ТЕМА. Графічний інтерфейс Kivy

Мета. Вивчення бібліотеки графічного інтерфейсу kivy і отримання практичних навиків розроблення програм з графічним інтерфейсом за допомогою мов Python і Kivy.

Вступ. Kivy - це кросплатформова бібліотека для Python, призначена для створення графічних інтерфейсів користувача засобами Python. Бібліотека Kivy має також засоби для створення програм для сенсорних екранів. Програми написані на Kivy можуть працювати не тільки на комп'ютерних платформах таких як Linux, Windows, Mac OS, але й також на смартфонах з операційними системами Android, IOS.

План.

1. Встановлення Kivy.
2. Створення базової програми на Python і Kivy.
3. Графічні елементи Kivy.
4. Графічний елемент "Надпис".
5. Графічний елемент "Кнопка".
6. Графічний елемент "Поле введення"
7. Графічний елемент "Випадаючий список".
8. Впорядкування елементів за допомогою контейнера layout.
9. Оброблення подій.
10. Підтримка дат і часу в Kivy.

1. Встановлення Kivy

Процес встановлення бібліотеки залежить від використовуваної операційної системи.
Інструкції встановлення Kivy в ОС Ubuntu.

1. Додати пакет kivy до пакетного менеджера за допомогою команди

```
sudo add-apt-repository ppa:kivy-team/kivy
```

2. Оновити список пакетів використовуючи пакетний менеджер

```
sudo apt-get update
```

3. Встановити kivy для версії Python3:

```
sudo apt-get install python3-kivy
```

Інструкції встановлення Kivy в ОС Windows:

1. Впевнитися, що встановлені останні версії `pip` та `wheel` за допомогою команди

```
python -m pip install --upgrade pip wheel setuptools
```

2. Встановити залежні пакети, які потрібні для Kivy

```
python -m pip install docutils pygments pypiwin32 kivy.deps.sdl2 kivy.deps.glew
```

3. Встановити Kivy

```
python -m pip install kivy
```

Для того щоб перевірити чи бібліотека встановилась коректно потрібно ввійти в оболонку Python і імпортувати kivy. Команда не повинна видати помилку, а результат буде наступним.

```
andriy@andriy-desktop:~/MEGA/PNU/Python/kivy$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import kivy
[INFO   ] [Logger      ] Record log in /home/andriy/.kivy/logs/kivy 19-05-05 2.txt
[INFO   ] [Kivy        ] v1.10.1
[INFO   ] [Python      ] v3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0]
>>>
```

2. Створення базової програми на Python і Kivy

Після встановлення бібліотеки Kivy можна створювати графічні інтерфейси за стосунків з використанням тільки мови Python або мов Python і Kivy. Використання мов Python і Kivy дозволяє розділити логіку застосунку від його презентації.

Приклад створення найпростішої Kivy програми з використанням Python для виведення повідомлення “Hello world”.

Щоб створити Kivy інтерфейс потрібно імпортувати модуль `app` з Kivy:

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

і графічний елемент Label:

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

Основна частина програми:

```
class FirstKivy(App):
    def build(self):
        return Label(text="Hello Kivy!")
```

В основній частині програми клас `FirstKivy` успадковує клас `App`. Для створення застосунку необхідно повернути графічний елемент у функції `build()`. У прикладі повертається надпис “Hello Kivy”.

Запуск програми на виконання:

```
FirstKivy().run()
```

Звичайно цей виклик розміщується у конструкції

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

для того щоб програма запускалася тільки при безпосередньому виконанні файлу, а не при його імпортуванні.

Приклад програми на Python:

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

class FirstKivy(App):
    def build(self):
        return Label(text='Hello world')

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

Приклад програми на Python і Kivy (1.8.0). Ківі програма посилається на батьківський тег <Labe> у Python програмі і присвоює його властивості text значення `'Hello'+' World!'`.

```
#FirstKivy.py
from kivy.app import App
from kivy.uix.label import Label
```

```
class FirstKivy(App):
    def build(self):
        return Label()

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

#FirstKivy.kv
#:kivy 1.8.0

<Label>
     text: 'Hello'+' World!'
```

Результат виконання програми:

Основна перевага у використанні мови Kivy – можливість задання властивостей графічних елементів в окремій програмі, що суттєво спрощує Python код. У цьому мова Kivy подібна до CSS і стилів, які використовуються в HTML.

3. Графічні елементи Kivy

Елементи графічного інтерфейсу в Kivy називаються віджетами. Всі віджети Kivy знаходяться в модулі kivy.uix. Для використання віджету спочатку необхідно його імпортувати, а потім створити його об'єкт. В Kivy доступні наступні віджети:

Таблиця 1 – Віджети Kivy

Label	Надпис
Button	Кнопка
Checkbox	Позначка в квадраті
Image	Картина
Slider	Повзунок
Progress bar	Індикатор заповнення
Text input	Поле введення
Toggle button	Перемикальна кнопка
Switch	Двопозиційний перемикач
Drop-down list	Випадаючий список елементів
FileChooser	Вибір файлів

Popup	Підказка
Spinner	Випадаючий список значень

4. Графічний елемент “Надпис”

У першій програмі використовувався лише один графічний елемент надпис (Label). Для задання розміру надпису потрібно визначити розмір шрифту за допомогою властивості `font_size`. Властивість задається в конструкторі Label:

```
Label(text="Hello Label", font_size='100')
```

Вигляд надпису із заданим розміром шрифту:

До надписів також можна застосовувати стилі, такі як: жирний текст, курсив, підкреслений та зміна кольору. Стилі задаються у властивості `text` тегами, як у мовах розмітки сторінок HTML. Наприклад, [u]…[/u] – підкреслення, [b]…[/b] – жирний шрифт, [i]…[/i] – курсив і [color]…[/color] – визначення кольору. Для цього також необхідно присвоїти властивості `markup` значення True.

```
Label(text='[color=ff0066] [b] Welcome [/b] [/color] To [i] [color=ff9933] Kivy [/i] [/color]', markup = True, font_size=45)
```

Надпис з такими заданими властивостями матиме вигляд:

5. Графічний елемент “Кнопка”

Часто використовуваним графічним елементом є Кнопка, яку можна натискати, вмикати/вимикати, змінювати розмір і позицію, колір, надпис.

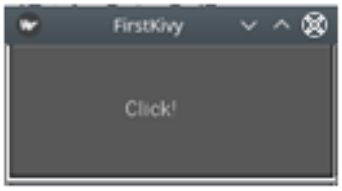
Щоб створити у формі кнопку потрібно імпортувати клас `Button`.

```
from kivy.app import App
from kivy.uix.button import Button
class FirstKivy(App):

    def build(self):
        return Button(text="Click!")

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

Як видно з результу виконання програми кнопка займає весь простір, бо її розміри не вказані. Для того щоб задати розмір та місце розташування кнопки потрібно використати властивості `size_hint` та `pos`. Параметр `pos` задає початок координат, який знаходиться в нижньому лівому куті вікна. Інші параметри задаються в конструкторі кнопки через кому:

```
Button(text="Click!", pos=(300,350), size_hint = (.25, .18))
```

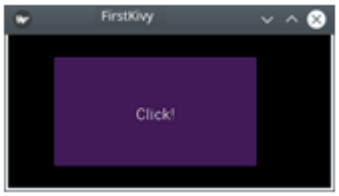
Якщо задати ці параметри в кнопці попереднього прикладу то її вигляд зміниться:



Щоб задати колір кнопки потрібно використати властивість `background_color`. Перші три параметри властивості відповідають кольорам RGB, а четвертий – прозорості. Діапазон значень параметрів від 0 до 1.

```
Button(text="Click!",pos=(300,350),size_hint=(.25,.18),background_color=(0.7,
0.3,1,1))
```

Кнопка із такими властивостями матиме вигляд:



Задати активний чи неактивний стан кнопки можна за допомогою властивості `disabled`. Якщо `disabled=True`, то кнопка перейде в неактивний стан і стане більш тьмяною, а натискання не будуть відображатись.

```
Button(text="Click!",pos=(300,350),size_hint=(.25,.18),
background_color=(0.75,0.3,1,1),disabled = True)
```

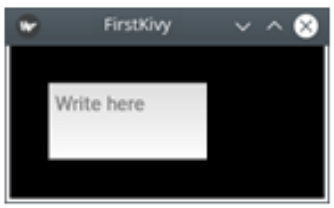
6. Графічний елемент “Поле введення”

Поле введення використовується для передачі даних від користувача до програми. Для створення поля введення потрібно імпортувати клас `TextInput`.

Приклад програми з одним графічним елементом “Поле введення”:

```
from kivy.app import App
from kivy.uix.textinput import TextInput
class FirstKivy(App):

   def build(self):
       return     TextInput(hint_text='Write     here',     size_hint=(.4,.1),
pos=(250,250))

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

Результат виконання програми:



В цьому прикладі для поля введення задані розмір (щоб воно не займало весь екран) і позиція розміщення. Також задана властивість `hint_text`, яка визначає текст підказки. Підказка

виводиться на фоні порожнього поля введення, як тільки користувач введе один символ підказка зникає. Введені користувачем дані зберігається у властивості `text`.

Для однорядкового поля введення задається властивість `multiline=False`:

```
TextInput(hint_text='Write here', multiline=False)
```

7. Графічний елемент “Випадаючий список”

Графічний елемент “Випадаючий список” (Drop-down list) відображає список елементів під головним елементом до якого він прив’язаний. В списку можуть міститися будь-які графічні елементи. Список розгортається при натисканні на головний елемент. Випадаючий список елементів може використовуватися для створення меню.

Імпортування випадаючого списку

```
from kivy.uix.dropdown import DropDown
```

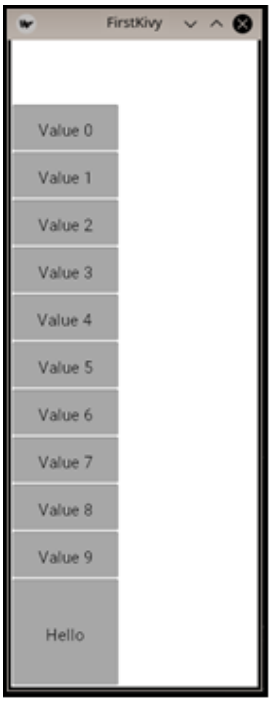
Створення об’єкту випадаючого списку `dropdown = DropDown()`. Після створення об’єкту випадаючого списку до нього можна добавляти графічні елементи за допомогою методу `dropdown.add_widget(btn)`. Прив’язується список до певного елемента методом `bind`. Для цього в елементі, під яким має розгортатися список, визначається подія для якої буде розгортатися список `mainbutton.bind(on_release=dropdown.open)`. `Mainbutton` це об’єкт кнопки під якою буде розгортатися список і в якій визначена подія `on_release` для розгортання списку (`dropdown.open`).

Приклад програми для створення випадаючого списку:

```
from kivy.app import App
from kivy.uix.dropdown import DropDown
from kivy.uix.button import Button

class FirstKivy(App):

 def build(self):
   dropdown = DropDown()
   for index in range(10):
     btn = Button(text='Value {0}'.format(index), size_hint_y=None,
height=44)
     dropdown.add_widget(btn)
   mainbutton = Button(text='Hello', size_hint=(None, None))
   mainbutton.bind(on_release=dropdown.open)
   return mainbutton

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

У програмі створено випадаючий список, до нього додано десять кнопок, створено кнопку під якою буде розгортатися список і в ній визначено подію для розгортання списку.

Результат виконання програми:



8. Впорядкування елементів за допомогою контейнера layout

Layout – це контейнер, який використовується для розміщення графічних елементів певним способом:

`AnchorLayout` – розміщення елементів вверху, внизу, зліва, справа.

`BoxLayout` – розміщення елементів послідовно вертикально або горизонтально.

`FloatLayout` – розміщення елементів необмежене.

`RelativeLayout` – відносне розміщення елементів.

`GridLayout` – розміщення елементів у сітці, яка визначається властивостями `rows` і `cols`.

`PageLayout` – створення простих багатосторінкових розміщень для швидкого перемикання між сторінками.

`ScatterLayout` – розміщення подібне до `RelativeLayout`, але елементи можуть бути розтягнуті, повернуті і масштабовані.

`StackLayout` – поміщаються у стек зліва направо і зверху вниз.

Приклад розміщення елементів у контейнері `BoxLayout`:

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.boxlayout import BoxLayout
class FirstKivy(App):

    def build(self):
        self.lay=BoxLayout(orientation='horizontal',padding=25,spacing=10)
        self.button1=Button(text="Click!")
        self.button2=Button(text="Click!")
        self.button3=Button(text="Click!")
        self.button4=Button(text="Click!")

        self.lay.add_widget(self.button1)
        self.lay.add_widget(self.button2)
        self.lay.add_widget(self.button3)
        self.lay.add_widget(self.button4)
        return self.lay
```

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

У цьому прикладі спочатку створюється контейнер BoxLayout, а потім до нього додають екземпляри кнопок. Метод `build` повертає `BoxLayout` з його внутрішніми елементами.

Результат виконання програми:

Приклад розміщення елементів у контейнері `GridLayout`. Спочатку створюється контейнер сітка розміром 2×2, а потім додаються кнопки. Контейнер `GridLayout` підтримує властивості відступу `padding` і `space`.

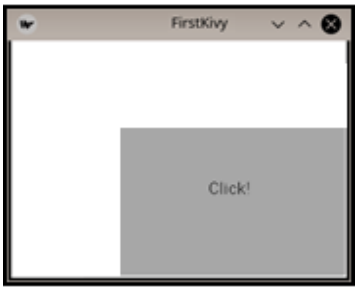
```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
class FirstKivy(App):

    def build(self):
        self.lay = GridLayout(cols=2,rows=2,padding=25, spacing=10)
        self.button1=Button(text="Click!")
        self.button2=Button(text="Click!")
        self.button3=Button(text="Click!")
        self.button4=Button(text="Click!")

        self.lay.add_widget(self.button1)
        self.lay.add_widget(self.button2)
        self.lay.add_widget(self.button3)
        self.lay.add_widget(self.button4)
        return self.lay

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

Результат виконання програми:

Приклад розміщення елементів у контейнері `AnchorLayout` відносно рамок контейнера. Контейнер `AnchorLayout` має два основних параметра `anchor_x` та `anchor_y`, які відповідають розміщенню елемента по горизонтальному та вертикальному краю контейнера відповідно. `anchor_x` може мати значення `left, center, right`, а `anchor_x - top, center, bottom`.

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.anchorlayout import AnchorLayout
class FirstKivy(App):

    def build(self):
        self.lay = AnchorLayout(anchor_x='right', anchor_y='bottom')
        self.button1=Button(text="Click!", size_hint = (0.3, 0.3))
        self.lay.add_widget(self.button1)
        return self.lay

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

Результат виконання програми:



Приклад розміщення елементів у контейнері `FloatLayout`.

```
#SimplyKivy.py
from kivy.app import App
from kivy.uix.floatlayout import FloatLayout

class SimpleKivy(App):
    def build(self):
        return FloatLayout()

if __name__ = ”__main__”:
    SimpleKivy().run

#SimpleKivy.kv
<Button>:
    Font_size: 40
    Color: 0,1,0,1
    Size_hint: 0.3, 0.2

<FloatLayout>:
    Button:
       text ”Kivy”
       pos_hint: {”x”: 0, ”y”: 0}

    Button:
       text ”Tutorials”
       pos_hint: {”right”: 0.5, “top”: 1}
```

Результат виконання програми:

9. Оброблення подій

Для того щоб програма була функціональною, тобто виконувала якісь дії, потрібно щоб елементи графічного інтерфейсу реагували на дії користувача, це реалізується за допомогою подій. В Kivy майже для всіх графічних елементів є список подій які активуються при певних діях користувача. Задання певних подій для елемента та їх прив'язка до певного метода, який буде обробляти події, виконується за допомогою функції `bind()`. Параметри елементів, незалежно від того чи вони були визначені в конструкторі елемента, можна змінювати в програмі, наприклад `self.btn.text="Click!"`. Тут в об'єкті `btn` змінюється властивість `text`, аналогічно можна змінювати і інші параметри.

Приклад програми:

```
from kivy.app import App
from kivy.uix.button import Button
import random

class FirstKivy(App):
    def btn_callback(self, instance):
        instance.background_color=(random.random(),random.random(),random.rando
 m(),1)

    def build(self):
        self.btn=Button(text="Click!", pos=(300,350), size_hint = (.25, .18))
        self.btn.bind(on_press=self.btn_callback)
        return self.btn

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

В цьому прикладі є тільки один графічний елемент кнопка із заданими розміром та позицією. За допомогою виклику `self.btn.bind(on_press=self.btn_callback)` до об'єкту кнопки прив'язується подія `on_press` і цій властивості присвоюється метод `btn_callback`. Тепер, при кожному натисканні кнопки викликатиметься метод `btn_callback`. Метод `btn_callback` має два аргументи `self` і `instance`. В `instance` передається посилання на об'єкт над яким зробили дію і яка визначена в ньому. У тілі метода `btn_callback` змінюється властивість об'єкта `Instance.background_color` випадковим чином за допомогою виклику `random.random()` для трьох перших аргументів (кольорів). При виконанні цього прикладу можна побачити, що при кожному натисканні колір кнопки випадково змінюється.

Результат виконання програми:

Кнопка також може обробляти події `on_release` (при відпусканні кнопи) і `state` (будь-яка зміні стану кнопки). Метод, який буде викликатися при події `state`, буде отримувати ще третій аргумент, стан кнопки. Кнопка може бути у двох станах – натиснутому (`down`) і (не натиснутому (`normal`).

Приклад програми:

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

```
import random

class FirstKivy(App):
   def btn_callback(self, instance, value):
      print(value)

   def build(self):
      self.btn=Button(text="Click!", pos=(300,350), size_hint = (.25, .18))
      self.btn.bind(state=self.btn_callback)
      return self.btn

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

В цьому прикладі при кожній зміні стану викликається метод `btn_callback` який виводитиме стан кнопки (параметр `value`) у консоль.

При роботі з формами, які мають багато полів введення, виникає необхідність визначення активності цих полів. Для цього поля введення обробляють події `text` і `focus`.

Подія `focus` виникає при активації поля введення користувачем. У прив'язаний метод подія передає параметри `self`, `instance` і `value`. Параметр `value` може мати два значення – `True`, якщо поле у фокусі і `False` в іншому випадку.

```
self.txt_in.bind(focus=self.focus_callback)
```

Подія `text` виникає при зміні вмісту поля введення. Подія передає параметри `self`, `instance` і `value`, де `value` – вміст поля введення.

```
self.txt_in.bind(text=self.txt_callback)
```

Приклад програми:

```
from kivy.app import App
from kivy.uix.textinput import TextInput
from kivy.uix.label import Label
from kivy.uix.floatlayout import FloatLayout

class FirstKivy(App):
   def txt_callback(self, instance, value):
       self.label1.text=value

   def focus_callback(self, instance, value):
       if value:
           self.label2.text="Focused"
       else:
           self.label2.text="Not focused"

   def build(self):
      self.layout = FloatLayout()
      self.txt_in=TextInput(hint_text='Write here', size_hint=(.4,.1),
pos=(250,250))
      self.label1=Label(text="Start typing...",pos=(15,75),font_size=25)
      self.label2=Label(text="Focus state",pos=(15,-75),font_size=25)
      self.txt_in.bind(text=self.txt_callback)
      self.txt_in.bind(focus=self.focus_callback)
      self.layout.add_widget(self.txt_in)
      self.layout.add_widget(self.label1)
      self.layout.add_widget(self.label2)
      return self.layout

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

У цьому прикладі створено поле введення та два надписи, які розміщені зверху і знизу поля введення. Для поля введення визначені події `text` і `focus`.

Подія `text` прив'язана до методу `txt_callback`, який присвоює вміст поля введення верхньому надпису. Оскільки подія виконується при кожній зміні вмісту, то верхній надпис дублюватиме все, що вводиться в полі введення.

Подія `focus` прив'язана до методу `focus_callback`, який присвоює нижньому надпису значення "Focused" або "Not focused" в залежності від фокусу поля.

Результат виконання програми:

Надписи також мають свого роду подію яка дозволяє зробити надписи інтерактивними. Для цього використовуються посилання в надписі, задані як стиль тексту. Текст виділений посиланням буде реагувати на натисканні на ньому. Посилання в надписі задаються тегом `[ref=variable]…[/ref]`.

```
widget = Label(text='Hello [ref=world]World[/ref]', markup=True)
widget.bind(on_ref_press=ref_click)
```

Тут в надписі визначене посилання, яке закріплене за словом `World`, а до самого надпису за подією `on_ref_press` прив'язаний метод `ref_click`. Подія передаватиме в метод `ref_click` параметри `self`, `instance` і `value` яке буде дорівнюватиме змінній яка задана в тегу `ref=world`.

Приклад програми:

```
from kivy.app import App
from kivy.uix.textinput import TextInput
from kivy.uix.label import Label
from kivy.uix.floatlayout import FloatLayout

class FirstKivy(App):

    def ref_click(self, instance, value):
        self.label2.text="Clicked on {0} word".format(value)

    def build(self):
        self.layout = FloatLayout()

        self.label1=Label(text='[ref=1]Hello[/ref]          [ref=2]World![/ref]
[ref=3]From[/ref] [ref=4]Kivy[/ref]', markup=True ,pos=(15,75),font_size=25)
        self.label2=Label(pos=(15,-75),font_size=25)

        self.label1.bind(on_ref_press=self.ref_click)

        self.layout.add_widget(self.label1)
        self.layout.add_widget(self.label2)
        return self.layout

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

У цьому прикладі є два надписи. В першому надписі є текст “Hello World! From Kivy!”, але кожне слово обгорнуте в посилання і в кожному посиланні змінній присвоєно номер слова. Для першому надпису визначена подія `on_ref_press` і до неї прив’язаний метод `ref_click`. Метод `ref_click` присвоює надпису “Clicked on {0} word”, де за допомогою функції `format()` замість {0} вставляється `value` яке відображає номер натиснутого слова.

Результат виконання програми:

10. Підтримка дат і часу в Kivy

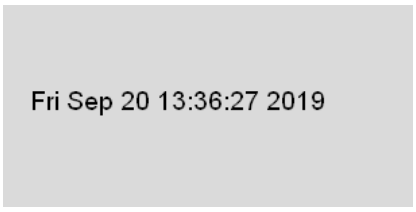
Об’єкт Kivy Clock використовується для роботи з датами і часом. Для імпортування Kivy Clock потрібно включити наступний рядок.

```
from kivy.clock import Clock
```

Приклад програми простого годинника:

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

import timr

class IncrediblyCrudeClock(Label):
   def update(self, *args):
      self.text = time.asctime()

class TimeApp(App):
   def build(self):
      crudeclock = IncrediblyCrudeClock()
      Clock.schedule_interval(crydeclock.update, 1)
      Return crudeclock

If __name__ = ”__main__”
   TimeApp().run()
```

Результат виконання програми:



Підтримка часу в Kivy дозволяє викликати функції з певним інтервалом часу. Для цього ініціалізується об’єкт `Clock.schedule_interval(self.Clock_Callback, 1)`, де перший

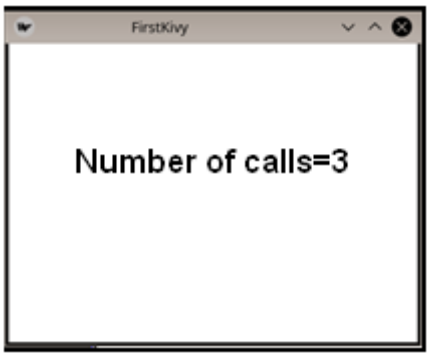
параметр визначає метод який буде викликатися, а другий – інтервал у секундах з яким буде викликатися метод.

Приклад програми, яка з інтервалом в одну секунду викликає метод, який при кожному виклику змінює надпис.

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

class FirstKivy(App):

    i=0

    def build(self):
        self.label1=Label(text='Number of calls=0')
        Clock.schedule_interval(self.Clock_Callback, 1)
        return self.label1

    def Clock_Callback(self, dt):
         self.i = self.i+1
         self.label1.text = "Number of calls={0}".format(self.i)

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

У класі `FirstKivy` додана змінна “i” яка буде лічильником. Імпортований та визначений Clock викликає метод `Clock_Callback` кожну секунду. Метод `Clock_Callback` збільшує змінну “i” на одиницю і змінює текст надпису на “Number of calls=”, в який підставляє значення змінної “i”.

Результат виконання програми:



Висновки.

Бібліотека Kivy дозволяє створювати програми з сучасним графічним інтерфейсом засобами Python і Kivy майже для всіх платформ.

Література.

1. Лутц М. Изучаем Python, 4-издание. Пер. С англ. СПб.: Символ-Плюс, 2011. – 1280 с.
2. https://kivy.org
3. https://likegeeks.com/kivy-tutorial/

Завдання

1. Створити програму з полем введення і кнопкою під нею, задаючи параметр `pos` для елементів.
2. Створити кнопку з розмірами 100x100 пікселів.
3. Створити однорядкове поле введення.
4. Створити надпис в якому три слова мають три різні кольори.

5. Створити кнопку червоного кольору.

6. Створити поле вводу і під ним кнопку за допомогою BoxLayout.

7. Впорядкувати 6 кнопок сіткою з розмірами 3×2.

8. Помістити елемент на середині лівого краю і зліва нижнього, за допомогою AnchorLayout.

9. Впорядкувати декілька елементів кнопок у вертикальний ряд.

10. Створити випадаючий список елементів.

11. Створити поле введення і кнопку. При натисканні кнопки вміст поля має очищуватися.

12. Створити поле введення і кнопку. Коли фокус у полі введення, кнопка повинна бути в активному стані. В іншому випадку кнопка переходить в неактивний стан.

13. Створити два надписи. Перший матиме декілька слів. В залежності на яке слово натиснув користувач на другому надписі повинен відображатися невеликий текст.

14. Створити надпис і дві кнопки. При натисканні першої кнопки шрифт надпису повинен збільшуватись, а при натисканні іншої – зменшуватись.

15. Створити функцію яка буде змінювати надпис при її викликах 1 раз в секунду.