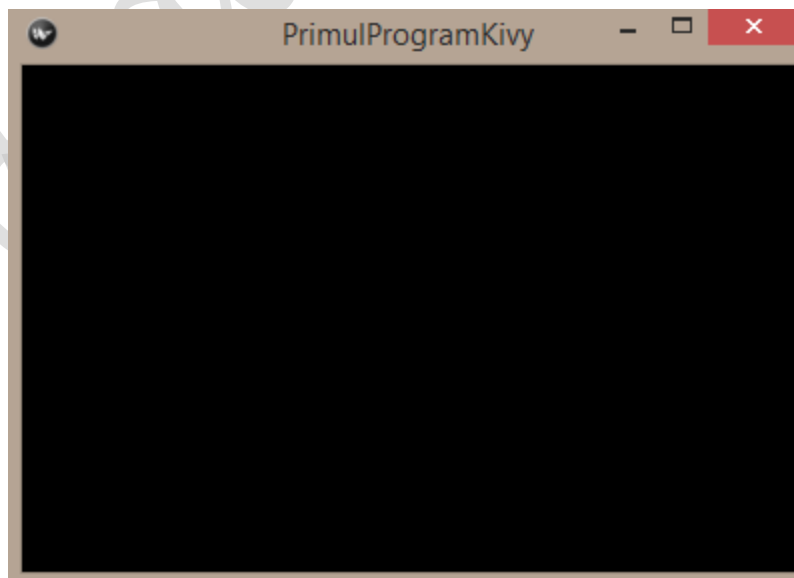


Fig .11

Puteti să salvati acest fisier sub un fisier .py și sa-l rulați. În urmatoarea sectiune vom explica cum functioneaza cel mai simplu program.

```
from kivy.app import App
```

Este necesara operatia de mosterire, adica clasa de baza să mosteneasca clasa app. Acest modul app se regaseste in:
kivy_installation_dir/kivy/app.py.

```
class PrimulProgramKivy(App):
```

Acasta contine o modalitate de a defini clasa de baza pt. aplicatia kivy, singurul lucru care poate fi schimbat aici este numele aplicatiei din `PrimulProgramKivy` în cum doriti să se numeasca.

Pt ca aplicatia să ruleze efectiv avem urmatoarele linii:

```
    PrimulProgramKivy().run()
```

Aceste linii au rolul de a initializa clasa `PrimulProgramKivy` și de a rula metoda `run()`

Iata un alt exemplu simplu de aplicatie:

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

```
from kivy.app import App  
from kivy.uix.label import Label
```

```
class MyApp(App):
```

```
    def build(self):  
        return Label(text='Hello world')
```

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



Fig.12

Puteti să salvati acest fisier sub un fisier .py și sa-l rulați.

In urmatoarea sectiune vom explica cum functioneaza programul rulat anterior. În primul rand trebuie să intelegem cum arata ciclul de viata Kivy App.

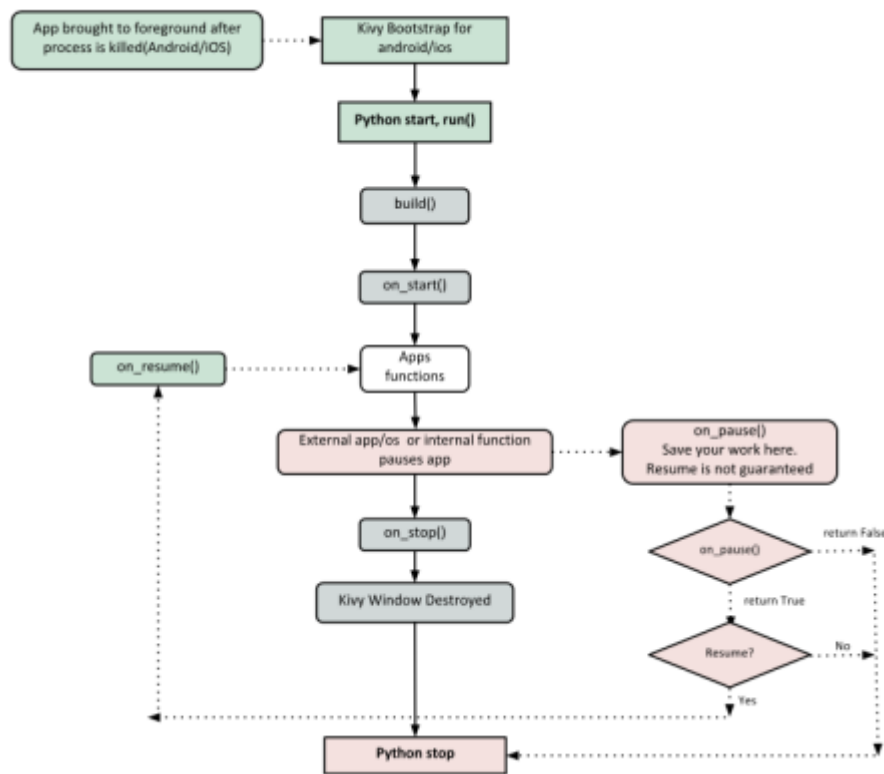


Fig.10

Dupa cum vedeti mai sus, punctual de intrare în aplicatie este metoda `run()`, care în cazul nostru este `MyApp().run()`. Inainte de a trece linia a treia, să discutăm un pic aceste concepte prezentate în Fig.10.

`on_start`: Aplicat cand programul a inceput; este apelata de `run()`.

`on_stop`: Aplicat cand programul a fost oprit.

`on_pause`: Aplicat caind aplicatia este pusa în modul pauza. Este un mod experimental, deci s-ar putea să nu se retuneze corect aplicatia momentan. În versiunile urmatoare, cu siguranta putem utiliza acet mod.

`on_resume`: Aplicat cand programul este repornit din modul pauza.

Linia a patra:

```
from kivy.app import App
```

Este necesar ca clasa de baza să mosteneasca clasa `app`. Aceasta se regaseste in

kivy_installation_dir/kivy/app.py.

```
from kivy.uix.label import Label
```

La aceasta linie dorim să importam din kivy o anumita functie numita Label().

De retinut ca modulul uix mentine elementele de user interface precum butoane si alte widget-uri.

Linia 6:

```
class MyApp(App):
```

Aceasta contine o modalitate de a define clasa de baza pt. aplicatia kivy, singurul lucru care poate fi schimbat aici este numele aplicatiei din MyApp în cum doriti să se numeasca.

Linia 7:

```
def build(self):
```

Aceasta functie are scopul să initializeze și returneze Root Widget.

```
return Label(text='Hello world')
```

Returnarea Root Widget. Acest root Widget este initializat cu o eticheta (label) cu textul "Hello World" si returneaza instanta

Pt ca aplicatia să ruleze efectiv avem urmatoarele linii:

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

Aceste linii au rolul de a initializa clasa Myapp și de a rula metoda run()

Prin rularea acestui cod va rezulta urmatoarea fereastră:

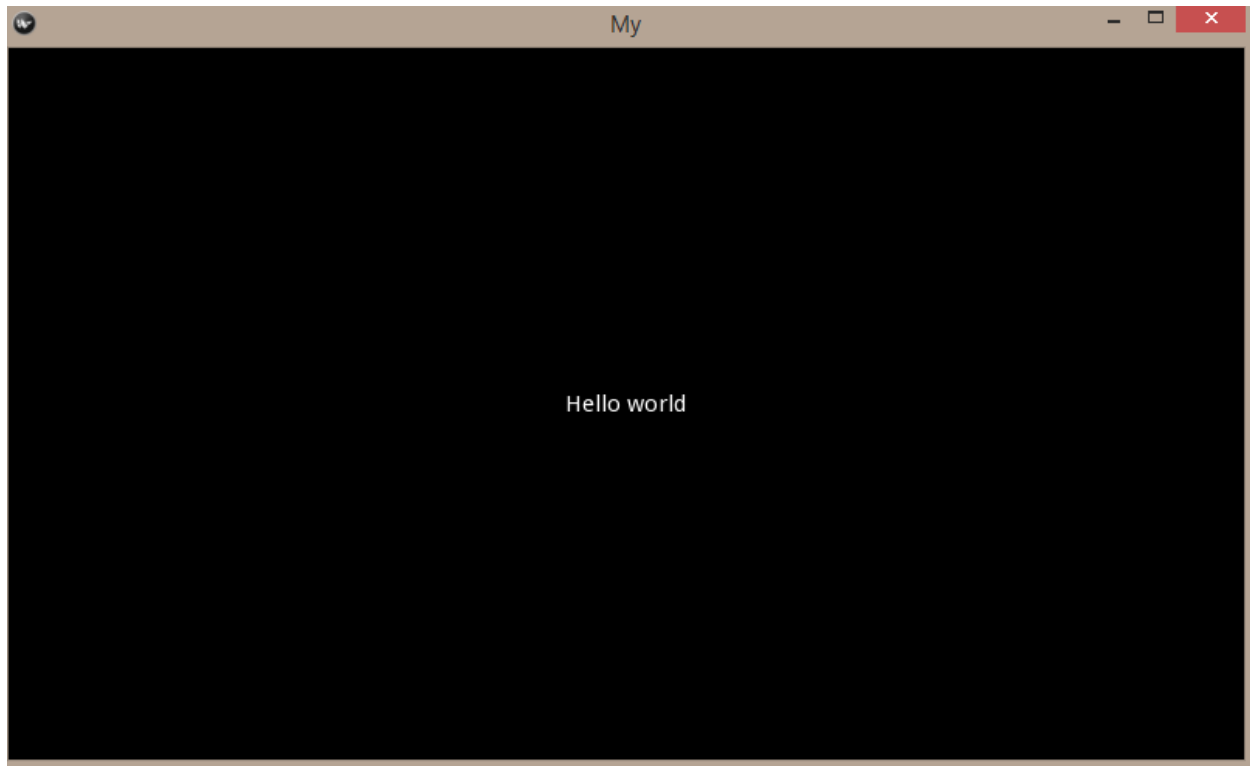


Fig.11

Label-ul este un widget. Subiect-ul widget este discutat in urmatoarea sectiune din aceasta sedinta.

Widget-uri

Când creai o aplicație, trebuie să te întrebi trei întrebări principale:

- Ce date are procesul meu de aplicație?
- Cum pot reprezenta vizual datele?
- Cum utilizatorul interacționează cu aceste date?

Dacă doriți de exemplu să scrieți o aplicație foarte simplă de desen, cel mai probabil doriți ca utilizatorul să deseneze doar pe ecran cu degetele. Acesta este modul în care utilizatorul interacționează cu aplicația dumneavoastră. În timp ce faci acest lucru, aplicația va memora pozițiile în care degetul utilizatorului a fost, astfel încât să puteți trasa mai târziu linii între aceste poziții. Deci, punctele în care degetele au fost ar fi datele și liniile pe care le trasezi dintr-o aceste puncte ar fi o reprezentare vizuală.

În Kivy, interfața de utilizator a aplicației este compusă din Widgets (widget= mic dispozitiv). Tot ceea ce vedeți pe ecran este oarecum desenat de un widget. Widget-ul răspunde la cele 3 întrebări : un widget încapsulează date, definește interacțiunea

utilizatorului cu acele datele și desenează reprezentarea sa vizuală. Puteți construi apoi orice, de la o simplă interfață la interfețe complexe pt. utilizator prin nesting widget.

Asa cum ne-am și imaginat widget-urile în kivy sunt organizate sub formă aborescentă. Aplicația noastră are un root widget, care va avea copii. Fiecare copil poate avea copii la rândul său, așa cum fiecare ramură mai mare poate avea ramurile, reprezentate în kivy ca o listă de proprietăți ale părintelui. Tree-ul widget poate fi manipulat cu următoarele metode:

- `add_widget()`: adaugă un widget ca un copil
- `remove_widget()`: elimină un copil de pe lista de copii
- `clear_widgets()`: elimină toți copii existenți de la un widget

Mai jos regăsim o listă cu toate Widget-urile kivy. Fiecare widget are un link asociat către pagina oficială <http://kivy.org/docs> pentru a reprezenta un ghid de început cu toate referințele necesare.

- **UX widgets:** Widget-uri clasice pt. user interface

[Label](#), [Button](#), [CheckBox](#), [Image](#), [Slider](#), [Progress Bar](#), [Text Input](#), [Toggle button](#), [Switch](#), [Video](#)

- **Layouts:** Un widget layout nu face redare de widget-uri, dar acționează ca un element de ordonare a widget-urilor. Puteți citi mai multe în secțiunea [Layout](#).

[Grid Layout](#), [Box Layout](#), [Anchor Layout](#), [Stack Layout](#)

- **Complex UX widgets:** Rezultatul combinării unor widget-uri. Le tratăm ca fiind complexe deoarece utilizarea lor nu este una ordinară.

[Bubble](#), [Drop-Down List](#), [FileChooser](#), [Popup](#), [Spinner](#), [List View](#), [TabbedPanel](#), [Video player](#), [VKeyboard](#),

- **Behaviors widgets:** Un widget behavior (eng. comportament) nu face redare de widget-uri, dar acționează în funcție de interacțiunile sau instrucțiunile grafice.

[Scatter](#), [Stencil View](#)

- **Screen manager:** Administrează ecrane și tranziții de la un ecran la altul.

[Screen Manager](#)

Nu toate widget-urile vor fi discutate deoarece unele sunt experimentale sau se pot utiliza doar utilizând limbajul kivy. Limbajul kivy este un limbaj special ce combină

tehnici python precum indentare cu tehnici XML pentru a crea o interfata grafica. Acest limbaj este prezentat în capitolul 8 (sedinta 8). Primul widget despre care trebuie să vorbim este Layout. Acesta se ocupa de modul cum sunt aranjate celelalte widget-uri în mod automat, deci toate celelalte widget-uri vor fi copii la un tip de layout. Am spus la un tip de layout deoarece nu putem utiliza modulul ca atare (deoarece exista mutiple module penutr fiecare tip), ci doar să mostenim un tip de layout și sa-l utilizam.

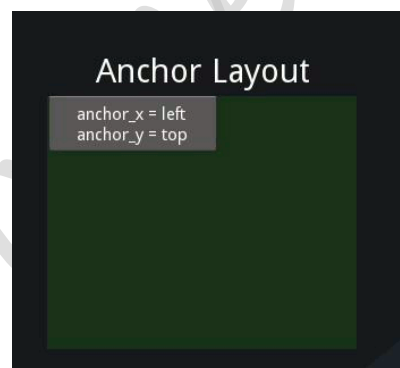
Layouts utilizeaza `size_hint` și `pos_hint` pt. a determina dimensiunea și pozitia copiilor.

Module: [kivy.uix.layout](#)

Tipuri de layout:

- Anchor layout :

Un layout simplu ce utilizeaza doar pozitiile copiilor. Permite să asezam copilul la o pozitie relativa fata de margimea layout-ului. Ancora `anchor_x` poate lua valorile left,center sau right. Ancora `anchor_y` poate lua valorile top,center sau bottom. Proprietatea `size_hint` a copilului nu va fi onorata.



- Box layout

Aranjeaza widget-urile intr-o maniera adiacenta, fie vertical, fie orizontal. Scopul este să umplem tot spatial layout-ului. Proprietatea `size_hint` a copilului poate fi utilizată pt. a schimba proportiile fiecarui copil sau să setam valori fixe pentru unele din ele.



- Float layout

Permite plasarea copiilor în locatii arbitrare și marimi arbitrare facand referire la valoarea absoluta sau relativa a layout-ului. Marimea standard a layout-ului este `size_hint (1, 1)`. Aceasta va face fiecare copil de asemenea dimensiune cu layout-ul deci e recomandat să o schimbam dacă dorim să avem mai mult de un copil. Putem seta `size_hint` la `(None, None)` pentru a utiliza marimea absoluta setata cu `size`. Acest widget ia în calcul și `pos_hint`, parametru care dicteaza pozitia relativa fata de layout.



- Grid layout :

Aranjeaza widget-urile intr-o grila. Trebuie să specifici cel puțin o dimensiune a grilei pt. a putea să calculeze mărimea elementelor și cum le aranjeaza.



- Stack layout :

Aranjeaza widget-urile adiacent unul fata de altul, dar cu o mărime setată în una din dimensiuni fără a încerca să umple tot spațiul.



Principalele caracteristici a unui layout este :

- “orientation” este o opțiune de tip proprietate de orientare și standard este ‘horizontal’. Poate fi ‘vertical’ sau ‘horizontal’. Pentru Stack opțiuni de orientare sunt: ‘lr-tb’, ‘tb-lr’, ‘rl-tb’, ‘tb-rl’, ‘lr-bt’, ‘bt-lr’, ‘rl-bt’ și ‘bt-rl’; unde lr înseamnă left to right și bt înseamnă bottom to top. Standard este ‘lr-tb’
- Mărimea copiilor este dată de size_hint_x (orizontal) și size_hint_y (vertical) unde valoarea este cuprinsă între 0 și 1 unde 1 este întreg layout-ul.
- “padding” reprezintă spațiul dintre layout-ul box și copil. De asemenea acceptă și două argumente separate: padding_horizontal, padding_vertical, dar și o formă unitară padding. Poate lua valorile: număr, padding_left, padding_top, padding_right, padding_bottom.
- spacing este o opțiune de tip proprietate ce setează spațiul dintre copii. Standard este 0.
- Pozitia child este utilizată doar de anchor. Si se realizează cu ajutorul cuvântului cheie pos_hint

Mai jos regăsim câteva exemple:

```
# Program kivy
# Explica funcțiile kivy - BoxLayout
# Ion Studentul - 1/26/13
```



```

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

class LoginScreen(BoxLayout):

    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.orientation='horizontal'
        self.padding=60
        self.buton1 = Button(text='Primul buton')
        self.add_widget(self.buton1)
        self.buton2 = Button(text='Al doilea buton')
        self.add_widget(self.buton2)

class MyApp(App):

    def build(self):
        x=LoginScreen()
        return x

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

```

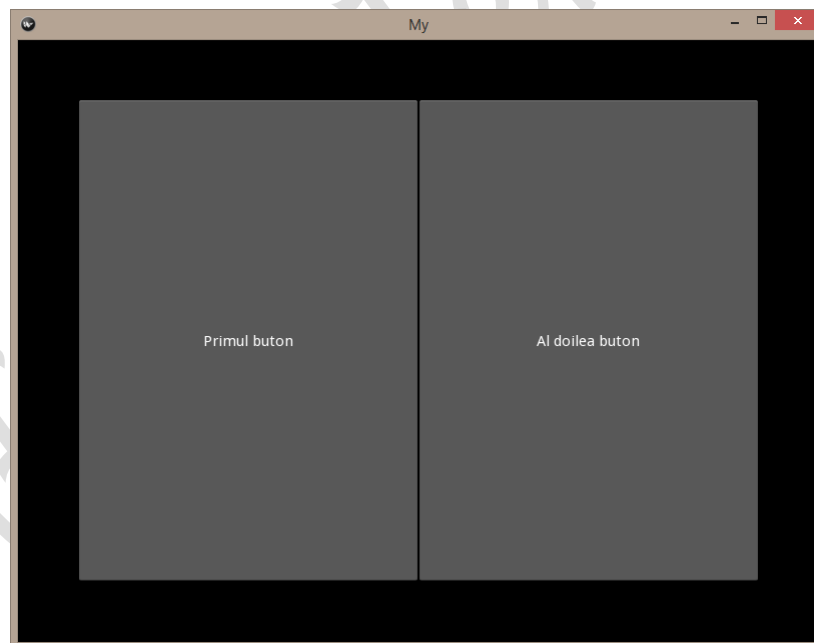


Fig.15

Observam ca în loc de widget-ul de tip label aici am utilizat un widget de tip buton, care are caracteristica ca dacă apasam pe el isi schimba culoarea. De asemenea are și un eveniment atasat la aceasta actiune a utilizatorului. Despre widget-ul button vom vb. Intr-o alta sectiune. Sa explicam fiecare linie in parte:

La următoarea linie importam widget-ul de tip Button:

```
from kivy.uix.button import Button
```

La următoarea linie importam widget-ul de tip BoxLayout:

```
from kivy.uix.gridlayout import BoxLayout
```

Aceasta clasa este folosita ca o bază pentru Root Widget (LoginScreen):

```
class LoginScreen(BoxLayout):
```

Urmatoarele doua linii au rolul de a mosteni toate attributele pe care le are boxlayout-ul. Este o scriere standard in Kivy. Clasa LoginScreen, va suprascrie `__init__` metoda `__init__()` pentru a adăuga widgeturi și pentru a defini comportamentul lor.

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

Nu trebuie să uităm să apelăm `super` în scopul de a implementa funcționalitatea clasei inițiale suprascrise. De asemenea, rețineți că aceasta este o bună practică să nu omită `kwargs`. Acest `kwargs` are rolul de a accepta argumente alterioare standard, deci fara aceasta parte nu va functiona.

```
self.orientation='horizontal'
```

Aceasta linie de mai sus are rolul de a seta orientarea Boxlayout-ului. Default este horizontal.

```
self.padding=60
```

Aceasta linie de mai sus are rolul de a seta un spatiu dintre marginea Boxlayout-ului si copii. Default este zero.

```
self.buton1 = Button(text='Primul buton')
```

Aceasta linie de mai sus are rolul de a crea un widget . Acesta va afisa textul `<<Primul buton>>`.

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

Aceasta linie de mai sus are rolul de a adauga la root widgetul `self.buton1` creat anterior.

```
self.buton2 = Button(text='Al doilea buton')
```

Aceasta linie de mai sus are rolul de a crea un widget . Acesta va afisa textul <<Al doilea buton>>.

```
self.add_widget(self.buton2)
```

Aceasta linie de mai sus are rolul de a adauga la root widgetul self.buton2 creat anterior.

Iata și primele exercitii singuri. Va indemn să rulați același cod și să realizați următoarele cerințe:

- Sa modificați padding la 20, apoi să rulați.
- Sa modificați orientarea la “vertical”, apoi să rulați
- Sa adaugați spacing cu valoarea 20, apoi să rulați
- Sa adaugați size_hint_x cu valoarea 0.5 și size_hint_y cu valoarea 0.3, apoi să rulați

Mai jos se poate vedea și soluția exercitiului propus:

```
# Program kivy6
# Explica functiile kivy - BoxLayout
# Ion Studentul - 1/26/13

import kivy

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

class LoginScreen(BoxLayout):

    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.orientation='vertical'
        #self.orientation='horizontal'
        self.padding=20
        self.spacing=20
        self.size_hint_x =0.5
        self.size_hint_y =0.3
        self.buton1 = Button(text='Primul buton')
        self.add_widget(self.buton1)
        self.buton2 = Button(text='Al doilea buton')
        self.add_widget(self.buton2)

class MyApp(App):

    def build(self):
        x=LoginScreen()
        return x
```

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

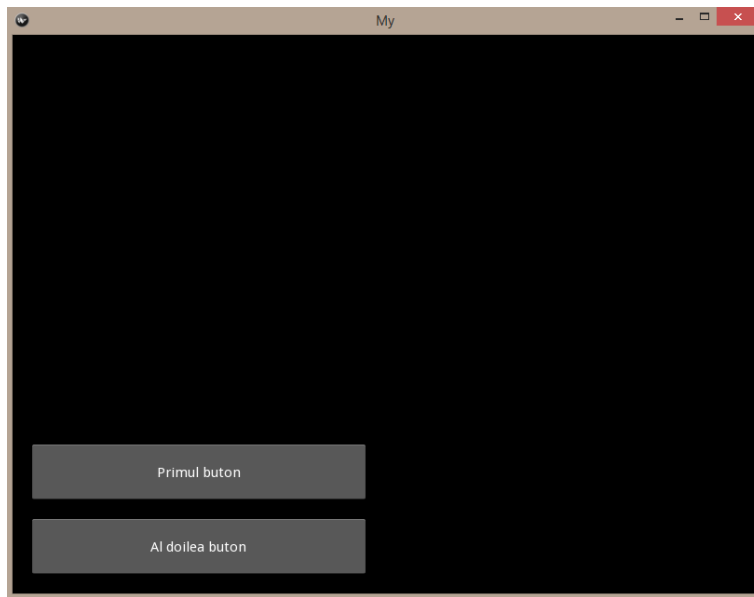


Fig .16

Iata si explicatiile pentru codul aditional ce se regaseste in programul de mai sus.

```
self.spacing=20
```

Spacing reprezinta un atribut al layout-ului ce indica spatiul dintre widget-urile child.

```
self.size_hint_x =0.5
self.size_hint_y =0.3
```

Size_hint este un tuplu de doua valori ce au scopul de a indica o marime ce nu este impusa, ci doar daca poate fi aplicata.

Mai jos regasim un exemplu similar care foloseste StackLayout in loc de BoxLayout.

```
# Program kivy7
# Explica functiile kivy - Stacklayout
# Ion Studentul - 1/26/13

import kivy

from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.stacklayout import StackLayout

class Stack_Layout(StackLayout):
```

```

def __init__(self, **kwargs):
    super(Stack_Layout, self).__init__(**kwargs)
    #self.orientation='vertical'
    self.orientation='lr-bt'
    self.padding=60
    self.size_hint_x =0.3
    self.size_hint_y =0.3
    self.buton1 = Button(text='Primul buton')
    self.add_widget(self.buton1)
    self.buton2 = Button(text='Al doilea buton')
    self.add_widget(self.buton2)

class MyApp(App):

    def build(self):
        x=Stack_Layout()
        return x

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

```

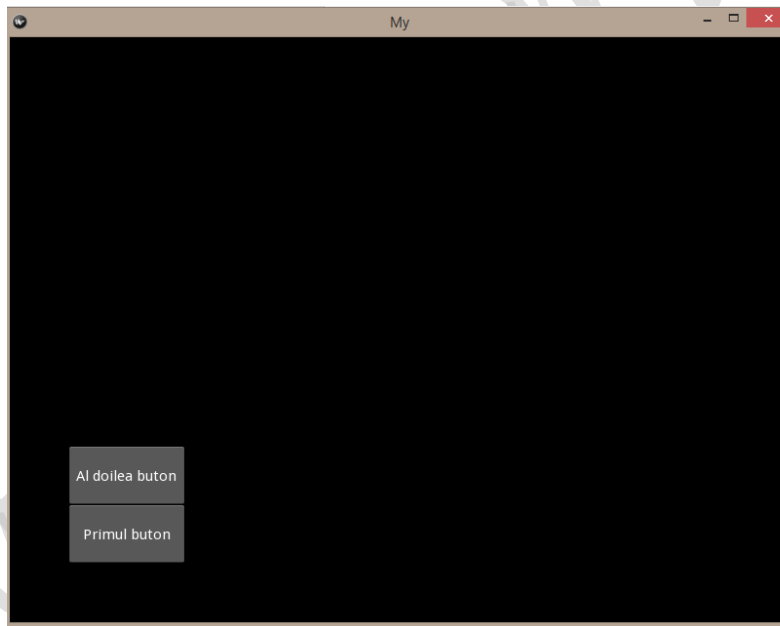


Fig.17

Crearea unui fundal pentru widget-ul de tip layout este o abordare destul de intalnita. Vom putea observa cum se realizeaza acest lucru intr-un program de mai jos.

```

# Program kivy6
# Explica functiile kivy - BoxLayout
# Ion Studentul - 1/26/13

import kivy

```

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

class Box_Layout(BoxLayout):

    def __init__(self, **kwargs):
        super(Box_Layout, self).__init__(**kwargs)
        self.orientation='vertical'
        self.padding=20
        self.spacing=20
        self.buton1 = Button(text='Primul buton')
        self.add_widget(self.buton1)
        self.buton2 = Button(text='Al doilea buton')
        self.add_widget(self.buton2)

class MyApp(App):

    def build(self):
        root = Box_Layout()
        root.bind(size=self._update_rect, pos=self._update_rect)

        with root.canvas.before:
            Color(0.4, 0.5, 0.6, 1) # blue;
            self.rect = Rectangle(size=root.size, pos=root.pos)
        return root

    def _update_rect(self, instance, value):
        self.rect.pos = instance.pos
        self.rect.size = instance.size

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

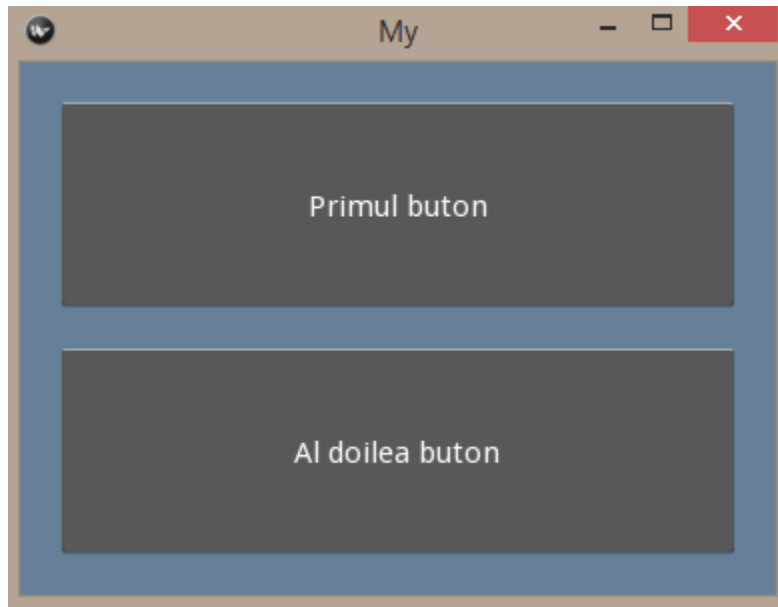


Fig.18

Layout-urile, prin natura lor, nu au o reprezentare vizuala deoarece nu au instructiuni de tip canvas(grafice). Totusi putem adauga instructiuni vizuale (canvas) destul de usor cum ar fi adaugarea de culoare ca fundal. Astfel:

Importam din modulul grafice `color`(eng. culoare) și `rectangle`(dreptunghi):

```
from kivy.graphics import Color, Rectangle
```

Adaugarea de canvas la instanta layout. Culoarele sunt exprimate ca fiind (R,G,B,A) cu valori de la 0 la 1. Adica R reprezinta rosu(aici 0.4), G reprezinta verde de la eng. green (aici 0.5) , B reprezinta albastru de la eng. Blue (aici 0.6) și A reprezinta luminozitatea de la eng alpha(aici 1). Dacă alpha este 0 atunci culoarea este negru indiferent de celelalte valori. Dacă toate sunt 1 atunci avem alb.

with layout_instance.canvas.before:

```
Color(0.4, 0.5, 0.6, 1) # blue
```

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

Din pacate aceste linii vor desena un dreptunghi colorat doar la desenarea initiala a layout-ului. Pentru a fi siguri ca `self.rect` este desenat de fiecare data cand layout-ul isi modifica dimensiunea trebuie să ascultam dupa schimbari și să luam o actiune. Asa cum am vazut anterior, acest lucru este posibil cu un eveniment, deci bind. Apoi vom rula de fiecare dată cand pozitia sau dimensiunea se modifica `update_rect`:

```
# listen to size and position changes
```

```
layout_instance.bind(pos=update_rect, size=update_rect)
```

```
with layout_instance.canvas.before:
```

```
    Color(0, 1, 0, 1) # green; colors range from 0-1 instead of 0-255
```

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

```
def update_rect(instance, value):
```

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

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

Trebuie să reținem ca fiecare widget din kivy are propriul canvas. Când tu creezi un widget creezi toate instrucțiunile pentru desenare.

În cele ce urmează vom discuta mai în amănunt despre primele 3 widget-uri pe care le-am folosit. Să începem cu Label. După cum ați văzut, label este utilizat pt. afișarea de text.

Pentru a schimba afișarea unui text putem utiliza [Text Markup](#). Acestea sunt similare cu BBcode (Bulletin Board Code) utilizate în html. Accesați [aici](#) pentru o listă completă de text markup.

Iată și două exemple pentru o înțelegere mai bună a conceptului.

```
# Program kivy8
# Explica funcțiile kivy
# Ion Studentul - 1/26/13

import kivy

from kivy.app import App
from kivy.uix.label import Label

class MyApp(App):

    def build(self):
        return Label(text='Salut \n [b]Lume[/b] !', markup=True )

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

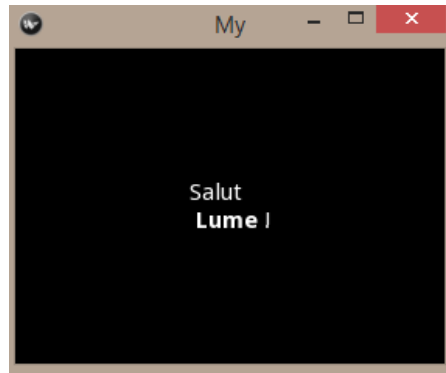



Fig.19

```
# Program kivy8
# Explica functiile kivy
# Ion Studentul - 1/26/13
```

```
import kivy
```

```
from kivy.app import App
from kivy.uix.label import Label
```

```
class MyApp(App):
```

```
    def build(self):
        return Label(text='Salut \n [size=32]Lume![/size] ', markup=True )
```

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

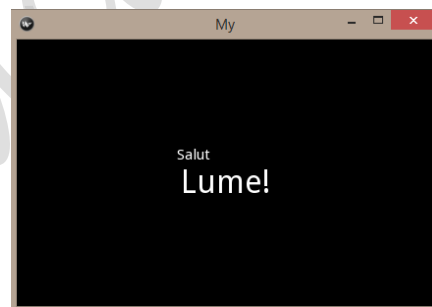


Fig.20

Label are și un eveniment pe care îl putem utiliza: `on_ref_press`

Acesta se activează când utilizatorul da click pe grupul de cuvinte dintre `[ref]` tag.

Alte proprietăți

- `font_size` - mărimea font-ului
- `color` – culoarea textului

- italic - dacă textul este italic. True sau False
- padding - reprezinta spatierea dintre marginea label-ului și text. Poate avea și padding_x și padding_y.

De retinut ca un Label va avea ca background culoarea layout-ului !

```
# Program kivy
# Explica functiile kivy - caracteristici label
# Ion Studentul - 1/26/13

from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.boxlayout import BoxLayout

class LoginScreen(BoxLayout):

    def __init__(self, **kwargs):
        super(LoginScreen, self).__init__(**kwargs)
        self.orientation='horizontal'
        self.label1=Label(text='Salut Lume!',
                           italic=True,
                           font_size=46,
                           color=(1,0.2,0.2,0.8))
        self.add_widget(self.label1)

class MyApp(App):

    def build(self):
        x=LoginScreen()
        return x

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



Fig.21

Asa cum se poate vedea in programul de mai sus, putem avea text-ul de la label scris inclinat(italic), putem dicta font size-ul (marimea caracterelor) sau culoarea.

Urmatorul widget este Text Input. Acesta ofera posibilitatea ca utilizatorul să scrie un text.

Evenimente :

- `on_text_validate` adica la apasarea enter. Acest eveniment s-a utilizat și în primele exemple.
- `focus` – selectarea widget-ului.

Exemplu:

```
textinput = TextInput()  
textinput.bind(focus=metoda_aplicata)
```

Proprietati:

- `multiline` – accepta o singura linie (`multiline = False`) sau multiline (`multiline = True`). Standard `multiline = True`
- Text-ul widget-ului `textinput` este stocat în obiect `text`
- `font_size` - marimea font-ului

- foreground_color – culoarea textului
- background_color – culoarea fundalului
- hint_text_color – culoarea hint text
- hint_text – Afiseaza un text inainte să dai click sau să scrii ceva.
- padding - reprezinta spatierea dintre marginea label-ului și text. Poate avea și padding_x și padding_y.

Iata și un exemplu cu aceste charactersistici:

```
# Program kivy9
# Explica functiile kivy - text input
# 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

class Test_input(GridLayout):

    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.add_widget(Label(text='un Label'))
        self.text_input_1 = TextInput(background_color = (0.4,0.6,0.8,1))
        self.text_input_1.foreground_color = (0,1,0,1)
        self.text_input_1.font_size = 26
        self.text_input_1.padding=20
        self.text_input_1.hint_text_color = (1,0,0,1)
        self.text_input_1.hint_text="07xx.xxx.xxx!"
        self.add_widget(self.text_input_1)

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

Iata și rularea acestui program:

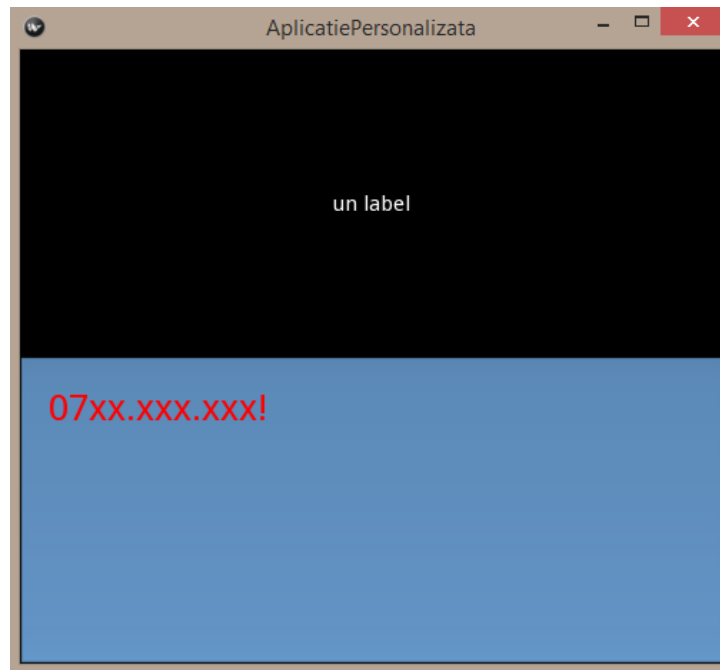


Fig. 22 - Program rulat fara a selecta text input

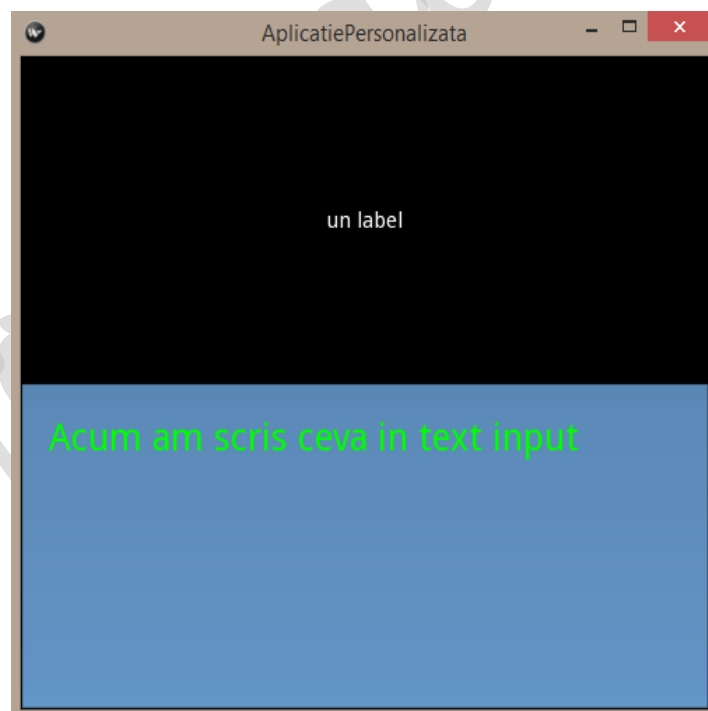


Fig. 23

Alte caracteristici ale text input:

- password – textul introdus de utilizator va fi de forma **** (stelute)
- background_normal – aceasta optiune permite adaugarea unei poze de fundal în loc de cea clasica. Starea este neselectata
- background_active – aceasta optiune permite adaugarea unei poze de fundal în loc de cea clasica. Starea este selectata

Mai jos regasim și un exemplu:

```
# Program kivy9
# Explica functiile kivy - text input
# 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

class Test_input(GridLayout):

    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.add_widget(Label(text='un Label'))
        self.text_input_1 = TextInput()
        self.text_input_1.background_normal = 'text-input_normal.png'
        self.text_input_1.background_active = 'text-input_active.png'
        self.text_input_1.password=True
        self.text_input_1.font_size = 80
        self.text_input_1.padding=60
        self.text_input_1.hint_text_color = (1,0,0,1)
        self.text_input_1.hint_text="07xx.xxx.xxx!"
        self.add_widget(self.text_input_1)

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

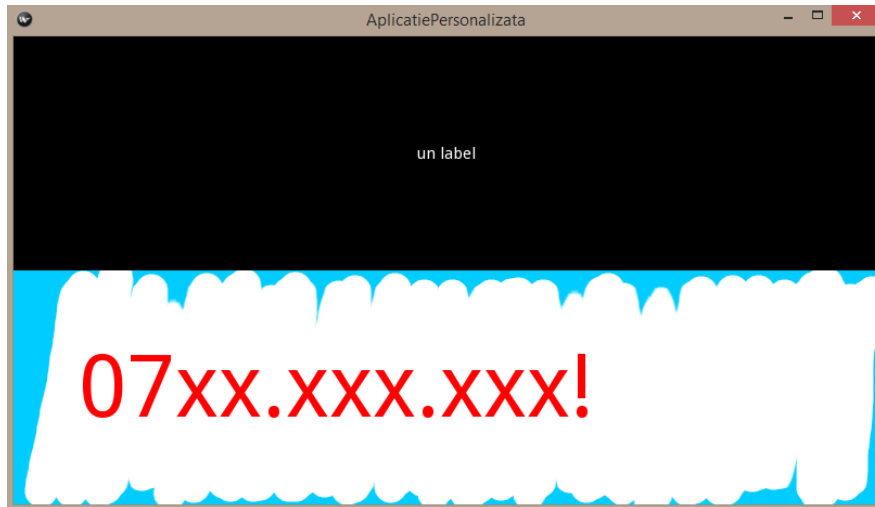


Fig.24 – Text input neselectat

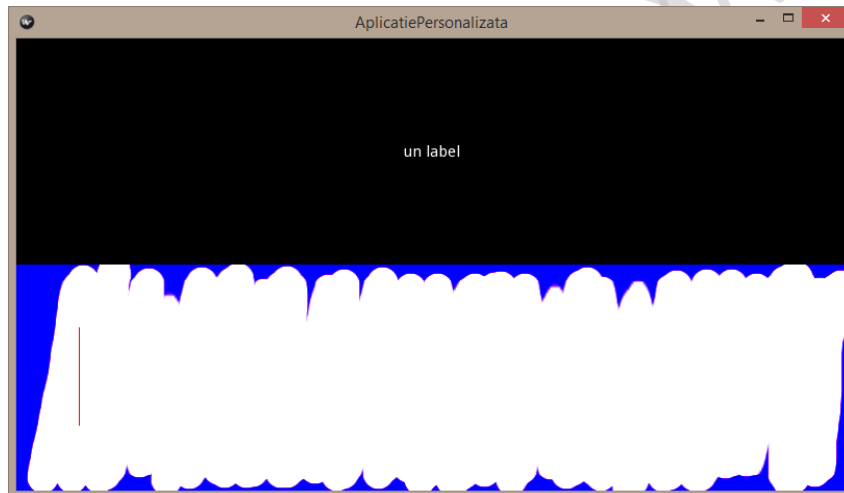


Fig.25 Text input selectat

În cele ce urmează vom discuta despre checkbox. Acest widget poate fi utilizat ca checkbox sau ca buton radio. Mai jos regăsim primul exemplu unde vom utiliza un checkbox. Ca eveniment vom folosi `active`. Acesta se activează de fiecare dată când schimbăm starea butonului. De asemenea vom ține o înregistrare a stării prin variabila globală `check1` pentru a ști starea actuală.

```
# Program kivy9
# Explica functiile kivy - text input
# 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.checkbox import CheckBox
check1=0

class Test_input(GridLayout):

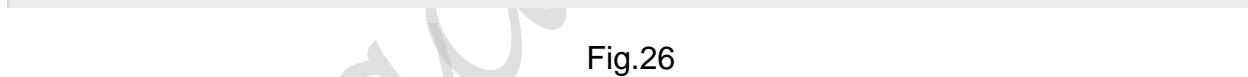
    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.text = Label(text='un Label')
        self.add_widget(self.text)
        self.checkbox = CheckBox()
        self.add_widget (self.checkbox)
        self.checkbox.bind(active=self.Ruleaza_la_activare)

    def Ruleaza_la_activare(self ,value ,instance):
        global check1
        if (check1 %2 == 0) :
            print('The checkbox', self.checkbox, 'is active')
        else:
            print('The checkbox', self.checkbox, 'is inactive')
        check1 += 1

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

10



```
# Program kivy9
# Explica functiile kivy - text input
# Ion Studentul - 1/26/13
```

```
class Test_input(GridLayout):

    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.text = Label(text='un Label')
        self.add_widget(self.text)
        self.checkbox1 = CheckBox(text="1")
        self.checkbox1.group = "test"
        self.add_widget(self.checkbox1)
        self.checkbox2 = CheckBox()
        self.checkbox2.group = "test"
        self.add_widget(self.checkbox2)
        self.checkbox1.bind(active=self.Ruleaza_la_activare)

    def Ruleaza_la_activare(self, value, instance):
        global check1
        if (check1 % 2 == 0) :
            print('The checkbox1 is active')
        else:
            print('The checkbox2 is active')
        check1 += 1

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

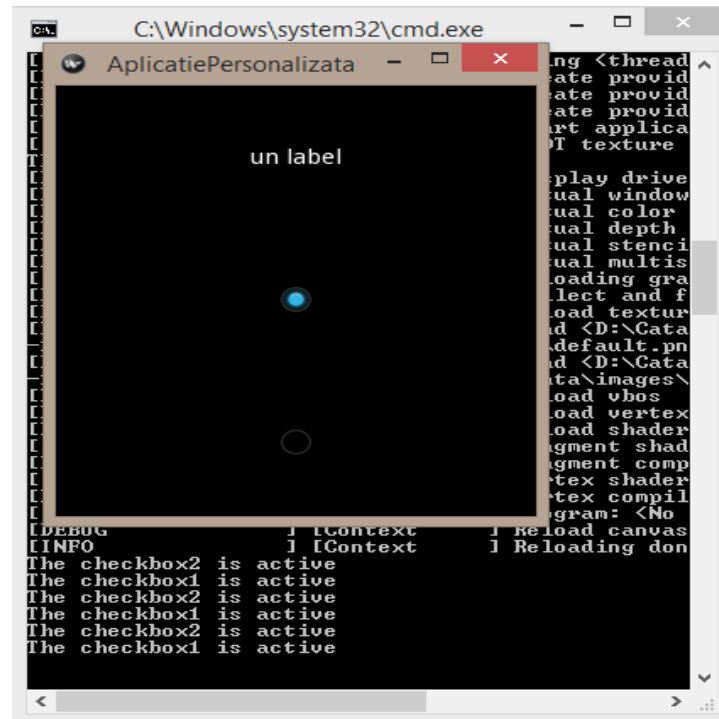


Fig. 27

Similar cu widget-ul checkbox , dar mai apropiat de un buton normal este Toggle Button. Acesta are în plus fata de checkbox și posibilitatea de a avea un label asociat cu o descriere a acestei stari. Poate fi descris ca un buton care va fi în starea selectat sau deselectat. Toggle Button se poate utiliza ca un buton normal, avand acelasi proprietati și evenimente ca un buton normal.

```
# Program kivy9
# Explica functiile kivy - toggle button
# 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.togglebutton import ToggleButton
check1=0

class Test_input(GridLayout):

    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.text = Label(text='un label')
        self.add_widget(self.text)
        self.Toggle = ToggleButton(text="Bifeaza-ma!")
        self.add_widget (self.Toggle)
        self.Toggle.bind(on_press=self.Ruleaza_la_activare)
```

```

def Ruleaza_la_activare(self ,value ):
    global check1
    if (check1 %2 == 0) :
        print('The Toggle button is active')
    else:
        print('The Toggle button is inactive')
    check1 += 1

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

```

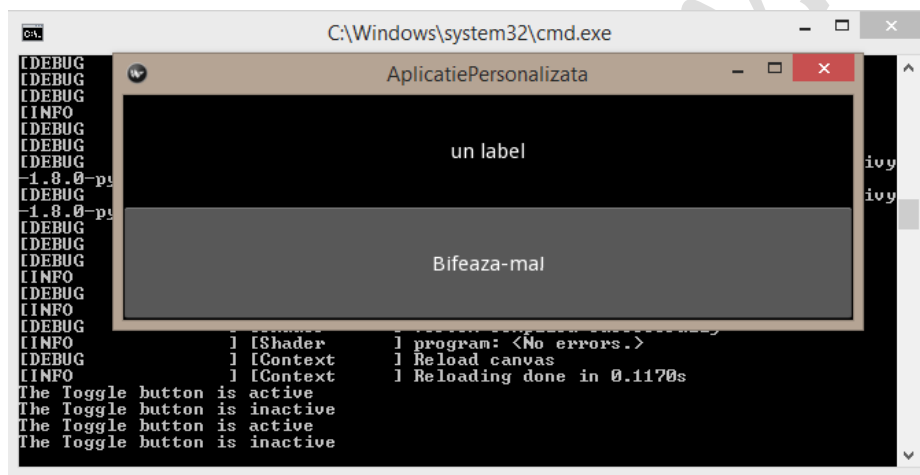


Fig.28 Toggle Button cu starea neselectata.

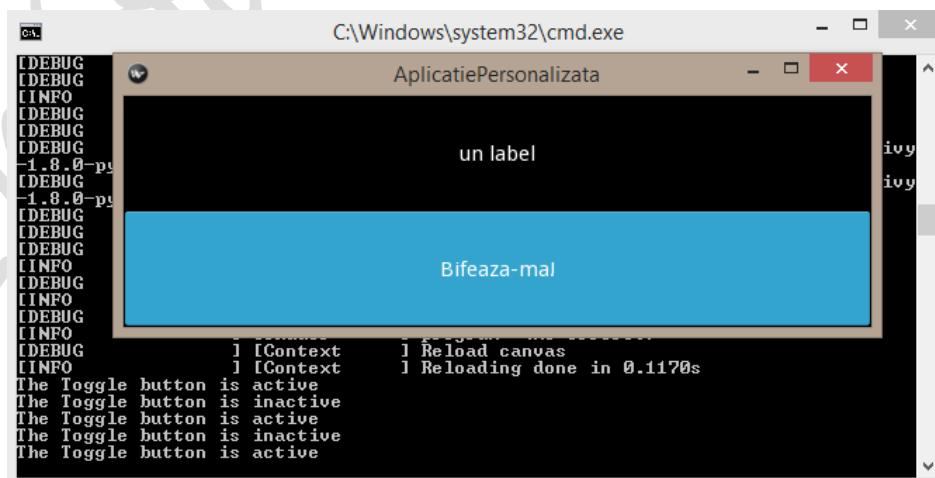


Fig.29 Toggle Button cu starea selectata.

Mai jos regasim un al doilea exemplu în care folosim Toggle button ca un buton radio. Obiectele de tip Toggle button trebuie sa faca parte din acealasi grup.

Aici groupul este "Eu cu cine votez?"

```
# Program kivy10
# Explica functiile kivy - toggle button
# 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.togglebutton import ToggleButton
check1=0

class Test_input(GridLayout):

    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.text = Label(text='un label')
        self.add_widget(self.text)
        self.Toggle1 = ToggleButton(text="Voteaza-ma!")
        self.Toggle1.bold = "True"
        self.Toggle1.group = "Eu cu cine votez?"
        self.Toggle1.background_color = (1,0,0,1)
        self.add_widget (self.Toggle1)
        self.Toggle2 = ToggleButton(text="Ba voteaza-ma pe mine!")
        self.Toggle2.bold = "True"
        self.Toggle2.group = "Eu cu cine votez?"
        self.Toggle2.background_color = (1,0,0,1)
        self.add_widget (self.Toggle2)
        self.Toggle3 = ToggleButton(text="Eu promit să nu fur ... prea mult!")
        self.Toggle3.bold = "True"
        self.Toggle3.group = "Eu cu cine votez?"
        self.Toggle3.background_color = (1,0,0,1)
        self.Toggle3.state = "down"
        self.add_widget (self.Toggle3)
        self.Toggle1.bind(on_press=self.Ruleaza_la_activare_t1)
        self.Toggle2.bind(on_press=self.Ruleaza_la_activare_t2)
        self.Toggle3.bind(on_press=self.Ruleaza_la_activare_t3)

    def Ruleaza_la_activare_t1(self ,value ):
        global check1
        check1 = 1

    def Ruleaza_la_activare_t2(self ,value ):
        global check1
        check1 = 2

    def Ruleaza_la_activare_t3(self ,value ):
```

```

global check1
check1 = 3

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

```

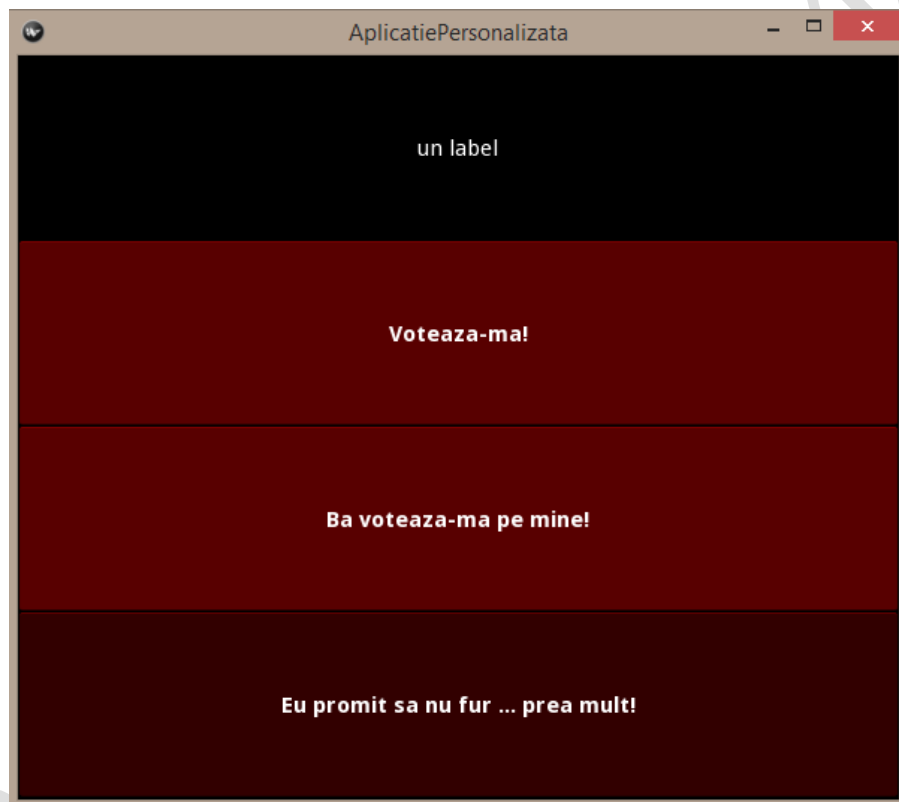


Fig 30

Asa cum se poate vedea putem selecta una din aceste stari. Starea default este reprezentata de state="down". Doar un singur buton poate fi state down pe o perioada de timp dacă face parte din acelasi grup. Aici grupul este "Eu cu cine votez?". De asemenea, s-au utilizat proprietatile bold = true, am schimbat culoarea toggle button în rosu.

Asa cum o să vedem și în cazul butoanelor, dar similar cu checkbox widget putem schimba imaginea clasica a unui toggle button. Astfel mai jos regasim un exemplu în care utilizam schimbarea imaginii default a unui toggle buton.

```

# Program kivy11
# Explica functiile kivy - toggle button
# 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.togglebutton import ToggleButton
check1=0
check2=0

class Test_input(GridLayout):

    def __init__(self, **kwargs):
        super(Test_input, self).__init__(**kwargs)
        self.cols = 1
        self.Toggle1 = ToggleButton(text = "muzica")
        self.Toggle1.background_normal = "on.png"
        self.Toggle1.background_down = "off.png"
        self.add_widget (self.Toggle1)
        self.Toggle2 = ToggleButton(text = "efecte")
        self.Toggle2.background_normal = "on.png"
        self.Toggle2.background_down = "off.png"
        self.add_widget (self.Toggle2)
        self.Toggle1.bind(on_press=self.Ruleaza_la_activare_t1)
        self.Toggle2.bind(on_press=self.Ruleaza_la_activare_t2)

    def Ruleaza_la_activare_t1(self ,value ):
        global check1
        if (check1 %2 == 0) :
            print('The Toggle1 button is active')
        else:
            print('The Toggle1 button is inactive')
        check1 += 1

    def Ruleaza_la_activare_t2(self ,value ):
        global check2
        if (check2 %2 == 0) :
            print('The Toggle2 button is active')
        else:
            print('The Toggle2 button is inactive')
        check2 += 1

class AplicatiePersonalizata(App):
    def build(self):
        return Test_input()

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

```

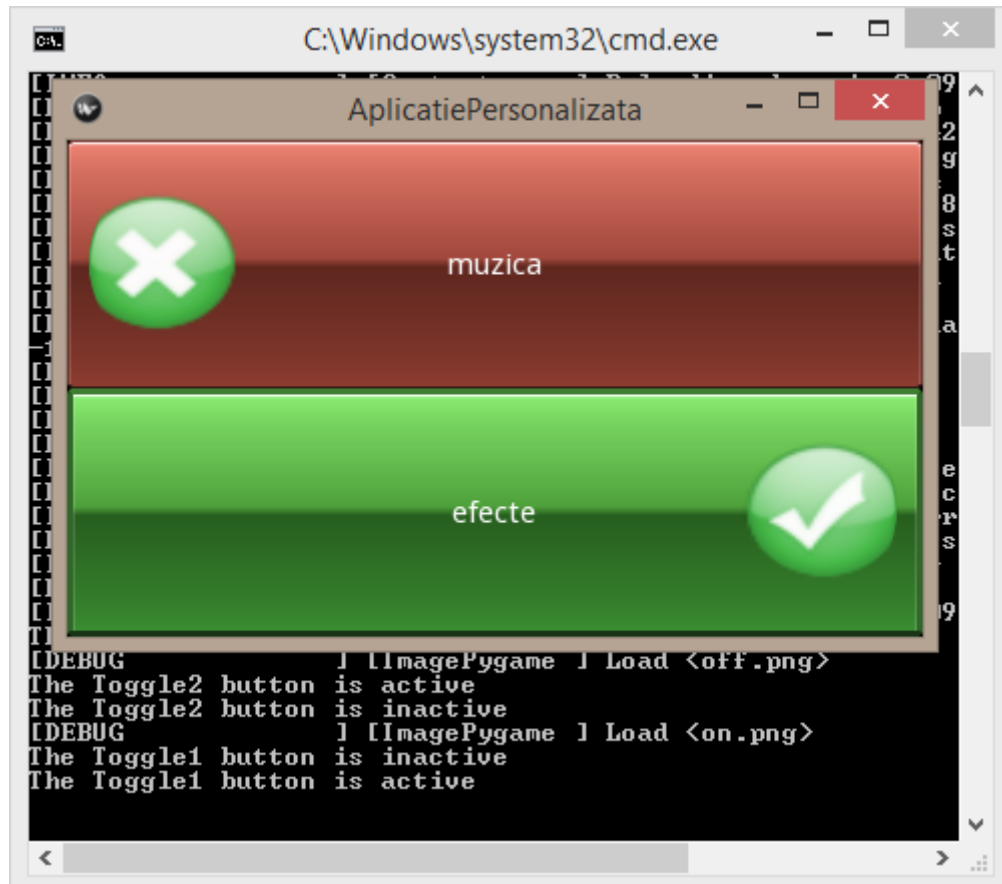


Fig.31

Asa cum se poate vedea, cu ajutorul proprietatilor `background_normal` și `background_down` putem incarca doua imagini ce au ca rol crearea celor doua stari neapasat (normal) și apasat (down).

Ultimul subiect propus azi este widget-ul button. Acesta se prezinta ca un widget ce are tot doua stari ca toggle cu diferenta ca nu ramane apasat. Mai jos regasim un exemplu complex ce surprinde și widget-ul button în actiune. Astfel putem adauga doua poze pentru cele doua stari ale butonului normal și down(apasat).

```
# Program kivy12
# Explica functiile kivy - Button
# 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.button import Button
import sys

class Test_input(GridLayout):
```



```

def __init__(self, **kwargs):
    super(Test_input, self).__init__(**kwargs)
    self.cols = 1
    self.text = Label(text='un label')
    self.add_widget(self.text)
    self.buton1 = Button(text="Seteaza Copyright")
    self.buton1.background_normal = "normal.png"
    self.buton1.background_down = "apasat.png"
    self.add_widget(self.buton1)
    self.buton1.bind(on_press=self.Ruleaza_la_activare)

def Ruleaza_la_activare(self, value):
    self.copyright = "Infoacademy"
    print "Copyright setat"

class AplicatiePersonalizata(App):

    def build(self):
        return Test_input()

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

```

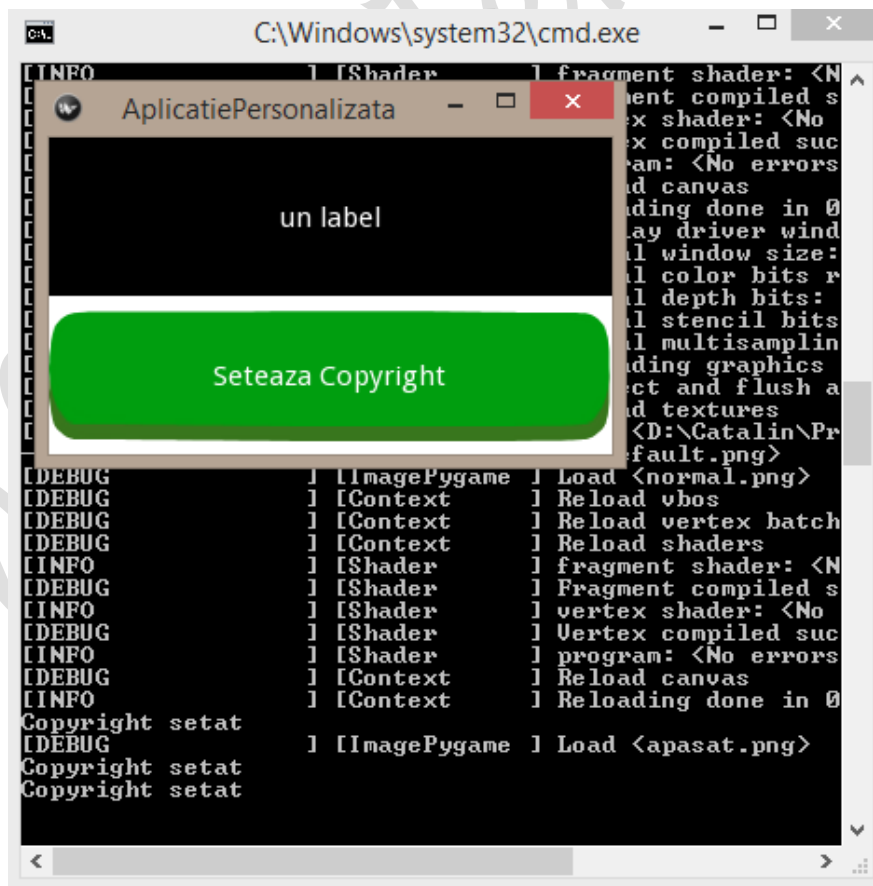


Fig.32

Personalizarea unei aplicatii

În cele ce urmează vom crea o aplicație ce are ca scop autentificarea la o aplicație.

În prima fază creăm un program care doar creează cadrul necesar autentificării. În a doua etapă vom crea un mic program ce va realiza și o autentificare.

Încercăm să manipulăm un program pentru a arăta flexibilitatea pe care o deținem, dar și să înțelegem cum realizăm un program. Pentru prima fază programul propus este următorul:

```
# Program kivy2
# 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

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)

class AplicatiePersonalizata(App):

    def build(self):
        return LoginScreen()

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

Să explicăm un pic programul de mai sus:

În următoarea linie importăm GridLayout:

```
from kivy.uix.gridlayout import GridLayout
```

Aceasta clasa este folosita ca o bază pentru Root Widget (LoginScreen) definit la linia 9:

```
class LoginScreen(GridLayout):
```

La linia 12 în clasa LoginScreen, va suprascrie `__init__` metoda `__init__()` pentru a adăuga widgeturi și pentru a defini comportamentul lor:

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

Nu trebuie să uităm să apelăm `super` în scopul de a implementa funcționalitatea clasei inițiale suprascrise. De asemenea, rețineți că aceasta este o bună practică să nu omită `kwargs` **. Acest `kwargs` are rolul de a accepta argumente alterioare standard, deci fara aceasta parte nu va functiona.

```
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)
```

În secțiunea de mai sus, cerem `GridLayout` să gestioneze copii sai în două coloane. Pe fiecare rand se adaugă o etichetă și un `TextInput` pentru numele de utilizator și parola.

Rularea scriptului genereaza o fereastră ca în figura de mai jos

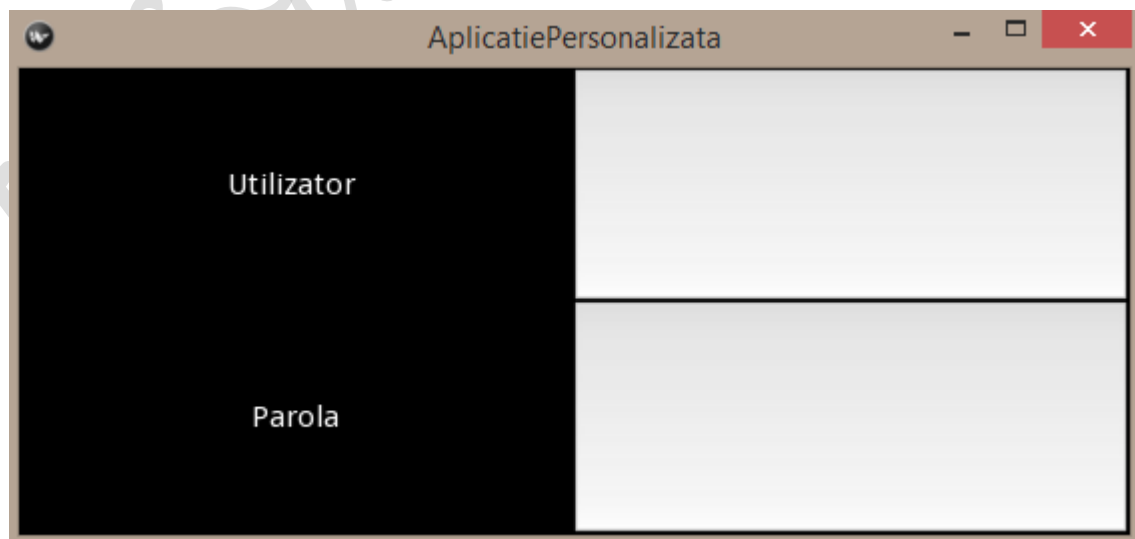


Fig.11

Etapa a doua presupune unirea programului invatat anterior în care afisam o fereastra cu un label pentru care cream o protectie de securitate de tip user și parola.

```
# Program kivy2
# 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

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):
        return LoginScreen()

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

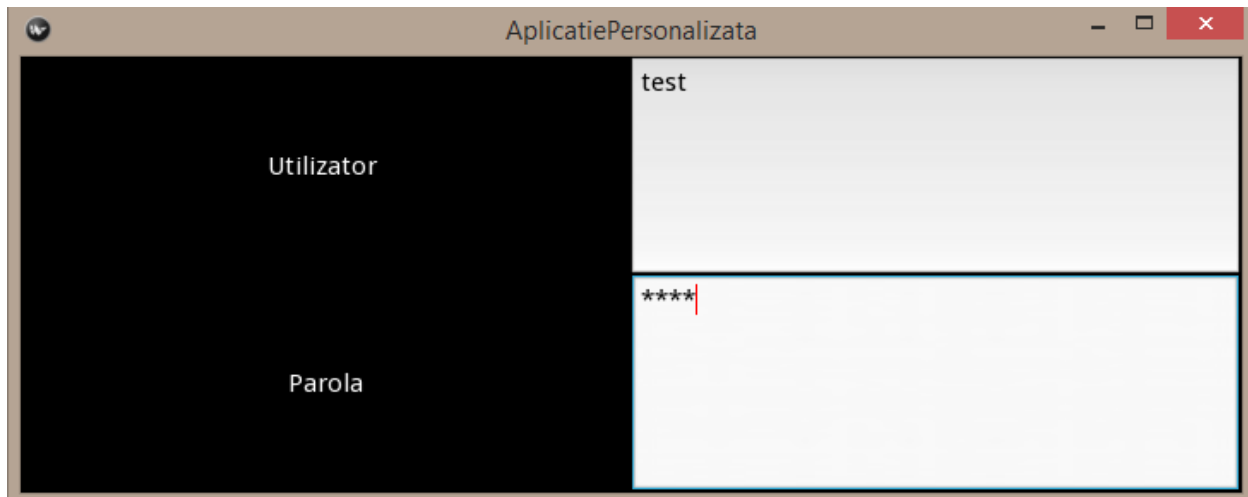


Fig.12

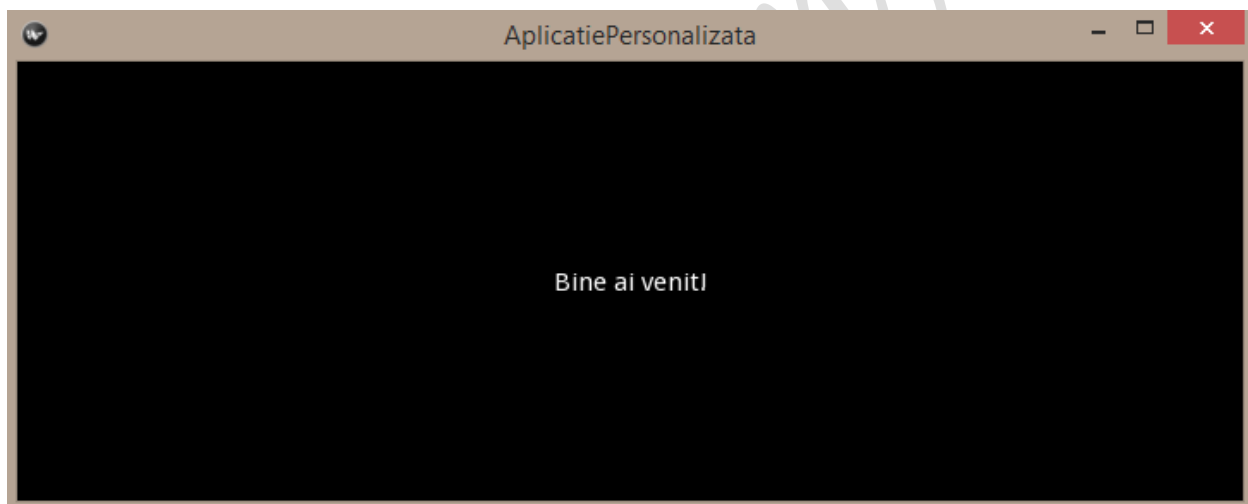


Fig.13

Elementele noi în acest program se regasesc mai jos:

```
self.password.bind(on_text_validate= self.verific_user_si_parola)
```

Aceasta linie are rolul de a lega un eveniment (aici enter) de o metoda. Prin urmare, actiunea enter data de utilizator în widget-ul text input numit self.password va activa rulara metodei verific_user_si_parola.

Evenimentele sunt nelipsite în programarea grafica, astfel prin modularizarea unor mici functionalitati de baza, codul poate fi rescris mai repede. Fiecare widget are diferite evenimente la care noi putem să asociem anumite metode construite de noi. Trebuie de asemenea să intelegem ca noi construim peste root widget toate celelalte child widget.

Vom incerca să exemplificam alte evenimente posibile în cazul text input pentru a intelege mai usor cum utilizam aceste :

on_text_validate

Declansat doar dacă multiline este False și utilizatorul tasteaza 'enter'.

on_double_tap

Declansat cand un text input este dublu click-uit. Comportamentul standard este să selecteze textul din jurul cursorului. Mai multe informatii la [on_double_tap\(\)](#).

on_triple_tap

Declansat cand un text input este triplu click-uit. Comportamentul standard este să selecteze linia din jurul pozitiei . Mai multe informatii la [on_triple_tap\(\)](#).

Urmatoarea linie are scopul de a crea o noua metoda ce va fi apelata cand vom da enter în campul password.

```
def verific_user_si_parola(self,t):
```

Liniile de mai jos au rolul de a extrage textul introdus de utilizator

```
    text_user = self.username.text
    text_pass = self.password.text
```

Urmeaza o verificare a acestor valori extrase. Astfel, dacă user-ul introdus și parola introdusa au ambele valoarea test, vom sterge toate widget-urile anterioare apoi vom crea un alt widget cu un label.

```
    if (text_user == "test" and text_pass=="test"):
        self.clear_widgets()
        self.add_widget(Label(text='Bine ai venit!'))
```

In cazul în care nu sunt indeplinite criteriile cerute atunci vom selecta textul introdus de utilizator în campul username și il vom sterge. La fel vom proceda și cu textul introdus în campul parola.

```
    else:
        self.username.select_all()
        self.username.delete_selection()
        self.password.select_all()
        self.password.delete_selection()
```